



# **Intel® Integrated Performance Primitives**

**Reference Manual, Volume 2: Image and Video Processing**

***IPP 7.0***

Document Number: A70805-033US

Legal Information



# Contents

<b>Legal Information.....</b>	<b>29</b>
<b>Chapter 1: Volume Overview</b>	
What's New.....	31
Notational Conventions.....	31
<b>Chapter 2: Intel® Integrated Performance Primitives Concepts</b>	
Function Naming.....	33
Data-Domain.....	33
Name.....	33
Data Types.....	34
Descriptors.....	35
Parameters.....	36
Function Prototypes in Intel IPP.....	36
Rounding Mode.....	37
Integer Result Scaling.....	38
Error Reporting.....	38
Structures and Enumerators.....	41
Function Context Structures.....	46
Image Data Types and Ranges.....	47
Major Operation Models.....	48
Neighborhood Operations.....	48
Regions of Interest in Intel IPP.....	48
Tiled Image Processing.....	51
<b>Chapter 3: Support Functions</b>	
Version Information Function.....	53
GetLibVersion.....	53
Status Information Function.....	54
ippGetStatusString.....	55
Memory Allocation Functions.....	55
Malloc.....	56
Free.....	57

## Chapter 4: Image Data Exchange and Initialization Functions

Convert.....	60
Scale.....	64
Set.....	67
Copy.....	69
CopyManaged.....	72
CopyConstBorder.....	74
CopyReplicateBorder.....	76
CopyWrapBorder.....	79
CopySubpix.....	82
CopySubpixIntersect.....	83
Dup.....	86
Transpose.....	86
SwapChannels.....	88
AddRandUniform_Direct.....	91
AddRandGauss_Direct.....	92
ImageJaehne.....	93
ImageRamp.....	95
SampleLine.....	96
ZigzagFwd8x8.....	97
ZigzagInv8x8.....	99
Row Oriented Resampling.....	99
ResampleRowGetSize.....	100
ResampleRowInit.....	101
ResampleRowGetBorderWidth.....	102
ResampleRow, ResampleRowReplicateBorder.....	102

## Chapter 5: Image Arithmetic and Logical Operations

Arithmetic Operations.....	107
Add.....	107
AddC.....	111
AddSquare.....	114
AddProduct.....	115
AddWeighted.....	116
Mul.....	118
MulC.....	120
MulScale.....	123
MulCScale.....	124
Sub.....	127
SubC.....	129



Div.....	132
Div_Round.....	135
DivC.....	137
Abs.....	140
AbsDiff.....	141
AbsDiffC.....	142
Sqr.....	143
Sqrt.....	145
Ln.....	147
Exp.....	149
Complement.....	151
DotProd.....	151
DotProdCol.....	153
Logical Operations.....	153
And.....	154
AndC.....	155
Or.....	157
OrC.....	159
Xor.....	161
XorC.....	162
Not.....	164
RShiftC.....	165
LShiftC.....	167
Alpha Composition.....	170
AlphaComp.....	171
AlphaCompC.....	173
AlphaPremul.....	175
AlphaPremulC.....	176

## Chapter 6: Image Color Conversion

Gamma Correction.....	186
CIE Chromaticity Diagram and Color Gamut.....	186
Color Models.....	187
Image Downsampling.....	198
RGB Image Formats.....	199
Pixel and Planar Image Formats.....	199
Color Model Conversion.....	203
RGBToYUV.....	203
YUVToRGB.....	205
RGBToYUV422.....	206
YUV422ToRGB.....	207

RGB565ToYUV422.....	209
RGBToYUV420 .....	210
YUV420ToRGB.....	211
BGRToYUV420.....	212
YUV420ToBGR.....	213
RGB565ToYUV420.....	214
YUV420ToRGB565, YUV420ToRGB555, YUV420ToRGB444.....	215
YUV420ToRGB565Dither, YUV420ToRGB555Dither, YUV420ToRGB444Dither.....	216
BGR565ToYUV420, BGR555ToYUV420.....	217
YUV420ToBGR565, YUV420ToBGR555, YUV420ToBGR444.....	218
YUV420ToBGR565Dither, YUV420ToBGR555Dither, YUV420ToBGR444Dither.....	219
RGBToYCbCr.....	220
YCbCrToRGB.....	221
YCbCrToBGR.....	222
YCbCrToBGR_709CSC.....	223
YCbCrToRGB565, YCbCrToRGB555, YCbCrToRGB444.....	224
YCbCrToRGB565Dither, YCbCrToRGB555Dither, YCbCrToRGB444Dither....	225
YCbCrToBGR565, YCbCrToBGR555, YCbCrToBGR444.....	226
YCbCrToBGR565Dither, YCbCrToBGR555Dither, YCbCrToBGR444Dither....	228
RGBToYCbCr422.....	229
YCbCr422ToRGB.....	230
RGBToYCrCb422.....	231
YCrCb422ToRGB.....	232
BGRToYCbCr422.....	233
YCbCr422ToBGR.....	234
BGR565ToYCbCr422, BGR555ToYCbCr422.....	235
RGBToCbYCr422, RGBToCbYCr422Gamma.....	236
CbYCr422ToRGB.....	237
BGRToCbYCr422.....	237
BGRToCbYCr422_709HDTV.....	238
CbYCr422ToBGR.....	239
CbYCr422ToBGR_709HDTV.....	240
YCbCr422ToRGB565, YCbCr422ToRGB555, YCbCr422ToRGB444.....	241
YCbCr422ToRGB565Dither, YCbCr422ToRGB555Dither, YCbCr422ToRGB444Dither.....	242
YCbCr422ToBGR565, YCbCr422ToBGR555, YCbCr422ToBGR444.....	244
YCbCr422ToBGR565Dither, YCbCr422ToBGR555Dither, YCbCr422ToBGR444Dither.....	245
RGBToYCbCr420.....	246

---

YCbCr420ToRGB.....	247
YCbCr420ToRGB565, YCbCr420ToRGB555, YCbCr420ToRGB444.....	248
YCbCr420ToRGB565Dither, YCbCr420ToRGB555Dither, YCbCr420ToRGB444Dither.....	249
RGBToYCrCb420.....	250
YCrCb420ToRGB.....	251
BGRToYCbCr420.....	252
BGRToYCbCr420_709CSC.....	253
BGRToYCbCr420_709HDTV.....	254
BGRToYCrCb420_709CSC.....	255
YCbCr420ToBGR.....	256
YCbCr420ToBGR_709CSC.....	257
YCbCr420ToBGR_709HDTV.....	258
BGR565ToYCbCr420, BGR555ToYCbCr420.....	260
YCbCr420ToBGR565, YCbCr420ToBGR555, YCbCr420ToBGR444.....	261
YCbCr420ToBGR565Dither, YCbCr420ToBGR555Dither, YCbCr420ToBGR444Dither.....	262
BGRToYCrCb420.....	263
BGR565ToYCrCb420, BGR555ToYCrCb420.....	263
BGRToYCbCr411.....	264
YCbCr411ToBGR.....	265
BGR565ToYCbCr411, BGR555ToYCbCr411.....	266
YCbCr411ToBGR565, YCbCr411ToBGR555.....	267
RGBToXYZ.....	268
XYZToRGB.....	269
RGBToLUV.....	270
LUVToRGB.....	272
BGRToLab.....	273
LabToBGR.....	275
RGBToYCC.....	276
YCCToRGB.....	277
RGBToHLS.....	278
HLSToRGB.....	279
BGRToHLS.....	281
HLSToBGR.....	282
RGBToHSV.....	283
HSVToRGB.....	284
RGBToYCoCg.....	285
YCoCgToRGB.....	286
BGRToYCoCg.....	287
SBGRToYCoCg.....	287

YCoCgToBGR.....	288
YCoCgToSBGR.....	289
BGRToYCoCg_Rev.....	290
SBGRToYCoCg_Rev.....	291
YCoCgToBGR_Rev.....	292
YCoCgToSBGR_Rev .....	293
Color - Gray Scale Conversions.....	294
RGBToGray.....	294
ColorToGray.....	295
CFAToRGB.....	296
DemosaicAHD.....	298
Format Conversion .....	299
RGBToRGB565, BGRToBGR565.....	299
RGB565ToRGB, BGR565ToBGR.....	300
YCbCr422.....	301
YCbCr422ToYCrCb422.....	302
YCbCr422ToCbYCr422.....	302
YCbCr422ToYCbCr420.....	303
YCbCr422To420_Interlace.....	305
YCbCr422ToYCrCb420.....	306
YCbCr422ToYCbCr411.....	307
YCrCb422ToYCbCr422.....	308
YCrCb422ToYCbCr420.....	309
YCrCb422ToYCbCr411.....	310
CbYCr422ToYCbCr422.....	310
CbYCr422ToYCbCr420.....	311
CbYCr422ToYCbCr420_Interlace.....	312
CbYCr422ToYCrCb420.....	313
CbYCr422ToYCbCr411.....	314
YCbCr420.....	315
YCbCr420ToYCbCr422.....	316
YCbCr420ToYCbCr422_Filter.....	317
YCbCr420To422_Interlace.....	319
YCbCr420ToCbYCr422.....	320
YCbCr420ToCbYCr422_Interlace.....	320
YCbCr420ToYCrCb420.....	321
YCbCr420ToYCrCb420_Filter.....	322
YCbCr420ToYCbCr411.....	324
YCrCb420ToYCbCr422.....	325
YCrCb420ToYCbCr422_Filter.....	326
YCrCb420ToCbYCr422.....	327

YCrCb420ToYCbCr420.....	327
YCrCb420ToYCbCr411.....	328
YCbCr411.....	329
YCbCr411ToYCbCr422.....	330
YCbCr411ToYCrCb422.....	331
YCbCr411ToYCbCr420.....	332
YCbCr411ToYCrCb420.....	334
Color Twist.....	334
ColorTwist.....	335
ColorTwist32f.....	336
Color Keying.....	338
CompColorKey.....	338
AlphaCompColorKey.....	340
Gamma Correction.....	341
GammaFwd.....	341
GammaInv.....	343
Intensity Transformation.....	346
ReduceBits.....	346
LUT.....	348
LUT_Linear.....	351
LUT_Cubic.....	355
LUTPalette, LUTPaletteSwap.....	358
ToneMapLinear, ToneMapMean.....	361

## **Chapter 7: Threshold and Compare Operations**

Thresholding.....	364
Threshold.....	364
Threshold_GT.....	366
Threshold_LT.....	368
Threshold_Val.....	370
Threshold_GTVal.....	372
Threshold_LTVal.....	374
Threshold_LTValGTVal.....	376
ComputeThreshold_Otsu.....	378
Compare Operations.....	380
Compare.....	381
CompareC.....	382
CompareEqualEps.....	383
CompareEqualEpsC.....	384

## **Chapter 8: Morphological Operations**

Dilate3x3.....	393
Erode3x3.....	394
Dilate.....	396
Erode.....	397
MorphologyInitAlloc.....	399
MorphologyFree.....	400
MorphologyInit.....	400
MorphologyGetSize.....	401
DilateBorderReplicate.....	402
ErodeBorderReplicate.....	404
MorphAdvInitAlloc.....	405
MorphAdvFree.....	406
MorphAdvInit.....	407
MorphAdvGetSize.....	408
MorphOpenBorder.....	408
MorphCloseBorder.....	410
MorphTophatBorder.....	411
MorphBlackhatBorder.....	412
MorphGradientBorder.....	413
MorphGrayInitAlloc.....	414
MorphGrayFree.....	415
MorphGrayInit.....	416
MorphGrayGetSize.....	417
GrayDilateBorder.....	417
GrayErodeBorder.....	418
MorphReconstructGetBufferSize.....	420
MorphReconstructDilate.....	420
MorphReconstructErode.....	423

## **Chapter 9: Filtering Functions**

Borders.....	430
FilterBox.....	432
SumWindowRow.....	433
SumWindowColumn.....	434
FilterMin.....	435
FilterMax.....	437
FilterMinGetBufferSize.....	438
FilterMaxGetBufferSize.....	439
FilterMinBorderReplicate.....	439
FilterMaxBorderReplicate.....	441
FilterBilateralGetBufSize.....	442

---

FilterBilateralInit.....	443
FilterBilateral.....	444
DecimateFilterRow, DecimateFilterColumn.....	447
Median Filters.....	448
FilterMedianGetBufferSize.....	449
FilterMedian.....	450
FilterMedianHoriz.....	452
FilterMedianVert.....	453
FilterMedianCross.....	454
FilterMedianWeightedCenter3x3.....	455
FilterMedianColor.....	456
General Linear Filters.....	457
Filter.....	458
FilterGetBufSize.....	460
Filter32f.....	460
Filter_Round.....	462
FilterRoundGetBufSize.....	464
Separable Filters.....	465
FilterColumn.....	466
FilterColumn32f.....	468
FilterRow.....	469
FilterRow32f.....	471
FilterRowBorderPipelineGetBufferSize, FilterRowBorderPipelineGetBufferSize_Low.....	472
FilterColumnPipelineGetBufferSize, FilterColumnPipelineGetBufferSize_Low.....	473
FilterRowBorderPipeline, FilterRowBorderPipeline_Low.....	474
FilterColumnPipeline, FilterColumnPipeline_Low.....	476
DotProdCol.....	479
Wiener Filters.....	480
FilterWienerGetBufferSize.....	480
FilterWiener.....	481
Convolution.....	484
ConvFull.....	485
ConvValid.....	487
Deconvolution.....	490
DeconvFFTInitAlloc.....	490
DeconvFFTFree.....	491
DeconvFFT.....	492
DeconvLRInitAlloc.....	492
DeconvLRFree.....	493

DeconvLR.....	494
Fixed Filters.....	495
FilterPrewittHoriz.....	495
FilterPrewittVert.....	496
FilterScharrHoriz.....	498
FilterScharrVert.....	499
FilterSobelHoriz, FilterSobelHorizMask.....	500
FilterSobelVert, FilterSobelVertMask.....	502
FilterSobelHorizSecond.....	503
FilterSobelVertSecond.....	504
FilterSobelCross.....	507
FilterRobertsDown.....	508
FilterRobertsUp.....	509
FilterLaplace.....	510
FilterGauss.....	511
FilterHipass.....	515
FilterLowpass.....	516
FilterSharpen.....	517
Fixed Filters with Border .....	518
FilterScharrHorizGetBufferSize.....	518
FilterScharrVertGetBufferSize.....	519
FilterSobelHorizGetBufferSize.....	520
FilterSobelVertGetBufferSize, FilterSobelNegVertGetBufferSize.....	520
FilterSobelHorizSecondGetBufferSize.....	521
FilterSobelVertSecondGetBufferSize.....	522
FilterSobelCrossGetBufferSize.....	523
FilterLaplacianGetBufferSize.....	523
FilterGaussGetBufferSize.....	524
FilterLowpassGetBufferSize.....	525
GenSobelKernel.....	525
FilterScharrHorizBorder.....	526
FilterScharrVertBorder.....	528
FilterSobelHorizBorder.....	529
FilterSobelVertBorder, FilterSobelNegVertBorder.....	531
FilterSobelHorizSecondBorder.....	533
FilterSobelVertSecondBorder.....	534
FilterSobelCrossBorder.....	537
FilterLaplacianBorder.....	538
FilterGaussBorder.....	540
FilterLowpassBorder.....	541



---

**Chapter 10: Image Linear Transforms**

Fourier Transforms.....	546
Real - Complex Packed (RCPack2D) Format.....	547
FFTInitAlloc.....	548
FFTFree.....	549
FFTGetBufSize.....	550
FFTFwd.....	550
FFTInv.....	554
DFTInitAlloc.....	556
DFTFree.....	557
DFTGetBufSize.....	557
DFTFwd.....	558
DFTInv.....	560
MulPack.....	562
MulPackConj.....	566
Magnitude.....	567
MagnitudePack.....	568
Phase.....	570
PhasePack.....	571
PolarToCart.....	572
PackToCplxExtend.....	573
CplxExtendToPack.....	574
Windowing Functions.....	575
WinBartlett, WinBartlettSep.....	576
WinHamming, WinHammingSep.....	577
Discrete Cosine Transforms.....	579
DCTFwdInitAlloc.....	579
DCTInvInitAlloc.....	580
DCTFwdFree.....	581
DCTInvFree.....	581
DCTFwdGetBufSize.....	582
DCTInvGetBufSize.....	583
DCTFwd.....	583
DCTInv.....	585
DCT8x8Fwd.....	586
DCT8x8Inv, DCT8x8Inv_A10.....	588
DCT8x8FwdLS.....	589
DCT8x8InvLSClip.....	590
DCT8x8Inv_2x2, DCT8x8Inv_4x4.....	590
DCT8x8To2x2Inv, DCT8x8To4x4Inv.....	591

## Chapter 11: Image Statistics Functions

Sum.....	595
Integral.....	597
SqrIntegral.....	599
TiltedIntegral.....	601
TiltedSqrIntegral.....	602
Mean.....	603
Mean_StdDev.....	606
RectStdDev.....	608
TiltedRectStdDev.....	609
HistogramRange.....	611
HistogramEven.....	613
CountInRange.....	615
Min.....	616
MinIndx.....	618
Max.....	619
MaxIndx.....	621
MinMax.....	622
MinMaxIndx.....	624
MaxEvery.....	626
MinEvery.....	627
FindPeaks3x3GetBufferSize.....	628
FindPeaks3x3.....	628
Image Moments.....	632
MomentInitAlloc.....	633
MomentFree.....	634
MomentGetStateSize.....	634
MomentInit.....	635
Moments.....	635
GetSpatialMoment.....	637
GetCentralMoment.....	638
GetNormalizedSpatialMoment.....	638
GetNormalizedCentralMoment.....	639
GetHuMoments.....	640
Image Norms.....	641
Norm_Inf.....	642
Norm_L1.....	644
Norm_L2.....	646
NormDiff_Inf.....	649
NormDiff_L1.....	651

NormDiff_L2.....	654
NormRel_Inf.....	656
NormRel_L1.....	658
NormRel_L2.....	661
Image Quality Index.....	664
QualityIndex.....	665
Image Proximity Measures.....	668
SqrDistanceFull_Norm.....	670
SqrDistanceSame_Norm.....	673
SqrDistanceValid_Norm.....	675
CrossCorrFull_Norm.....	676
CrossCorrSame_Norm.....	679
CrossCorrValid_Norm.....	682
CrossCorrValid.....	683
CrossCorrFull_NormLevel.....	685
CrossCorrSame_NormLevel.....	686
CrossCorrValid_NormLevel.....	690

## Chapter 12: Image Geometry Transforms

ROI Processing in Geometric Transforms.....	695
Geometric Transform Functions.....	697
Resize.....	697
ResizeCenter.....	701
ResizeSqrPixel.....	705
ResizeGetBufSize.....	712
GetResizeRect.....	713
ResizeSqrPixelGetBufSize.....	714
GetResizeFract.....	715
ResizeShift.....	716
SuperSampling.....	718
SuperSamplingGetBufSize.....	720
ResizeYUV422.....	720
ResizeYUV420.....	722
ResizeYUV420GetBufSize.....	723
ResizeFilterGetSize.....	724
ResizeFilterInit.....	724
ResizeFilter.....	725
Mirror.....	726
Remap.....	728
Rotate.....	733
GetRotateShift.....	736

AddRotateShift.....	737
GetRotateQuad.....	737
GetRotateBound.....	738
RotateCenter.....	741
Shear.....	745
GetShearQuad.....	749
GetShearBound.....	750
WarpAffine.....	751
WarpAffineBack.....	754
WarpAffineQuad.....	756
GetAffineQuad.....	758
GetAffineBound.....	759
GetAffineTransform.....	760
WarpPerspective.....	763
WarpPerspectiveBack.....	766
WarpPerspectiveQuad.....	768
GetPerspectiveQuad.....	770
GetPerspectiveBound.....	771
GetPerspectiveTransform .....	771
WarpBilinear.....	775
WarpBilinearBack.....	778
WarpBilinearQuad.....	780
GetBilinearQuad.....	782
GetBilinearBound.....	783
GetBilinearTransform .....	784

## **Chapter 13: Wavelet Transforms**

WTFwdInitAlloc.....	793
WTFwdFree.....	794
WTFwdGetBufSize.....	795
WTFwd.....	795
WTInvInitAlloc.....	799
WTInvFree.....	800
WTInvGetBufSize.....	801
WTInv.....	802

## **Chapter 14: Computer Vision**

Using ippiAdd for Background Differencing.....	811
Feature Detection Functions.....	812
Corner Detection.....	812
Canny Edge Detector.....	813

---

CannyGetSize .....	814
Canny .....	815
EigenValsVecsGetBufferSize .....	816
EigenValsVecs .....	817
MinEigenValGetBufferSize .....	820
MinEigenVal .....	821
Hough Transform.....	823
HoughLineGetSize .....	823
HoughLine .....	824
HoughLine_Region .....	825
Distance Transform Functions.....	826
DistanceTransform.....	826
GetDistanceTransformMask .....	829
TrueDistanceTransformGetBufferSize.....	831
TrueDistanceTransform.....	832
FastMarchingGetBufferSize.....	833
FastMarching.....	833
Image Gradients.....	835
GradientColorToGray .....	835
Flood Fill Functions.....	836
FloodFillGetSize .....	837
FloodFillGetSize_Grad .....	838
FloodFill.....	838
FloodFill_Grad.....	840
FloodFill_Range.....	843
Motion Analysis and Object Tracking.....	845
Motion Template Functions.....	845
Motion Representation.....	845
Updating MHI Images.....	846
UpdateMotionHistory .....	846
Optical Flow .....	847
OpticalFlowPyrLKInitAlloc .....	848
OpticalFlowPyrLKFree .....	849
OpticalFlowPyrLK .....	849
Pyramids Functions.....	853
PyrDownGetBufSize .....	855
PyrUpGetBufSize .....	855
PyrDown .....	856
PyrUp .....	857
Universal Pyramids .....	858
PyramidInitAlloc .....	859

PyramidFree .....	860
PyramidLayerDownInitAlloc .....	860
PyramidLayerDownFree .....	862
PyramidLayerUpInitAlloc .....	862
PyramidLayerUpFree .....	864
GetPyramidDownROI.....	864
GetPyramidUpROI .....	865
PyramidLayerDown .....	866
PyramidLayerUp .....	868
Example of Using General Pyramid Functions.....	869
Image Inpainting.....	871
InpaintInitAlloc .....	871
InpaintFree .....	873
Inpaint .....	873
Image Segmentation .....	875
LabelMarkersGetBufferSize .....	875
LabelMarkers .....	875
SegmentWatershedGetBufferSize .....	877
SegmentWatershed .....	877
SegmentGradientGetBufferSize .....	880
SegmentGradient .....	881
BoundSegments .....	882
ForegroundHistogramInitAlloc .....	883
ForegroundHistogramFree .....	885
ForegroundHistogram .....	885
ForegroundHistogramUpdate .....	888
ForegroundGaussianInitAlloc .....	889
ForegroundGaussianFree .....	890
ForegroundGaussian .....	891
Pattern Recognition .....	893
Object Detection Using Haar-like Features .....	893
HaarClassifierInitAlloc .....	894
TiltedHaarClassifierInitAlloc .....	895
HaarClassifierFree .....	896
GetHaarClassifierSize .....	897
TiltHaarFeatures .....	897
ApplyHaarClassifier .....	898
ApplyMixedHaarClassifier .....	900
Camera Calibration and 3D Reconstruction.....	901
Correction of Camera Lens Distortion.....	901
UndistortGetSize .....	902

UndistortRadial .....	903
CreateMapCameraUndistort .....	904
Image Enhancement Functions.....	906
Algorithm Overview .....	906
Point Spread Function.....	907
Error Functions .....	908
SRHNInit Alloc_PSF3x3, SRHNInitAlloc_PSF2x2.....	909
SRHNFree_PSF3x3, SRHNFree_PSF2x2.....	909
SRHNCalcResidual_PSF3x3, SRHNCalcResidual_PSF2x2.....	910
SRHNUpdateGradient_PSF3x3, SRHNUpdateGradient_PSF2x2.....	911

## Chapter 15: Image Compression Functions

Support Functions.....	914
ippjGetLibVersion.....	914
Color Conversion Functions.....	914
RGBToY_JPEG.....	915
BGRToY_JPEG.....	917
RGBToYCbCr_JPEG.....	918
YCbCrToRGB_JPEG.....	919
RGB565ToYCbCr_JPEG, RGB555ToYCbCr_JPEG.....	920
YCbCrToRGB565_JPEG, YCbCrToRGB555_JPEG.....	921
BGRToYCbCr_JPEG.....	921
YCbCrToBGR_JPEG.....	922
BGR565ToYCbCr_JPEG, BGR555ToYCbCr_JPEG.....	923
YCbCrToBGR565_JPEG, YCbCrToBGR555_JPEG .....	924
RGBToYCbCr422_JPEG.....	925
YCbCr422ToRGB_JPEG.....	926
BGRToYCbCr422_JPEG.....	927
YCbCr422ToBGR_JPEG.....	928
RGBToYCbCr411_JPEG.....	928
YCbCr4112ToRGB_JPEG.....	929
BGRToYCbCr411_JPEG.....	930
YCbCr4112ToBGR_JPEG.....	931
CMYKToYCKK_JPEG.....	932
YCKKToCMYK_JPEG.....	933
Combined Color Conversion Functions.....	934
RGBToYCbCr444LS_MCU.....	935
RGBToYCbCr422LS_MCU.....	936
RGBToYCbCr411LS_MCU.....	938
BGRToYCbCr444LS_MCU.....	939
BGR565ToYCbCr444LS_MCU, BGR555ToYCbCr444LS_MCU.....	939

BGRToYCbCr422LS_MCU.....	940
BGR565ToYCbCr422LS_MCU, BGR555ToYCbCr422LS_MCU.....	941
BGRToYCbCr411LS_MCU.....	942
GR565ToYCbCr411LS_MCU, BGR555ToYCbCr411LS_MCU.....	943
CMYKToYCKK444LS_MCU.....	943
CMYKToYCKK422LS_MCU.....	944
CMYKToYCKK411LS_MCU.....	945
YCbCr444ToRGBLS_MCU.....	945
YCbCr422ToRGBLS_MCU.....	946
YCbCr411ToRGBLS_MCU.....	947
YCbCr444ToBGRLS_MCU.....	947
YCbCr444ToBGR565LS_MCU, YCbCr444ToBGR555LS_MCU.....	948
YCbCr422ToBGRLS_MCU.....	949
YCbCr422ToBGR565LS_MCU, YCbCr422ToBGR555LS_MCU.....	950
YCbCr411ToBGRLS_MCU.....	950
YCbCr411ToBGR565LS_MCU, YCbCr411ToBGR555LS_MCU.....	951
YCKK444ToCMYKLS_MCU.....	952
YCKK422ToCMYKLS_MCU.....	953
YCKK411ToCMYKLS_MCU.....	953
Quantization Functions.....	954
QuantFwdRawTableInit_JPEG.....	954
QuantFwdTableInit_JPEG.....	955
QuantFwd8x8_JPEG.....	955
QuantInvTableInit_JPEG.....	958
QuantInv8x8_JPEG.....	959
Combined DCT Functions.....	959
DCTQuantFwd8x8_JPEG.....	960
DCTQuantFwd8x8LS_JPEG.....	961
DCTQuantInv8x8_JPEG.....	963
DCTQuantInv8x8LS_JPEG.....	964
DCTQuantInv8x8LS_1x1_JPEG, DCTQuantInv8x8LS_2x2_JPEG, DCTQuantInv8x8LS_4x4_JPEG.....	965
DCTQuantInv8x8To4x4LS_JPEG, DCTQuantInv8x8To2x2LS_JPEG.....	966
Level Shift Functions.....	966
Sub128_JPEG.....	967
Add128_JPEG.....	967
Sampling Functions.....	968
SampleDownH2V1_JPEG.....	969
SampleDownH2V2_JPEG.....	970
SampleDownRowH2V1_Box_JPEG.....	970
SampleDownRowH2V2_Box_JPEG.....	973



---

SampleUpH2V1_JPEG.....	973
SampleUpH2V2_JPEG.....	975
SampleUpRowH2V1_Triangle_JPEG.....	976
SampleUpRowH2V2_Triangle_JPEG.....	976
SampleDown444LS_MCU.....	977
SampleDown422LS_MCU.....	978
SampleDown411LS_MCU.....	978
SampleUp444LS_MCU.....	979
SampleUp422LS_MCU.....	980
SampleUp411LS_MCU.....	980
Planar-to-Pixel and Pixel-to-Planar Conversion Functions.....	981
Split422LS_MCU.....	981
Join422LS_MCU.....	982
Huffman Codec Functions.....	983
EncodeHuffmanRawTableInit_JPEG.....	984
EncodeHuffmanSpecGetBufSize_JPEG.....	985
EncodeHuffmanSpecInit_JPEG.....	986
EncodeHuffmanSpecInitAlloc_JPEG.....	986
EncodeHuffmanSpecFree_JPEG.....	987
EncodeHuffmanStateGetBufSize_JPEG.....	987
EncodeHuffmanStateInit_JPEG.....	988
EncodeHuffmanStateInitAlloc_JPEG.....	988
EncodeHuffmanStateFree_JPEG.....	989
EncodeHuffman8x8_JPEG.....	989
EncodeHuffman8x8_Direct_JPEG.....	990
GetHuffmanStatistics8x8_JPEG.....	991
GetHuffmanStatistics8x8_DCFirst_JPEG.....	992
GetHuffmanStatistics8x8_ACFirst_JPEG.....	992
GetHuffmanStatistics8x8_ACRefine_JPEG.....	993
EncodeHuffman8x8_DCFirst_JPEG.....	994
EncodeHuffman8x8_DCRefine_JPEG.....	995
EncodeHuffman8x8_ACFirst_JPEG.....	996
EncodeHuffman8x8_ACRefine_JPEG.....	997
DecodeHuffmanSpecGetBufSize_JPEG.....	998
DecodeHuffmanSpecInit_JPEG.....	998
DecodeHuffmanSpecInitAlloc_JPEG.....	999
DecodeHuffmanSpecFree_JPEG.....	999
DecodeHuffmanStateGetBufSize_JPEG.....	1000
DecodeHuffmanStateInit_JPEG.....	1000
DecodeHuffmanStateInitAlloc_JPEG.....	1001
DecodeHuffmanStateFree_JPEG.....	1001

DecodeHuffman8x8_JPEG.....	1002
DecodeHuffman8x8_Direct_JPEG.....	1003
DecodeHuffman8x8_DCFirst_JPEG.....	1004
DecodeHuffman8x8_DCRefine_JPEG.....	1005
DecodeHuffman8x8_ACFirst_JPEG.....	1006
DecodeHuffman8x8_ACRefine_JPEG.....	1007
Functions for Lossless JPEG Coding.....	1008
DiffPredFirstRow_JPEG.....	1008
DiffPredRow_JPEG.....	1010
ReconstructPredFirstRow_JPEG.....	1011
ReconstructPredRow_JPEG.....	1011
GetHuffmanStatisticsOne_JPEG.....	1012
EncodeHuffmanOne_JPEG.....	1013
DecodeHuffmanOne_JPEG.....	1014
EncodeHuffmanRow_JPEG.....	1015
DecodeHuffmanRow_JPEG.....	1016
Wavelet Transform Functions.....	1017
Low-Level Operations.....	1018
WTFwdRow_B53_JPEG2K.....	1018
WTInvRow_B53_JPEG2K.....	1020
WTFwdCol_B53_JPEG2K.....	1021
WTFwdColLift_B53_JPEG2K.....	1022
WTInvCol_B53_JPEG2K.....	1023
WTInvColLift_B53_JPEG2K.....	1025
WTFwdRow_D97_JPEG2K.....	1026
WTInvRow_D97_JPEG2K.....	1027
WTFwdCol_D97_JPEG2K.....	1028
WTFwdColLift_D97_JPEG2K.....	1030
WTInvCol_D97_JPEG2K.....	1031
WTInvColLift_D97_JPEG2K.....	1032
Tile-oriented Transforms.....	1033
WTGetBufSize_B53_JPEG2K.....	1033
WTFwd_B53_JPEG2K.....	1034
WTInv_B53_JPEG2K.....	1036
WTGetBufSize_D97_JPEG2K.....	1037
WTFwd_D97_JPEG2K.....	1038
WTInv_D97_JPEG2K.....	1039
JPEG2000 Entropy Coding and Decoding Functions.....	1040
EncodeInitAlloc_JPEG2K.....	1041
EncodeFree_JPEG2K.....	1042
EncodeLoadCodeBlock_JPEG2K.....	1042

---

EncodeStoreBits_JPEG2K.....	1044
EncodeGetTermPassLen_JPEG2K.....	1045
EncodeGetRate_JPEG2K.....	1046
EncodeGetDist_JPEG2K.....	1047
DecodeGetBufSize_JPEG2K.....	1047
DecodeCodeBlock_JPEG2K.....	1048
DecodeCBProgrGetStateSize_JPEG2K.....	1049
DecodeCBProgrInit_JPEG2K.....	1050
DecodeCBProgrInitAlloc_JPEG2K.....	1050
DecodeCBProgrFree_JPEG2K.....	1051
DecodeCBProgrAttach_JPEG2K.....	1051
DecodeCBProgrSetPassCounter_JPEG2K.....	1052
DecodeCBProgrGetPassCounter_JPEG2K.....	1053
DecodeCBProgrGetCurBitPlane_JPEG2K.....	1053
DecodeCBProgrStep_JPEG2K.....	1054
Component Transform Functions.....	1054
RCTFwd_JPEG2K.....	1055
RCTInv_JPEG2K.....	1057
ICTFwd_JPEG2K.....	1059
ICTInv_JPEG2K.....	1060
Functions for RLE Coding.....	1061
RLE Coding Model.....	1061
PackBitsRow_TIFF.....	1062
UnpackBitsRow_TIFF.....	1063
SplitRow_TIFF.....	1064
JoinRow_TIFF.....	1064
Texture Compression Functions.....	1065
TextureEncodeBlockFromRGBA.....	1066
TextureEncodeBlockFromRG.....	1067
TextureEncodeBlock.....	1067
TextureEncodeBlock4x4FromRGBA.....	1068
TextureDecodeBlockToRGBA.....	1069
TextureDecodeBlockToRGB.....	1070
TextureDecodeBlockToRG.....	1070
TextureDecodeBlock.....	1071
TextureEncodeBlockFromYCoCg.....	1072
High Definition Photo Coding.....	1073
Photo Core Transform Functions.....	1073
PCTFwd_HDP.....	1073
PCTInv.....	1074
JPEG XR Coding.....	1075

Core Transform Functions.....	1076
PCTFwd_JPEGXR.....	1076
PCTInv_JPEGXR.....	1077
Adaptive Variable and Fixed Length Coding Functions.....	1078
VLCGetStateSize_JPEGXR.....	1078
VLCInit_JPEGXR.....	1079
VLCAdapt_JPEGXR.....	1079
VLCScanSet_JPEGXR, VLCScanReset.....	1080
VLCScan4x4DC_JPEGXR.....	1080
VLCEncode_JPEGXR.....	1081
VLCDecode_JPEGXR.....	1082
FLCEncode_JPEGXR.....	1083
FLCDecode_JPEGXR.....	1084

## Chapter 16: Video Coding

General Functions.....	1088
Structures and Enumerators.....	1094
UV Block.....	1096
Variable Length Decoding.....	1097
Using Subtables.....	1098
HuffmanTableInitAlloc.....	1103
HuffmanRunLevelTableInitAlloc.....	1104
HuffmanTableFree.....	1105
DecodeHuffmanOne.....	1105
DecodeHuffmanPair.....	1106
Motion Compensation.....	1106
Predicted Blocks.....	1107
Bi-Predicted Blocks.....	1117
Motion Estimation.....	1135
Difference Evaluation.....	1137
Sum of Squares of Differences Evaluation.....	1152
Block Variance and Mean Evaluation.....	1156
Evaluation of Variances and Means of Blocks of Difference Between Two Blocks.....	1157
Block Variance Evaluation.....	1159
Evaluation of Block Deviation.....	1160
Edges Detection.....	1162
SAD Functions.....	1163
Sum of Differences Evaluation.....	1183
Scanning Functions.....	1187
ScanInv .....	1187

---

ScanFwd .....	1188
Color Conversion .....	1189
CbYCr422ToYCbCr420_Rotate .....	1189
ResizeCCRotate .....	1191
Video Processing.....	1193
Deinterlacing Functions.....	1193
Denoising Functions.....	1206
MPEG-1 and MPEG-2.....	1213
Video Data Decoding.....	1217
Variable Length Decoding.....	1220
Inverse Quantization.....	1226
Inverse Discrete Cosine Transformation .....	1227
Motion Compensation.....	1231
Video Data Encoding.....	1232
Motion Estimation.....	1233
Quantization.....	1234
Forward Discrete Cosine Transformation .....	1236
Huffman Encoding Functions.....	1237
DV.....	1239
DCT Block.....	1240
DV Decoding Functions.....	1245
Variable Length Decoding.....	1245
Inverse Quantization.....	1257
Inverse Discrete Cosine Transformation.....	1265
DV Encoding Functions.....	1269
Discrete Cosine Transformation.....	1270
DV Color Conversion Functions.....	1272
YCrCb411ToYCbCr422_5MBDV,	
YCrCb411ToYCbCr422_ZoomOut2_5MBDV,	
YCrCb411ToYCbCr422_ZoomOut4_5MBDV,	
YCrCb411ToYCbCr422_ZoomOut8_5MBDV.....	1272
YCrCb411ToYCbCr422_16x4x5MB_DV,	
YCrCb411ToYCbCr422_8x8MB_DV.....	1273
YCrCb411ToYCbCr422_EdgeDV,	
YCrCb411ToYCbCr422_ZoomOut2_EdgeDV,	
CrCb411ToYCbCr422_ZoomOut4_EdgeDV,	
YCrCb411ToYCbCr422_ZoomOut8_EdgeDV.....	1274
YCrCb420ToYCbCr422_5MBDV,	
YCrCb420ToYCbCr422_ZoomOut2_5MBDV,	
YCrCb420ToYCbCr422_ZoomOut4_5MBDV,	
YCrCb420ToYCbCr422_ZoomOut8_5MBDV.....	1275

YCrCb420ToYCbCr422_8x8x5MB_DV.....	1276
YCrCb422ToYCbCr422_5MBDV, YCrCb422ToYCbCr422_ZoomOut2_5MBDV, YCrCb422ToYCbCr422_ZoomOut4_5MBDV, YCrCb422ToYCbCr422_ZoomOut8_5MBDV.....	1277
YCrCb422ToYCbCr422_8x4x5MB_DV.....	1279
YCrCb422ToYCbCr422_10HalvesMB16x8_DV100.....	1279
MPEG-4.....	1281
MPEG-4 Video Decoder Functions.....	1283
High Level Description.....	1284
Data Types and Structures.....	1285
Motion Compensation.....	1285
Sprite and Global Motion Compensation.....	1287
Inverse Quantization.....	1292
VLC Decoding.....	1294
Postprocessing.....	1300
MPEG-4 Video Encoder Functions.....	1303
Data Types and Structures.....	1303
Quantization.....	1304
VLC Encoding.....	1306
H.261.....	1309
H.261 Decoder Functions.....	1309
Decoding INTRA and INTER Macroblocks.....	1310
H.261 Encoder Functions.....	1316
EncodeCoeffsIntra_H261 .....	1316
EncodeCoeffsInter_H261 .....	1317
Filter8x8_H261 .....	1318
H.263.....	1319
H.263 Decoder Functions.....	1321
INTRA and INTER Macroblocks Decoding .....	1322
VLC Decoding.....	1323
Inverse Quantization.....	1326
Prediction.....	1329
Resampling.....	1330
Boundary Filtering.....	1335
Middle Level Functions.....	1337
H.263 Encoder Functions.....	1340
VLC Encoding.....	1340
Quantization.....	1343
Resampling.....	1345
H.264.....	1346

H.264 Decoder Functions.....	1356
CAVLC Parsing.....	1361
Inverse Quantization and Inverse Transform.....	1364
Intra Prediction.....	1371
Inter Prediction.....	1376
Macroblock Reconstruction.....	1408
Deblocking Filtering.....	1442
H.264 Encoder Functions.....	1458
Forward Transform and Quantization.....	1459
CAVLC Functions.....	1467
CABAC Functions.....	1471
Inverse Quantization and Transform.....	1479
AVS.....	1480
AVS Decoder Functions.....	1483
Entropy Decoding.....	1484
Inter Prediction.....	1486
Macroblock Reconstruction.....	1489
Deblocking Filtering.....	1493
AVS Encoder Functions.....	1500
Forward Transform and Quantization.....	1501
Macroblock Disassembling.....	1502
VC-1.....	1504
VC-1 Decoder.....	1509
Inverse Transform.....	1511
Interpolation.....	1519
Smoothing.....	1527
Deblocking.....	1536
Dequantization.....	1539
Range Reduction.....	1545
VC-1 Encoder.....	1546
Forward Transform.....	1547
Quantization.....	1552

## **Appendix A: Handling of Special Cases**

## **Appendix B: Interpolation in Image Geometric Transform Functions**

Overview of Interpolation Modes.....	1559
Mathematical Notation.....	1560
Nearest Neighbor Interpolation.....	1560
Linear Interpolation.....	1561
Cubic Interpolation.....	1562

Super Sampling.....1563

Lanczos Interpolation.....1563

Interpolation with Two-Parameter Cubic Filters.....1564

**Bibliography**

**Glossary**



INFORMATION IN THIS DOCUMENT IS PROVIDED IN CONNECTION WITH INTEL® PRODUCTS. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. EXCEPT AS PROVIDED IN INTEL'S TERMS AND CONDITIONS OF SALE FOR SUCH PRODUCTS, INTEL ASSUMES NO LIABILITY WHATSOEVER, AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO SALE AND/OR USE OF INTEL PRODUCTS INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

UNLESS OTHERWISE AGREED IN WRITING BY INTEL, THE INTEL PRODUCTS ARE NOT DESIGNED NOR INTENDED FOR ANY APPLICATION IN WHICH THE FAILURE OF THE INTEL PRODUCT COULD CREATE A SITUATION WHERE PERSONAL INJURY OR DEATH MAY OCCUR.

Intel may make changes to specifications and product descriptions at any time, without notice. Designers must not rely on the absence or characteristics of any features or instructions marked "reserved" or "undefined." Intel reserves these for future definition and shall have no responsibility whatsoever for conflicts or incompatibilities arising from future changes to them. The information here is subject to change without notice. Do not finalize a design with this information.

The products described in this document may contain design defects or errors known as errata which may cause the product to deviate from published specifications. Current characterized errata are available on request.

Contact your local Intel sales office or your distributor to obtain the latest specifications and before placing your product order.

Copies of documents which have an order number and are referenced in this document, or other Intel literature, may be obtained by calling 1-800-548-4725, or go to

<http://www.intel.com/design/literature.htm>

Intel processor numbers are not a measure of performance. Processor numbers differentiate features within each processor family, not across different processor families. See [http://www.intel.com/products/processor\\_number](http://www.intel.com/products/processor_number) for details.

MPEG-1, MPEG-2, MPEG-4, H.261, H.263, H.264, MP3, DV, VC-1, MJPEG, AC3, and AAC are international standards promoted by ISO, IEC, ITU, ETSI and other organizations. Implementations of these standards, or the standard enabled platforms may require licenses from various entities, including Intel Corporation.

Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products. For more information go to <http://www.intel.com/performance>.

BunnyPeople, Celeron, Celeron Inside, Centrino, Centrino Inside, Cilk, Core Inside, i960, Intel, the Intel logo, Intel AppUp, Intel Atom, Intel Atom Inside, Intel Core, Intel Inside, Intel Inside logo, Intel NetBurst, Intel NetMerge, Intel NetStructure, Intel SingleDriver, Intel SpeedStep, Intel Sponsors of Tomorrow., the Intel Sponsors of Tomorrow. logo, Intel StrataFlash, Intel Viiv, Intel vPro, Intel XScale, InTru, the InTru logo, InTru soundmark, Itanium, Itanium Inside, MCS, MMX, Moblin, Pentium, Pentium Inside, skool, the skool logo, Sound Mark, The Journey Inside, vPro Inside, VTune, Xeon, and Xeon Inside are trademarks of Intel Corporation in the U.S. and other countries.

\* Other names and brands may be claimed as the property of others.

Microsoft, Windows, Visual Studio, Visual C++, and the Windows logo are trademarks, or registered trademarks of Microsoft Corporation in the United States and/or other countries.

Java and all Java based trademarks and logos are trademarks or registered trademarks of Sun Microsystems, Inc. in the U.S. and other countries.

Microsoft product screen shot(s) reprinted with permission from Microsoft Corporation.

Copyright© 2000-2012, Intel Corporation. All rights reserved.



# Volume Overview

This manual describes the structure, operation, and functions of the Intel® Integrated Performance Primitives (Intel® IPP) that operate on two-dimensional signals and are used for image and video processing. The manual explains the Intel IPP concepts, as well as specific data type definitions and operation models used in the image and video processing domain, and provides detailed descriptions of the Intel IPP image and video processing functions. The Intel IPP functions are combined in groups by their functionality. Each group of functions is described in a separate chapter (chapters 3 through 16).

For more information about image and video processing concepts and algorithms, refer to the books and materials listed in the [Bibliography](#).

## What's New

This Reference Manual documents Intel® Integrated Performance Primitives (Intel® IPP) 7.0 Update 7 release. Known bugs have been fixed in the document.

## Notational Conventions

The code and syntax used in this manual for function and variable declarations are written in the ANSI C style. However, versions of Intel IPP for different processors or operating systems may, of necessity, vary slightly.

### Optimization Notice

Intel's compilers may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include SSE2, SSE3, and SSSE3 instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on microprocessors not manufactured by Intel. Microprocessor-dependent optimizations in this product are intended for use with Intel microprocessors. Certain optimizations not specific to Intel microarchitecture are reserved for Intel microprocessors. Please refer to the applicable product User and Reference Guides for more information regarding the specific instruction sets covered by this notice.

Notice revision #20110804

This manual uses the following notational conventions:

Convention	Explanation	Example
THIS TYPE STYLE	Used in the text for the Intel IPP constant identifiers.	IPPI_INTER_LINEAR

Convention	Explanation	Example
This type style	Mixed with the uppercase in structure names; also used in function names, code examples and call statements.	IppiSize, ippiMomentInitAlloc()
<i>This type style</i>	Parameters in function prototypes and parameters description.	<i>value, srcStep</i>
$x(n)$ and $x[n]$	Used to represent a discrete 1D signal. The notation $x(n)$ refers to a conceptual signal, while the notation $x[n]$ refers to an actual vector. Both of these are annotated to indicate a specific finite range of values.	$x[n], 0 \leq n < len$ Typically, the number of elements in vectors is denoted by <i>len</i> . Vector names contain square brackets as distinct from vector elements with current index <i>n</i> . The expression $pDst[n] = pSrc[n] + val$ implies that each element $pDst[n]$ of the vector <i>pDst</i> is computed for each <i>n</i> in the range from 0 to <i>len</i> -1. Special cases are regarded and described separately.
Ipp<data-domain> and Ipp prefixes	All structures and enumerators, specific for the image and video processing domain have the Ippi prefix, while those common for entire Intel IPP software have the Ipp prefix.	IppiPoint, IppDitherType

## See Also

- [Function Naming](#)

# Intel® Integrated Performance Primitives Concepts

## 2

This chapter explains the purpose and structure of the Intel® Integrated Performance Primitives (Intel® IPP) for Intel® Architecture software and looks over some of the basic concepts used in the image and video processing part of Intel IPP. It also describes the supported data formats and operation modes, and defines function naming conventions in the manual.

## Function Naming

Naming conventions for the Intel IPP functions are similar for all covered domains.

Function names in Intel IPP have the following general format:

```
ipp<data-domain><name>_<datatype>[_<descriptors>](<parameters>);
```

The elements of this format are explained in the sections that follow.

### Data-Domain

The *data-domain* is a single character that denotes the subset of functionality to which a given function belongs. The current version of Intel IPP supports the following data-domains:

s	for signals (expected data type is a 1D signal)
i	for images and video (expected data type is a 2D image)
m	for matrices (expected data type is a matrix)
r	for realistic rendering functionality and 3D data processing (expected data type depends on supported rendering techniques)
g	for signals of fixed length

For example, function names that begin with `ippi` signify that respective functions are used for image or video processing.

### Name

The *name* field identifies what function does and has the following format:

```
<name> = <operation>[_<modifier>]
```

The *operation* component is one or more words, acronyms, and abbreviations that describe the core operation.

The *modifier* component, if present, is a word or abbreviation that denotes a slight modification or variation of the given function.

For example, names without modifiers: `Add`, `RGBToYCbCr`, `MorphAddGetSize`; with modifiers: `DCT8x8Inv_2x2`, `DCT8x8Inv_4x4`, `RGBToYCbCr_JPEG`.

### Data Types

The *datatype* field indicates data types used by the function, in the following format:

<bit depth><bit interpretation> ,

where

bit depth = <1|8|16|32|64>

and

bit interpretation = <u|s|f>[c]

Here *u* indicates “unsigned integer”, *s* indicates “signed integer”, *f* indicates “floating point”, and *c* indicates “complex”.

Intel IPP supports the following data types for image and video processing functions:

8u	8 bit, unsigned data
8s	8 bit, signed data
16u	16 bit, unsigned data
16uc	16-bit, complex unsigned short data <sup>†</sup>
16s	16 bit, signed data
16sc	16-bit, complex short data
32u	32 bit, unsigned data
32s	32 bit, signed data
32sc	32-bit, complex int data
32f	32-bit, single-precision real floating point data
32fc	32-bit, single-precision complex floating point data
64s	64-bit, quadword signed data
64f	64-bit, double-precision real floating point data

---

<sup>†</sup> - only partial support for intermediate result after transforms (in the so-called “time” domain).



**NOTE.** Intel IPP does not support `1u` data type, therefore bitonal images must be converted to `8u` gray scale images for further processing by the library functions. There is a special function for such conversion `ippiConvert_1u8u_C1R`.

---

The formats for complex data are represented in Intel IPP by structures defined as follows:

```
typedef struct {
    Ipp16s re;
    Ipp16s im;
} Ipp16sc;

typedef struct {
    Ipp32s re;
    Ipp32s im;
} Ipp32sc;

typedef struct {
    Ipp32f re;
    Ipp32f im;
} Ipp32fc;
```

where *re*, *im* denote the real and imaginary parts, respectively.

Complex data formats are used by several arithmetic image processing functions. The *32fc* format is also used to store input/output data in some Fourier transform functions.

The 64-bit formats, *64s* and *64f*, are used for storing data computed by some image statistics functions.

For functions that operate on a single data type, the *datatype* field contains only one of the values listed above.

If a function operates on source and destination images that have different data types, the respective data type identifiers are listed in the function name in order of source and destination as follows:

*<datatype>* = *<src1Datatype>*[*src2Datatype*][*dstDatatype*]

For example, the function that converts 8-bit unsigned source image data to 32-bit floating point destination image data has the *8u32f* value for the *datatype* field.



**NOTE.** In the lists of function parameters (arguments), the *Ipp* prefix is written in the data type. For example, the 8-bit unsigned data is denoted as *Ipp8u* type. These Intel IPP-specific data types are defined in the respective library header files.

## Descriptors

The *descriptors* field further describes the operation. Descriptors are individual characters that indicate additional details of the operation.

The following descriptors are used in image and video processing functions:

Descriptor	Description	Example
A	Image data contains an alpha channel as the last channel, requires C4, alpha-channel is not processed	<code>ippiAddC_8u_AC4R</code> ,
A0	Image data contains an alpha channel as the first channel, requires C4, alpha-channel is not processed	<code>ippiLUT-Palette_8u_C3A0C4R</code>
C1, C2, C3, C4	Image data is in pixel order and made up of 1, 2, 3 or 4 discrete interleaved channels	<code>ippiFilter_32f_C1R ippiFilter_32f_C3R ippiFilter_32f_C4R ippiResizeCCRotate_8u_C2R</code>
C	Channel of interest (COI) is used in the operation	<code>ippiSet_8u_C3CR</code>

Descriptor	Description	Example
I	Operation is performed in-place - that is result of operation is written back into the source (default is not-in-place)	<code>ippiMulPack_32f_C3IR</code>
M	Operation uses a mask to determine pixels to be processed	<code>ippiCopy_8u_C1MR</code>
P2, P3, P4	Image data is in planar order and made up of 2, 3 or 4 discrete planar (non-interleaved) channels, with a separate pointer to each plane	<code>ippiRe-sizeYUV420_8u_P2R</code> , <code>ippiGammaFwd_8u_P3R</code> , <code>ippiCopy_32f_P4R</code>
R	Function operates on a defined region of interest (ROI) for each source image	<code>ippiMean_8u_C4R</code>
Sfs	Saturation and fixed scaling mode is used	<code>ippiSubC_8u_C1RSfs</code>
S	Saturation and no scaling (default)	<code>ippiCon-vert_16s16u_C1Rs</code>

If more than one descriptor is used, they are presented in the function name in alphabetical order.

Every function that operates on image data has a channel count descriptor `Cn` (for interleaved image) or `Pn` (for planar). No default channel count is defined.

If input and output channel layouts are different, both source and destination layouts are listed. For example, the function `ippiHLSToBGR_8u_C3P3R` converts three-channel interleaved HLS image to the three-plane BGR image.

## Parameters

The parameters in functions are in the following order: all source operands, all destination operands, all other operation-specific parameters.

Source parameters are named `Src` or `SrcN`, if there is more than one input image. Destination parameters are named `Dst`. For in-place operations, the input/output parameter contains the name `SrcDst`. All parameters defined as pointers start with lowercase `p`, for example, `pSrc`, `pMean`, `pSpec`.

## Function Prototypes in Intel IPP

Function names in Intel IPP contain *datatype* and *descriptor* fields after the *name* field (see [Function Naming](#) section in this chapter). Most of the Intel IPP functions for image processing have a number of flavors that differ in data types associated with the operation, and in some additional parameters.

Each function flavor has its unique prototype used in function definition and for calling the function from the application program. For many flavors of a given function, these prototypes look quite similar.

To avoid listing all the similar prototypes in function description sections of some chapters in this manual, only different templates for such prototypes followed by the table of applicable data types and descriptors for each function may be given. For simplicity, in such cases the data type and descriptor fields in the function name are denoted as *mod*:

`<mod> = <datatype>_<datatype>`

For example, the template for the prototype of the image dilation function that performs not-in-place operation, looks like this:



```
IppStatus ippiDilate_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pDst,
int dstStep, IppiSize roiSize);
```

where the supported values for *mod* are:

```
8u_C1R
8u_C3R
8u_AC4R
```

This notation means that the `ippiDilate` function has three flavors for a not-in-place operation, which process 8-bit unsigned data (of `Ipp8u` type) and differ in the number of channels in processed images. These flavors have the following prototypes:

```
IppStatus ippiDilate_8u_C1R(const Ipp8u* pSrc, int srcStep,
Ipp8u* pDst, int dstStep, IppiSize roiSize);

IppStatus ippiDilate_8u_C3R(const Ipp8u* pSrc, int srcStep,
Ipp8u* pDst, int dstStep, IppiSize roiSize);

IppStatus ippiDilate_8u_AC4R(const Ipp8u* pSrc, int srcStep,
Ipp8u* pDst, int dstStep, IppiSize roiSize);
```

Thus, to obtain the full name and parameters list for the specific function flavor, not listed directly, do the following:

1. Choose the function operation mode (denoted in this manual as **Case 1**, **2** and so on) and look in the table for the supported data types and descriptors.
2. Set the *mod* field in the function name as the concatenation of the chosen data type and descriptor, delimited by the underscore.
3. Use the respective template, substituting all the *datatype* fields in the parameters list with the chosen data type. Note that `Ipp` prefix is written before the *datatype* in the parameters list (see [Data Types](#) in this chapter for details).

## Example

To get the prototype for the `ippiSet` function flavor that sets each channel of a 3-channel destination image to 16-bit signed values, choose **Case 2: Setting each color channel to a specified value** and use *datatype* = `16s`, *descriptors* = `C3R`.

After substituting the *mod* field with `16s_C3R`, obtain the required prototype as

```
ippiSet_16s_C3R(const Ipp16svalue[3], Ipp16* pDst, int dstStep, IppiSize roiSize);
```

## Rounding Mode

As many Intel IPP functions have to meet the bit-exact requirement, image processing functions use rounding. The default rounding mode for all functions can be described as "nearest even", that is the fixed point number  $x=N + \alpha$ ,  $0 \leq \alpha < 0.5$ , where  $N$  is an integer number, is rounded as given by:

$$\lceil x \rceil = \begin{cases} N, & 0 \leq \alpha < 0.5 \\ N+1, & 0.5 < \alpha < 1 \\ N, & \alpha = 0.5, N - \text{even} \\ N+1, & \alpha = 0.5, N - \text{odd} \end{cases}$$

For example, 1.5 is rounded to 2, and 2.5 to 2.

Some image processing functions have additional rounding modes, which are set by the parameter *roundMode*.

## Integer Result Scaling

The default for image processing functions is to saturate the results without scaling them.

Some image processing functions operating on integer data use scaling of the internally computed output results by the integer *scaleFactor*, which is specified as one of the function parameters. These functions have the *Sfs* descriptor in their names.

The scale factor can be negative, positive, or zero. Scaling is applied because internal computations are generally performed with a higher precision than the data types used for input and output images.



**NOTE.** The result of integer operations is always saturated to the destination data type range even when scaling is used.

The scaling of an integer result is done by multiplying the output pixel values by  $2^{-\text{scaleFactor}}$  before the function returns. This helps retain either the output data range or its precision. Usually the scaling with a positive factor is performed by the shift operation. The result is rounded off to the nearest even integer number (see [Rounding Mode](#)).

For example, the integer *Ipp16s* result of the square operation *ippiSqr* for the input value 200 is equal to 32767 instead of 40000, that is, the result is saturated and the exact value cannot be restored. The scaling of the output value with the factor *scaleFactor* = 1 yields the result 20000, which is not saturated, and the exact value can be restored as  $20000 \times 2$ . Thus, the output data range is retained.

The following example shows how the precision can be partially retained by means of scaling. The integer square root operation *ippiSqr* (without scaling) for the input value 2 gives the result equal to 1 instead of 1.414. Scaling of the internally computed output value with the factor *scaleFactor* = -3 gives the result 11, and permits the more precise value to be restored as  $11 \times 2^{-3} = 1.375$ .

## Error Reporting

The Intel IPP functions return status codes of the performed operation to report errors and warnings to the calling program. Thus, it is up to the application to perform error-related actions and/or recover from the error. The last value of the error status is not stored, and the user is to decide whether to check it or not as the function returns. The status codes are of *IppStatus* type and are global constant integers.

The status codes and corresponding messages reported by Intel IPP for image and video processing are listed in Table .

**Status Codes and Messages**

<b>Status Code</b>	<b>Message</b>
ippStsNotSupportedModeErr	The requested mode is currently not supported
ippStsDecimateFractionErr	Unsupported fraction in decimate
ippStsWeightErr	Wrong value of weight
ippStsQualityIndexErr	Quality Index cannot be calculated for image filled with a constant
ippStsResizeNoOperationErr	One of the output image dimensions is less than 1 pixel
ippStsBlockStepErr	Step for Block less than 8
ippStsMBStepErr	Step for MB less than 16
ippStsNoiseRangeErr	Noise value for Wiener Filter is out of range
ippStsLPCCalcErr	Linear prediction could not be evaluated
ippStsJPEG2KBadPassNumber	Pass number exceeds allowed limits [ 0 , nOfPasses-1 ]
ippStsJPEG2KDamagedCodeBlock	Codeblock for decoding is damaged
ippStsH263CBPYCodeErr	Illegal Huffman code during CBPY stream processing
ippStsH263MCBPCInterCodeErr	Illegal Huffman code while MCBPC Inter stream processing
ippStsH263MCBPCIntraCodeErr	Illegal Huffman code while MCBPC Intra stream processing
ippStsNotEvenStepErr	Step value is not pixel multiple
ippStsHistoNofLevelsErr	Number of levels for histogram is less than 2
ippStsLUTNofLevelsErr	Number of levels for LUT is less than 2
ippStsMP4BitOffsetErr	Incorrect bit offset value
ippStsMP4QPErr	Incorrect quantization parameter
ippStsMP4BlockIdxErr	Incorrect block index
ippStsMP4BlockTypeErr	Incorrect block type
ippStsMP4MVCodeErr	Illegal Huffman code while MV stream processing
ippStsMP4VLCCodeErr	Illegal Huffman code while VLC stream processing
ippStsMP4DCCodeErr	Illegal code while DC stream processing
ippStsMP4FcodeErr	Incorrect fcode value
ippStsMP4AlignErr	Incorrect buffer alignment
ippStsMP4TempDiffErr	Incorrect temporal difference
ippStsMP4BlockSizeErr	Incorrect size of block or macroblock
ippStsMP4ZeroBABErr	All BAB values are zero
ippStsMP4PredDirErr	Incorrect prediction direction
ippStsMP4BitsPerPixelErr	Incorrect number of bits per pixel
ippStsMP4VideoCompModeErr	Incorrect video component mode
ippStsMP4LinearModeErr	Incorrect DC linear mode
ippStsH263PredModeErr	Prediction Mode value error
ippStsH263BlockStepErr	The step value is less than 8
ippStsH263MBStepErr	The step value is less than 16
ippStsH263FrameWidthErr	The frame width is less than 8
ippStsH263FrameHeightErr	The frame height is less than or equal to zero
ippStsH263ExpandPelsErr	The expand pixels number is less than 8
ippStsH263PlaneStepErr	Step value is less than the plane width
ippStsH263QuantErr	Quantizer value is less than or equal to zero, or greater than 31
ippStsH263MVCodeErr	Illegal Huffman code while MV stream processing

Status Code	Message
ippStsH263VLCCodeErr	Illegal Huffman code while VLC stream processing
ippStsH263DCCodeErr	Illegal code while DC stream processing
ippStsH263ZigzagLenErr	Zigzag compact length is more than 64
ippStsJPEGHuffTableErr	JPEG Huffman table is destroyed
ippStsJPEGDCTRangeErr	JPEG DCT coefficient is out of range
ippStsJPEGOutOfBufErr	Attempt to access out of the buffer
ippStsChannelOrderErr	Wrong order of the destination channels
ippStsZeroMaskValueErr	All values of the mask are zero
ippStsRangeErr	Bad values of bounds: the lower bound is greater than the upper bound
ippStsQPErr	Incorrect value of quantizer parameter
ippStsQuadErr	The quadrangle is nonconvex or degenerates into triangle, line or point
ippStsRectErr	Size of the rectangular region is less than or equal to 1
ippStsCoeffErr	Unallowable values of the transformation coefficients
ippStsNoiseValErr	Bad value of noise amplitude for dithering
ippStsDitherLevelsErr	Number of dithering levels is out of range
ippStsNumChannelsErr	Bad or unsupported number of channels
ippStsDataTypeErr	Bad or unsupported data type
ippStsCOIErr	COI is out of range
ippStsOutOfRangeErr	Argument is out of range or point is outside the image
ippStsDivisorErr	Divisor is equal to zero, function is aborted
ippStsAlphaTypeErr	Illegal type of image compositing operation
ippStsGammaRangeErr	Gamma range bound is less than or equal to zero
ippStsGrayCoefSumErr	Sum of the conversion coefficients must be less than or equal to 1
ippStsChannelErr	Illegal channel number
ippStsJaehneErr	Magnitude value is negative
ippStsStepErr	Step value is less than or equal to zero
ippStsStrideErr	Stride value is less than the row length
ippStsEpsValErr	Negative epsilon value error
ippStsScaleRangeErr	Scale bounds are out of range
ippStsThresholdErr	Invalid threshold bounds
ippStsWtOffsetErr	Invalid offset value of wavelet filter
ippStsAnchorErr	Anchor point is outside the mask
ippStsMaskSizeErr	Invalid mask size
ippStsShiftErr	Shift value is less than zero
ippStsSampleFactorErr	Sampling factor is less than or equal to zero
ippStsResizeFactorErr	Resize factor(s) is less or equal to zero
ippStsDivByZeroErr	An attempt to divide by zero
ippStsInterpolationErr	Invalid interpolation mode
ippStsMirrorFlipErr	Invalid flip mode
ippStsMoment00ZeroErr	Moment value M(0,0) is too small to continue calculations
ippStsThreshNegLevelErr	Negative value of the level in the threshold operation
ippStsContextMatchErr	Context parameter does not match the operation
ippStsFftFlagErr	Invalid value of the FFT flag parameter

Status Code	Message
<code>ippStsFftOrderErr</code>	Invalid value of the FFT order parameter
<code>ippStsMemAllocErr</code>	Not enough memory for the operation
<code>ippStsNullPtrErr</code>	Null pointer error
<code>ippStsSizeErr</code>	Wrong value of data size
<code>ippStsBadArgErr</code>	Invalid or bad argument
<code>ippStsNoErr</code>	No error
<code>ippStsNoOperation</code>	No operation has been executed
<code>ippStsMisalignedBuf</code>	Misaligned pointer in operation in which it must be aligned
<code>ippStsSqrtNegArg</code>	Negative value(s) of the argument in the function <code>Sqrt</code>
<code>ippStsInvZero</code>	INF result. Zero value was met by <code>InvThresh</code> with zero level
<code>ippStsEvenMedianMaskSize</code>	Even size of the Median Filter mask was replaced by the odd number
<code>ippStsDivByZero</code>	Zero value(s) of the divisor in the function <code>Div</code>
<code>ippStsLnZeroArg</code>	Zero value(s) of the argument in the function <code>Ln</code>
<code>ippStsLnNegArg</code>	Negative value(s) of the argument in the function <code>Ln</code>
<code>ippStsNanArg</code>	Not a Number argument value warning
<code>ippStsDoubleSize</code>	Sizes of image are not multiples of 2
<code>ippStsJPEGMarker</code>	JPEG marker was met in the bitstream
<code>ippStsResFloor</code>	All result values are floored
<code>ippStsAffineQuadChanged</code>	4th vertex of destination quad is not equal to customer's one
<code>ippStsWrongIntersectROI</code>	Wrong ROI that has no intersection with the source or destination image. No operation
<code>ippStsWrongIntersectQuad</code>	Wrong quadrangle that has no intersection with the source or destination ROI. No operation
<code>ippStsSymKernelExpected</code>	The kernel is not symmetric
<code>ippStsEvenMedianWeight</code>	Even weight of the Weighted Median Filter was replaced by the odd one
<code>ippStsNoAntialiasing</code>	The mode does not support antialiasing

The status codes ending with `Err` (except for the `ippStsNoErr` status) indicate an error; the integer values of these codes are negative. When an error occurs, the function execution is interrupted.

The status code `ippStsNoErr` indicates no error. All other status codes indicate warnings. When a specific case is encountered, the function execution is completed and the corresponding warning status is returned. For example, if the function `ippiDiv` meets an attempt to divide a positive value by zero, the function execution is not aborted. The result of the operation is set to the maximum value that can be represented by the source data type, and the user is warned by the output status `ippStsDivByZero`. See appendix A [Handling of Special Cases](#) for more information.

## Structures and Enumerators

This section describes the structures and enumerators used by the Intel Integrated Performance Primitives for image and video processing.

The `IppStatus` constant enumerates the status code values returned by Intel IPP functions, indicating whether the operation was error-free or not.

See section [Error Reporting](#) in this chapter for more information on the set of valid status codes and corresponding error messages for image and video processing functions.

The structure `IppiPoint` for storing the geometric position of a point is defined as

```
typedef struct {  
    int x;  
    int y;  
} IppiPoint;
```

where `x`, `y` denote the coordinates of the point.

The structure `IppPointPolar` for storing the geometric position of a point in polar coordinates is defined as

```
typedef struct {  
    Ipp32f rho;  
    Ipp32f theta;  
} IppPointPolar;
```

where `rho` - a radial coordinate (radial distance from the origin), `theta` - an angular coordinate (counterclockwise angle from the `x`-axis).

The structure `IppiSize` for storing the size of a rectangle is defined as

```
typedef struct {  
    int width;  
    int height;  
} IppiSize;
```

where `width` and `height` denote the dimensions of the rectangle in the `x`- and `y`-directions, respectively.

The structure `IppiRect` for storing the geometric position and size of a rectangle is defined as

```
typedef struct {  
    int x;  
    int y;  
    int width;  
    int height;  
} IppiRect;
```

where `x`, `y` denote the coordinates of the top left corner of the rectangle that has dimensions `width` in the `x`-direction by `height` in the `y`-direction.

The `ippiConnectedComp` structure used in [Computer Vision functions](#) defines the connected component as follows:

```
typedef struct _IppiConnectedComp {  
    Ipp64f area;  
    Ipp64f value[3];  
    IppiRect rect;  
} IppiConnectedComp;
```

where `area` - area of the segmented component; `value[3]` - gray scale value of the segmented component; `rect` - bounding rectangle of the segmented component.

The `IppiMaskSize` enumeration defines the neighborhood area for some [morphological](#) and [filtering functions](#):

```
typedef enum {
    ippMskSize1x3 = 13,
    ippMskSize1x5 = 15,
    ippMskSize3x1 = 31,
    ippMskSize3x3 = 33,
    ippMskSize5x1 = 51,
    ippMskSize5x5 = 55
} IppiMaskSize;
```

The `IppCmpOp` enumeration defines the type of compare operation to be used in image [comparison functions](#):

```
typedef enum {
    ippCmpLess,
    ippCmpLessEq,
    ippCmpEq,
    ippCmpGreaterEq,
    ippCmpGreater
} IppCmpOp;
```

The `IppRoundMode` enumeration defines the rounding mode to be used in some conversion, filtering and arithmetic functions:

```
typedef enum {
    ippRndZero,
    ippRndNear,
    ippRndFinancial
} IppRoundMode;
```

The `IppHintAlgorithm` enumeration defines the type of code to be used in some image transform and statistics functions, that is, faster but less accurate, or vice-versa, more accurate but slower. For more information on using this enumerator, see Table [Hint Arguments for Linear Transform Functions](#) or Table [Hint Arguments for Image Moment Functions](#).

```
typedef enum {
    ippAlgHintNone,
    ippAlgHintFast,
    ippAlgHintAccurate
} IppHintAlgorithm;
```

The types of interpolation used by [geometric transform functions](#) are defined as follows:

```
enum {
    IPPI_INTER_NN          = 1,
    IPPI_INTER_LINEAR      = 2,
    IPPI_INTER_CUBIC       = 4,
    IPPI_INTER_CUBIC2P_BSPLINE,
    IPPI_INTER_CUBIC2P_CATMULLROM,
    IPPI_INTER_CUBIC2P_B05C03,
    IPPI_INTER_SUPER       = 8,
    IPPI_INTER_LANCZOS     = 16,
    IPPI_ANTIALIASING      =(1 << 29)
    IPPI_SUBPIXEL_EDGE     =(1 << 30)
    IPPI_SMOOTH_EDGE       =(1 << 31)
};
```

The `IppiAlphaType` enumeration defines the type of the compositing operation to be used in the [alpha composition functions](#):

```
typedef enum {
    ippAlphaOver,
    ippAlphaIn,
    ippAlphaOut,
    ippAlphaATop,
    ippAlphaXor,
    ippAlphaPlus,
    ippAlphaOverPremul,
    ippAlphaInPremul,
    ippAlphaOutPremul,
    ippAlphaATopPremul,
    ippAlphaXorPremul,
    ippAlphaPlusPremul
} IppiAlphaType;
```

The `IppiDitherType` enumeration defines the type of dithering to be used by the `ippiReduceBits` function:

```
typedef enum {
    ippDitherNone,
    ippDitherFS,
    ippDitherJN,
    ippDitherStucki,
    ippDitherBayer
} IppiDitherType;
```

The layout of the image slices used in some [image format conversion functions](#) is defined as follows:

```
enum {
    IPP_UPPER          = 1,
    IPP_LEFT           = 2,
    IPP_CENTER         = 4,
    IPP_RIGHT          = 8,
    IPP_LOWER          = 16,
    IPP_UPPER_LEFT     = 32,
    IPP_UPPER_RIGHT    = 64,
    IPP_LOWER_LEFT     = 128,
    IPP_LOWER_RIGHT    = 256
};
```

The `IppiShape` enumeration defines shapes of the structuring element used in some [morphological functions](#):

```
typedef enum {
    ippiShapeRect      = 0,
    ippiShapeCross      = 1,
    ippiShapeEllipse    = 2,
    ippiShapeCustom     = 100
} IppiShape;
```



The `IppiAxis` enumeration defines the flip axes for the `ippiMirror` functions or direction of the image intensity ramp for the `ippiImageRamp` functions:

```
typedef enum {
    ippAxsHorizontal,
    ippAxsVertical,
    ippAxsBoth
} IppiAxis;
```

The `IppiBorderType` enumeration defines the border type that is used by some [Separable Filters](#) and [Fixed Filters with Border](#) functions:

```
typedef enum _IppiBorderType {
    ippBorderConst      = 0,
    ippBorderRepl       = 1,
    ippBorderWrap       = 2,
    ippBorderMirror      = 3,
    ippBorderMirrorR     = 4,
    ippBorderMirror2     = 5,
    ippBorderInMem      = 6,
    ippBorderInMemTop    = 0x0010,
    ippBorderInMemBottom = 0x0020,
    ippBorderInMemLeft   = 0x0040,
    ippBorderInMemRoght  = 0x0080
} IppiBorderType;
```

The `IppiFraction` enumeration defines shapes of the structuring element used in some [decimate filter](#) functions:

```
typedef enum {
    ippPolyphase_1_2,
    ippPolyphase_3_5,
    ippPolyphase_2_3,
    ippPolyphase_7_10,
    ippPolyphase_3_4,
} IppiFraction;
```

The `IppiWTSubband` enumeration defines the appropriate wavelet transform subband used in the [JPEG2000 coding functions](#):

```
typedef enum {
    ippWTSubbandLxLy,
    ippWTSubbandLxHy,
    ippWTSubbandHxLy,
    ippWTSubbandHxHy
} IppiWTSubband;
```

The `IppiMQTermination` enumeration defines how the MQ-coder will be terminated during the operation of the [JPEG2000 coding functions](#):

```
typedef enum {
    ippMQTermSimple,
    ippMQTermNearOptimal,
    ippMQTermPredictable
} IppiMQTermination;
```

The `IppiMQRateAppr` enumeration defines the padding-approximation model in the estimation of rate and distortion procedures performed by the [JPEG2000 coding functions](#):

```
typedef enum {
    ippMQRateApprGood
} IppiMQRateAppr;
```

The code flags used by the [JPEG2000 coding functions](#) are defined as follows:

```
enum
{
    IPP_JPEG2K_VERTICALLY_CAUSAL_CONTEXT,
    IPP_JPEG2K_SELECTIVE_MQ_BYPASS,
    IPP_JPEG2K_TERMINATE_ON_EVERY_PASS,
    IPP_JPEG2K_RESETCTX_ON_EVERY_PASS,
    IPP_JPEG2K_USE_SEGMENTATION_SYMBOLS,
    IPP_JPEG2K_LOSSLESS_MODE,
    IPP_JPEG2K_DEC_CONCEAL_ERRORS,
    IPP_JPEG2K_DEC_DO_NOT_CLEAR_CB,
    IPP_JPEG2K_DEC_DO_NOT_RESET_LOW_BITS,
    IPP_JPEG2K_DEC_DO_NOT_CLEAR_SFBUFFER,
    IPP_JPEG2K_DEC_CHECK_PRED_TERM
};
```

The `IppiVLCAdaptType` enumeration defines the adaptive VLC model in the model adaptation procedure performed by the [JPEG XR coding functions](#):

```
typedef enum {
    ippVLCAdaptLowpass,
    ippVLCAdaptHighpass
} IppiVLCAdaptType;
```

The `IppiVLCScanType` enumeration defines the VLC scan type used by the [JPEG XR coding functions](#):

```
typedef enum {
    ippVLCScanVert,
    ippVLCScanHoriz,
    ippVLCScanRaster
} IppiVLCScanType;
```

## Function Context Structures

Some Intel IPP functions use special structures to store function-specific (context) information. For example, the `IppiFFTSpec` structure stores twiddle factors and bit reverse indexes needed in computing the fast Fourier transform.

Two different kinds of structures are used: structures that are not modified during function operation - they have the suffix `Spec` in their names, and structures that are modified during operation - they have the suffix `State` in their names.

These context-related structures are not defined in the public headers, and their fields are not accessible. It was done because the function context interpretation is processor dependent. Thus, you may only use context-related functions and may not create a function context as an automatic variable.

## Image Data Types and Ranges

The Intel IPP image processing functions support only absolute color images in which each pixel is represented by its channel intensities. The data storage for an image can be either pixel-oriented or plane-oriented (planar). For images in pixel order, all channel values for each pixel are clustered and stored consecutively, for example, RGBRGBRGB in case of an RGB image. The number of channels in a pixel-order image can be 1, 2, 3, or 4.

For images in planar order, all image data for each channel is stored contiguously followed by the next channel, for example, RRR...GGG...BBB.

Functions that operate on planar images are identified by the presence of `Pn` descriptor in their names. In this case, `n` pointers (one for each plane) may be specified.

The image data type is determined by the pixel depth in bits per channel, or bit depth. Bit depth for each channel can be 8, 16 or 32 and is included in the function name as one of these numbers (see [Function Naming](#) in this chapter for details). The data may be signed (`s`), unsigned (`u`), or floating-point real (`f`). For some arithmetic and FFT/DFT functions, data in complex format (`sc` or `fc`) can be used, where each channel value is represented by two numbers: real and imaginary part. All channels in an image must have the same data type.

For example, in an absolute color 24-bit RGB image, three consecutive bytes (24 bits) per pixel represent the three channel intensities in the pixel mode. This data type is identified in function names as the `8u_C3` descriptor, where `8u` represents 8-bit unsigned data for each channel and `C3` represents three channels.

Some functions operate with images in 16-bit packed RGB format (see [RGB Image Formats](#) in Chapter 6 for more details). In this case data of all 3 channels are represented as `16u` data type.

For example, in an absolute color 16-bit packed RGB image, two consecutive bytes (16 bits) per pixel represent the three channel intensities in the pixel mode. This data type is identified in function names as the `16u_C3` descriptor, where `16u` represents 16-bit unsigned data (not a bit depth) for all packed channels together and `C3` stands for three channels.

If an alpha (opacity) channel is present in image data, the image must have four channels, with alpha channel being the last or the first one. This data type is indicated by the `AC4` or `A0C4` descriptors respectively. The presence of the alpha channel can modify the function's behavior. For such functions, Intel IPP provides versions with and without alpha. If an alpha channel is specified, the operation usually is not performed on that channel.

The range of values that can be represented by each data type lies between the lower and upper bounds. The following table lists data ranges and constant identifiers used in Intel IPP to denote the respective range bounds:

### Image Data Types and Ranges

Data Type	Lower Bound		Upper Bound	
	Identifier	Value	Identifier	Value
8s	IPP_MIN_8S	-128	IPP_MAX_8S	127
8u		0	IPP_MAX_8U	255
16s	IPP_MIN_16S	-32768	IPP_MAX_16S	32767
16u		0	IPP_MAX_16U	65535
32s	IPP_MIN_32S	$-2^{31}$	IPP_MAX_32S	$2^{31}-1$
32u		0	IPP_MAX_32U	$2^{32}-1$

Data Type	Lower Bound		Upper Bound	
	Identifier	Value	Identifier	Value
32f <sup>†</sup>	IPP_MINABS_32F	1.175494351e <sup>-38</sup>	IPP_MAXABS_32F	3.402823466e <sup>38</sup>

<sup>†</sup> The range for absolute values

## Major Operation Models

Most Intel IPP image processing functions perform identical and independent operations on all channels of the processed image (except alpha channel). It means that the same operation is applied to each channel, and the computed results do not depend upon values of other channels. Some exceptions include the `ippiFilterMedianColor` function and color conversion functions, which process three channels together.

The Intel IPP image processing functions can be broken into two major models of operation: the functions that operate on one pixel to compute the result (also known as point operations), for example, `ippiAdd`, and the functions that operate on a group of pixels (also referred to as a neighborhood), for example, `ippiFilterBox`.

### Neighborhood Operations

The result of a neighborhood operation is based on values of a certain group of pixels, located near a given input pixel. The set of neighboring pixels is typically defined by a rectangular mask (or kernel) and anchor cell, specifying the mask alignment with respect to the position of the input pixel.

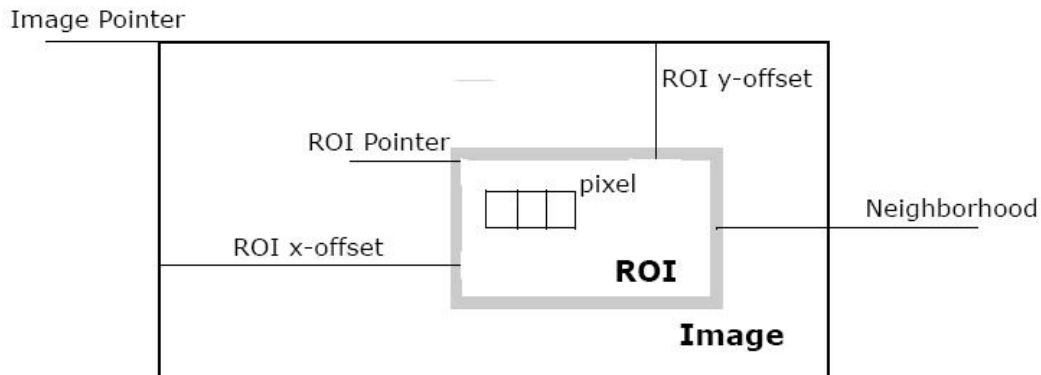
The Intel IPP functions that process a neighborhood operate on the assumption that all referred points of the image are available. To support this mode, the application must check that ROI parameters passed to the function have such values that all processed neighborhood pixels actually exist in the image.

### Regions of Interest in Intel IPP

Most Intel IPP image processing functions can operate not only on entire images but also on image areas. Image region of interest (ROI) is a rectangular area that may be either some part of the image or the whole image.

The Intel IPP functions with ROI support are distinguished by the presence of an  $\mathbb{R}$  descriptor in their names. ROI of an image is defined by the size and offset from the image origin as shown in Figure . The origin of an image is implied to be in the top left corner, with  $x$  values increasing from left to right and  $y$  values increasing downwards.

### Image, ROI, and Offsets



Both the source and destination images can have a region of interest. In such cases the sizes of ROIs are assumed to be the same while offsets may differ. The image processing is then performed on data of the source ROI, and the results are written to the destination ROI. In function call sequences, an ROI is specified by:

- *roiSize* parameter of the *IppiSize* type
- *pSrc* and *pDst* pointers to the starts of source and destination ROI buffers
- *srcStep* and *dstStep* parameters that are equal to distances in bytes between the starts of consecutive lines in source and destination images, respectively.

Thus, the parameters *srcStep*, *dstStep* set steps in bytes through image buffers to start processing a new line in the ROI of an image.

The following code example illustrates the use of the *dstStep* parameter in function calls:

#### Example

```
IppStatus alignedLine( void
) {
    Ipp8u x[8*3] = {0};
    IppiSize imgSize = {5,3};
    /// The image is of size 5x3. Width 8 has been
    /// chosen by the user to align every line of the image
    return ippiSet_8u_C1R( 7, x, 8, imgSize);
}
```

The resultant image *x* contains the following data:

```
07 07 07 07 07 00 00 00
07 07 07 07 07 00 00 00
07 07 07 07 07 00 00 00
```

If ROI is present,

- source and destination images can have different sizes
- lines may have padding at the end for aligning the line sizes
- application must correctly define the *pSrc*, *pDst*, and *roiSize* parameters.

The *pSrc* and *pDst* parameters are the shifted pointers to the image data. For example, in case of ROI operations on 3-bytes-per-pixel image data (8u\_C3R), *pSrc* points to the start of the source ROI buffer and can be interpreted as follows:

```
pSrc = pSrcImg + 3*(srcImgSize.width * srcRoiOffset.y + srcRoiOffset.x),
```

where

<i>pSrcImg</i>	points to the start of the source image buffer
<i>srcImgSize</i>	is the image size in pixels (of the <i>IppiSize</i> type)
<i>srcRoiOffset</i>	determines an offset of ROI relative to the start of the image as shown in Figure 2-1.

Another example for operations on four-channel image of 32f data type 32f\_AC4:

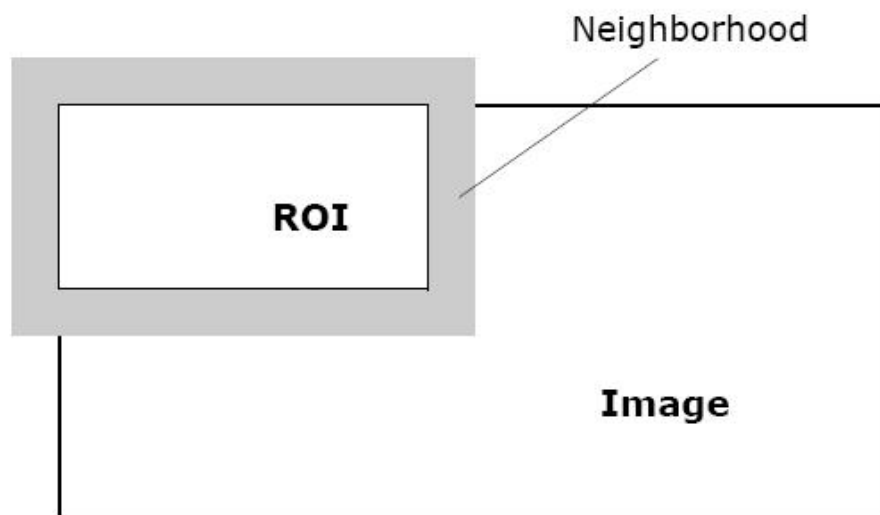
```
pSrc = (Ipp32f*)((Ipp8u*)pSrcImg + srcImgStep * srcRoiOffset.y +  
4*SizeOf(Ipp32f)*srcRoiOffset.x.
```

In this example the multiplier 4 is used because the AC4 pixel consists of 4 values - R, G, B, A. Pointer type conversion is required as in Intel IPP all image steps are always in bytes, and it's not recommended to use `step/SizeOf(Ipp32f)` as in a general case step value may be not a multiple of 4.

For functions using ROI with a neighborhood, you should correctly use values of the *pSrc* and *roiSize* parameters. These functions assume that the points in the neighborhood exist and that therefore *pSrc* is almost never equal to *pSrcImg*. Figure illustrates the case when neighborhood pixels can fall outside the source image.

### Using ROI with Neighborhood

---



To ensure valid operation when image pixels are processed, the application should correctly define additional border pixels (see [Borders](#)).



**WARNING.** If the required border pixels are not defined prior to calling neighborhood functions that attempt to process such pixels, you may get memory violation errors.

The following code example shows how to process an image with ROI:

### Example

```
IppStatus roi( void ) {  
    Ipp8u x[8*3] = {0};  
    IppiSize roiSize = {3,2};  
    IppiPoint roiPoint = {2,1};  
    /// place the pointer to the ROI start position  
    return ippiSet_8u_C1R( 7, x+8*roiPoint.y+roiPoint.x, 8, roiSize );  
}
```

The destination image *x* will contain the following data

```
00 00 00 00 00 00 00 00  
00 00 07 07 07 00 00 00  
00 00 07 07 07 00 00 00
```

### Tiled Image Processing

Intel IPP can process images composed from tiles, or tiled images. The specially created code sample “Tiled Image processing” demonstrates how to do it. See *Intel IPP Image Processing Samples* downloadable from <http://www.intel.com/cd/software/products/asmo-na/eng/220046.htm>





# Support Functions

This chapter describes the Intel® IPP support functions that are used to:

- retrieve information about the current Intel IPP software version
- get a brief explanation of the returned status codes
- allocate and free memory that is needed for the operation of other Intel IPP image and video processing functions.

The list of support functions is given in Table "Intel IPP Support Functions":

## Intel IPP Support Functions

Function Base Name	Operation
<b>Version Information Functions</b>	
<code>GetLibVersion</code>	Returns information about the active library version
<b>Status Information Functions</b>	
<code>ippGetStatusString</code>	Translates a status code into a message
<b>Memory Allocation Functions</b>	
<code>Malloc</code>	Allocates memory aligned to 32-byte boundary
<code>Free</code>	Frees memory allocated by the function <code>ippiMalloc</code>

## Version Information Function

This function returns the version number and other information about the active Intel IPP image processing software.

### GetLibVersion

*Returns information about the used version of Intel IPP software for image processing.*

#### Syntax

```
const IppLibraryVersion* ippiGetLibVersion(void);
```

## Description

The function `ippiGetLibVersion` is declared in the `ippi.h` file.

This function returns a pointer to a static data structure `IppLibraryVersion` that contains information about the current version of Intel IPP for image processing. You need not release memory referenced by the returned pointer, as it points to a static variable. The following fields of the `IppLibraryVersion` structure are available:

<i>major</i>	the major number of the current library version
<i>minor</i>	the minor number of the current library version
<i>majorBuild</i>	the number of builds of the major version
<i>build</i>	current build number
<i>Name</i>	the name of the current library version
<i>Version</i>	the library version string
<i>BuildDate</i>	the library version actual build date

For example, if the library version is 7.0 , library name is `ippip8_1.lib`, and build date is July, 20 2011, then the fields in this structure are set as:

```
major = 7, minor = 0, Name = "ippip8", Version = "7.0 build 205.68", BuildDate = "Jul 20 2011"
```



---

**NOTE.** Each sub-library that is used in the image processing domain has its own similar function to retrieve information about the active library version. These functions are: `ippiGetLibVersion`, `ippjGetLibVersion`, `ippcvGetLibVersion`, `ippvcGetLibVersion`, and `ippccGetLibVersion`. They are declared in the following header files: `ippcore.h`, `ippj.h`, `ippcv.h`, `ippvc.h`, `ippcc.h`, respectively, and have the same interface as the above described function.

---

## Example

The code example below shows how to use `ippiGetLibVersion`:

```
void libinfo(void) {  
    const IppLibraryVersion* lib = ippiGetLibVersion();  
    printf("%s %s %d.%d.%d.%d\n", lib->Name, lib->Version,  
        lib->major, lib->minor, lib->majorBuild, lib->build);  
}
```

Output:  
`ippip8_1.lib 7.0 build 205.68`

## Status Information Function

Use this function to get a brief description of the status code returned by the current Intel IPP software.

## ippGetStatusString

*Translates a status code into a message.*

---

### Syntax

```
const char* ippGetStatusString(IppStatus StsCode);
```

### Parameters

*StsCode* Code that indicates the status type (see Table 2-1)

### Description

The function `ippGetStatusString` is declared in the `ippcore.h` file. This function returns a pointer to the text string associated with a status code *StsCode*. Use this function to produce error and warning messages. The returned pointer is a pointer to an internal static buffer and needs not be released.

The status information function translates this code into the corresponding message `Null Pointer Error`:

### Example

---

A code example below shows how to use the `ippGetStatusString` function. If you call an Intel IPP function, in this example `ippiSet_8u_C1R`, with a `NULL` pointer, it returns an error code `-8`.

```
void statusInfo( void ) {  
    IppiSize roi = {0};  
    IppStatus st = ippiSet_8u_C1R(3, 0, 0, roi );  
    printf( " %d : %s\n", st, ippGetStatusString( st ));  
}
```

Output:

```
-8, Null Pointer Error
```

## Memory Allocation Functions

This section describes the Intel IPP functions that allocate aligned memory blocks for data of required type, or free previously allocated memory.



---

**NOTE.** The only function to free the memory allocated by any of these functions is `ippiFree()`.

---

## Malloc

*Allocates memory aligned to 32-byte boundary.*

---

### Syntax

```
Ipp<datatype>* ippiMalloc<mod>(int widthPixels, int heightPixels, int* pStepBytes);
```

Supported values for *mod*:

8u_C1	16u_C1	16s_C1	32s_C1	32f_C1	32sc_C1	32fc_C1
8u_C2	16u_C2	16s_C2	32s_C2	32f_C2	32sc_C2	32fc_C2
8u_C3	16u_C3	16s_C3	32s_C3	32f_C3	32sc_C3	32fc_C3
8u_C4	16u_C4	16s_C4	32s_C4	32f_C4	32sc_C4	32fc_C4
8u_AC4	16u_AC4	16s_AC4	32s_AC4	32f_AC4	32sc_AC4	32fc_AC4

### Parameters

<i>widthPixels</i>	Width of an image in pixels
<i>heightPixels</i>	Height of an image in pixels
<i>pStepBytes</i>	Pointer to the distance in bytes between starts of consecutive lines in the image

### Description

The function `ippiMalloc` is declared in the `ippi.h` file. This function allocates a memory block aligned to a 32-byte boundary for elements of different data types. Every line of the image is aligned in accordance with the *pStepBytes* parameter, which is calculated by the `ippiMalloc` function and returned for further use.

The function `ippiMalloc` allocates one continuous memory block, whereas functions that operate on planar images require an array of separate pointers (`IppType* plane[3]`) to each plane as input. The `ippiMalloc` function should be called three times in this case.

### Example

---

The code example below demonstrates how to construct such an array and set correct values to the pointers to use the allocated memory block with the Intel IPP functions operating on planar images. Note that *pStepBytes* should be specified for each plane. The example is given for 8u data type.

```
int stepBytes[3];
Ipp8u* plane[3];
plane[0] = ippiMalloc_8u_C1(widthPixels, heightPixels,
                           &(stepBytes [0]));
plane[1] = ippiMalloc_8u_C1(widthPixels/2, heightPixels/2,
                           &(stepBytes [1]));
plane[2] = ippiMalloc_8u_C1(widthPixels/2, heightPixels/2,
                           &(stepBytes [2]));
```

## Return Values

The return value of `ippiMalloc` function is a pointer to an aligned memory block.

If no memory is available in the system, the `NULL` value is returned.

To free the allocated memory block, use the `ippiFree` function.

## Free

*Frees memory allocated by the function `ippiMalloc`.*

---

## Syntax

```
void ippiFree(void* ptr);
```

## Parameters

<i>ptr</i>	Pointer to a memory block to be freed. This block must have been previously allocated by the function <code>ippiMalloc</code> .
------------	---

## Description

The function `ippiFree` is declared in the `ippi.h` file. This function frees the aligned memory block allocated by the function `ippiMalloc`.



---

**NOTE.** The function `ippiFree` cannot be used to free memory allocated by standard functions like `malloc` or `calloc`, nor can the memory allocated by `ippiMalloc` be freed by `free`.

---



# Image Data Exchange and Initialization Functions

This chapter describes the Intel® IPP image processing functions that perform image data manipulation, exchange and initialization operations. Table "Image Data Exchange and Initialization Functions" lists the functions described in more detail later in this chapter:

## Image Data Exchange and Initialization Functions

Function Base Name	Operation
<a href="#">Convert</a>	Converts pixel values in an image from one bit depth to another
<a href="#">Scale</a>	Scales pixel values of a image and converts them to another bit depth
<a href="#">Set</a>	Sets an array of pixels to a value
<a href="#">Copy</a>	Copies pixel values between two images
<a href="#">CopyConstBorder</a>	Copies pixels values between two images and adds the border pixels with a constant value
<a href="#">CopyReplicateBorder</a>	Copies pixel values between two images and adds the replicated border pixels
<a href="#">CopyWrapBorder</a>	Copies pixel values between two images and adds the border pixels
<a href="#">CopySubpix</a>	Copies pixel values between two images with subpixel precision
<a href="#">CopySubpixIntersect</a>	Copies pixel values of the intersection with specified window with subpixel precision
<a href="#">Dup</a>	Copies gray scale image to all channels of the color image
<a href="#">Transpose</a>	Transpose a source image
<a href="#">SwapChannels</a>	Changes the order of channels of the image
<a href="#">AddRandUniform_Direct</a>	Generates random samples with uniform distribution and adds them to an image data
<a href="#">AddRandGauss_Direct</a>	Generates random samples with Gaussian distribution and adds them to an image data

Function Base Name	Operation
<a href="#">ImageJaehne</a>	Creates the Jaehne's test image
<a href="#">ImageRamp</a>	Creates the test image that has an intensity ramp
<a href="#">SampleLine</a>	Reads all pixels on the raster line into the buffer
<a href="#">ZigzagFwd8x8</a>	Converts a conventional order to the zigzag sequence
<a href="#">ZigzagInv8x8</a>	Converts a zigzag sequence to the conventional order
<a href="#">ResampleRowGetSize</a>	Computes the sizes of the resample structure and the work buffer
<a href="#">ResampleRowInit</a>	Initializes the row resampling structure
<a href="#">ResampleRowGetBorderWidth</a>	Computes the width of the left and right borders
<a href="#">ResampleRow</a> , <a href="#">ResampleRowReplicateBorder</a>	Performs row-oriented resampling of the image

Many solutions and hints for use of these functions can be found in Intel® IPP Samples. See *Intel® IPP Image Processing and Media Processing Samples* downloadable from <http://www.intel.com/cd/software/products/asmo-na/eng/220046.htm>.

## Convert

*Converts image pixel values from one data type to another.*

### Syntax

#### Case 1: Conversion to increase bit depth and change signed to unsigned type

```
ippStatus ippiConvert_lu8u_C1R(const Ipp8u* pSrc, int srcStep, int srcBitOffset, Ipp8u* pDst, int dstStep, IppiSize roiSize);
```

```
ippStatus ippiConvert_<mod>(const Ipp<srcDatatype>* pSrc, int srcStep, Ipp<dstDatatype>* pDst, int dstStep, IppiSize roiSize);
```

Supported values for mod:

8u16u_C1R	8u16s_C1R	8u32s_C1R	8u32f_C1R	8s32s_C1R
8u16u_C3R	8u16s_C3R	8u32s_C3R	8u32f_C3R	8s32s_C3R
8u16u_C4R	8u16s_C4R	8u32s_C4R	8u32f_C4R	8s32s_C4R
8u16u_AC4R	8u16s_AC4R	8u32s_AC4R	8u32f_AC4R	8s32s_AC4R
8s32f_C1R	16u32s_C1R	16u32f_C1R	16s32s_C1R	16s32f_C1R
8s32f_C3R	16u32s_C3R	16u32f_C3R	16s32s_C3R	16s32f_C3R
8s32f_C4R	16u32s_C4R	16u32f_C4R	16s32s_C4R	16s32f_C4R
8s32f_AC4R	16u32s_AC4R	16u32f_AC4R	16s32s_AC4R	16s32f_AC4R
8s8u_C1Rs	16s16u_C1Rs	16u32u_C1R	32s32u_C1Rs	32u32f_C1R



8s16u_C1Rs	16s32u_C1Rs	32s32f_C1R
8s16s_C1R		
8s32u_C1Rs		

### Case 2: Conversion to reduce bit depth and change unsigned to signed type: integer to integer type

```
IppStatus ippiConvert_8ulu_C1R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst, int dstStep,
int dstBitOffset, IppiSize roiSize, Ipp8u threshold);
```

```
IppStatus ippiConvert_<mod>(const Ipp<scrDatatype>* pSrc, int srcStep, Ipp<dstDatatype>*
pDst, int dstStep, IppiSize roiSize);
```

Supported values for mod:

16u8u_C1R	16s8u_C1R	32s8u_C1R	32s8s_C1R
16u8u_C3R	16s8u_C3R	32s8u_C3R	32s8s_C3R
16u8u_C4R	16s8u_C4R	32s8u_C4R	32s8s_C4R
16u8u_AC4R	16s8u_AC4R	32s8u_AC4R	32s8s_AC4R

```
IppStatus ippiConvert_<mod>(const Ipp<scrDatatype>* pSrc, int srcStep, Ipp<dstDatatype>*
pDst, int dstStep, IppiSize roiSize, IppRoundMode roundMode, int scaleFactor);
```

Supported values for mod:

8u8s_C1RSfs	16u8s_C1RSfs	32u8u_C1RSfs	32s16u_C1RSfs
	16s8s_C1RSfs	32u8s_C1RSfs	32s16s_C1RSfs
	16u16s_C1RSfs	32u16u_C1RSfs	
		32u16s_C1RSfs	
		32u32s_C1RSfs	

### floating point to integer type

```
IppStatus ippiConvert_<mod>(const Ipp32f* pSrc, int srcStep, Ipp<dstDatatype>* pDst, int
dstStep, IppiSize roiSize, IppRoundMode roundMode);
```

Supported values for mod:

32f8u_C1R	32f8s_C1R	32f16u_C1R	32f16s_C1R
32f8u_C3R	32f8s_C3R	32f16u_C3R	32f16s_C3R
32f8u_C4R	32f8s_C4R	32f16u_C4R	32f16s_C4R
32f8u_AC4R	32f8s_AC4R	32f16u_AC4R	32f16s_AC4R

```
IppStatus ippiConvert_<mod>(const Ipp32f* pSrc, int srcStep, Ipp<dstDatatype>* pDst, int
dstStep, IppiSize roiSize, IppRoundMode roundMode, int scaleFactor);
```

Supported values for mod:

32f8u_C1RSfs	32f16u_C1RSfs	32f32u_C1RSfs
32f8s_C1RSfs	32f16s_C1RSfs	32f32s_C1RSfs

```
IppStatus ippiConvert_32f32u_C1IRSfs(Ipp32u* pSrcDst, int srcDstStep, IppiSize roiSize,
IppRoundMode roundMode, int scaleFactor);
```

## Parameters

<i>pSrc</i>	Pointer to the source image ROI.						
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.						
<i>srcBitOffset</i>	Offset (in bits) in the first byte of the source image row.						
<i>pDst</i>	Pointer to the destination image ROI.						
<i>pSrcDst</i>	Pointer to the source and destination image ROI for in-place operation.						
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.						
<i>srcDstStep</i>	Distance in bytes between starts of consecutive lines in the source and destination image for in-place operation.						
<i>dstBitOffset</i>	Offset (in bits) in the first byte of the destination image row.						
<i>roiSize</i>	Size of the source and destination ROI in pixels.						
<i>threshold</i>	Threshold level for Stucki's dithering.						
<i>scaleFactor</i>	Scale factor (see <a href="#">Integer Result Scaling</a> ).						
<i>roundMode</i>	Rounding mode, the following values are possible: <table> <tr> <td><code>ippRndZero</code></td><td>specifies that floating-point values are truncated toward zero,</td></tr> <tr> <td><code>ippRndNear</code></td><td>specifies that floating-point values are rounded to the nearest even integer when the fractional part equals 0.5; otherwise they are rounded to the nearest integer,</td></tr> <tr> <td><code>ippRndFinancial</code></td><td>specifies that floating-point values are rounded down to the nearest integer when the fractional part is less than 0.5, or rounded up to the nearest integer if the fractional part is equal or greater than 0.5.</td></tr> </table>	<code>ippRndZero</code>	specifies that floating-point values are truncated toward zero,	<code>ippRndNear</code>	specifies that floating-point values are rounded to the nearest even integer when the fractional part equals 0.5; otherwise they are rounded to the nearest integer,	<code>ippRndFinancial</code>	specifies that floating-point values are rounded down to the nearest integer when the fractional part is less than 0.5, or rounded up to the nearest integer if the fractional part is equal or greater than 0.5.
<code>ippRndZero</code>	specifies that floating-point values are truncated toward zero,						
<code>ippRndNear</code>	specifies that floating-point values are rounded to the nearest even integer when the fractional part equals 0.5; otherwise they are rounded to the nearest integer,						
<code>ippRndFinancial</code>	specifies that floating-point values are rounded down to the nearest integer when the fractional part is less than 0.5, or rounded up to the nearest integer if the fractional part is equal or greater than 0.5.						

## Description

The function `ippiConvert` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts pixel values in the source image ROI *pSrc* to a different data type and writes them to the destination image ROI *pDst*.

The result of integer operations is always saturated to the destination data type range. It means that if the value of the source pixel is out of the data range of the destination image, the value of the corresponding destination pixel is set to the value of the lower or upper bound (minimum or maximum) of the destination data range:

```
x = pSrc[i,j]
if (x > MAX_VAL) x = MAX_VAL
```

```
if (x < MIN_VAL) x = MIN_VAL
pDst[i,j] = (CASTING)x
```

If you want to shift data from the signed range to the unsigned range and vice-versa, see "Application Notes" below.

The function flavors with the *Sfs* descriptor in their names perform scaling of the internally computed results in accordance with the parameter *scaleFactor* (see [Integer Result Scaling](#)).

When converting from floating-point to integer type, rounding defined by *roundMode* is performed, and the result is saturated to the destination data type range

**1u to 8u conversion.** The source image has a 8u data type, where each byte represents eight consecutive pixels of the bitonal image (1 bit per pixel). In this case, additional parameter *srcBitOffset* is required to specify the start position of the source ROI buffer. Each source bit is transformed into the byte of the destination image in the following way: if an input pixel is 0, the corresponding output pixel is set to 0, if an input pixel is 1, the corresponding output pixel is set to 255. Note that in the source image the bit order in each byte is inverse relative to the pixel order, that is, the first pixel in a row is represented by the last (7th) bit of the first byte in the row.

**8u to 1u conversion.** Source image is converted to a bitonal image using the Stucki's error diffusion dithering algorithm. The destination image has a 8u data type, where each byte represents eight consecutive pixels of the bitonal image (1 bit per pixel). In this case, additional parameter *dstBitOffset* is required to specify the start position of the destination ROI buffer.



**CAUTION.** THE FUNCTION `ippiConvert_8ulu_C1R` IS DEPRECATED. Please use the function `ippiReduceBits_8ulu_C1R` instead.

## Application Notes

When data is converted from the signed integer to the corresponding unsigned integer and vice versa (8s --> 8u, 16u --> 16s), the pixel information may be lost because all negative values will be set to zero (signed-unsigned conversion), or unsigned values from the high half of the range will be set to the maximum value of the signed range (unsigned - signed conversion).

If you want to preserve complete pixel information, in other words, if you need just to shift the data from the signed range to the unsigned range and vice versa, use the function `ippiXorC` with the parameter *value* specified in such a way that the most significant bit is set to 1, and all other bits are set to 0. For example if you want to convert pixel values from `Ipps16s` type to `Ipp16u` type with the range shift call the function

```
ippiXorC_16u_C1R( (Ipp16u *)pSrc, srcStep, 0x8000, pDst, dstStep, roiSize);
```

In this case the pixels values are converted as follows:

```
-32768 --> 0
-32767 --> 1
...
-1 --> 32767
0 --> 32768
1 --> 32769
```

...

32766 --> 65534

32767 --> 65535

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error when any of the specified pointers is <code>NULL</code> , with the exception of second mode in Case 4.
<code>ippStsSizeErr</code>	Indicates an error condition if <code>roiSize</code> has a field with zero or negative value, or <code>srcBitOffset/dstBitOffset</code> is less than zero.
<code>ippStsStepErr</code>	Indicates an error condition if <code>srcStep</code> or <code>dstStep</code> has a zero or negative value.
<code>ippStsMemAllocErr</code>	Indicates an error condition if memory allocation fails.

## Example

The code example below shows data conversion without scaling.

```

IppStatus convert( void ) {
    IppiSize roi={5,4};
    Ipp32f x[5*4];

    Ipp8u y[5*4];
    ippiSet_32f_C1R( -1.0f, x, 5*sizeof(Ipp32f), roi );
    x[1] = 300; x[2] = 150;

    return ippiConvert_32f8u_C1R( x, 5*sizeof(Ipp32f), y, 5, roi, ippRndNear );
}

```

The destination image `y` contains:

```

00 FF 96 00 00
00 00 00 00 00
00 00 00 00 00
00 00 00 00 00

```

## Scale

*Scales pixel values of an image and converts them to another bit depth.*

---

### Syntax

#### Case 1: Scaling with conversion to integer data of increased bit depth

```

IppStatus ippiScale_<mod>(const Ipp<srcDatatype>* pSrc, int srcStep, Ipp<dstDatatype>*
pDst, int dstStep, IppiSize roiSize);

```

Supported values for `mod`:

<code>8u16u_C1R</code>	<code>8u16s_C1R</code>	<code>8u32s_C1R</code>
<code>8u16u_C3R</code>	<code>8u16s_C3R</code>	<code>8u32s_C3R</code>

8u16u_C4R	8u16s_C4R	8u32s_C4R
8u16u_AC4R	8u16s_AC4R	8u32s_AC4R

### Case 2: Scaling with conversion to floating-point data

```
IppStatus ippiScale_<mod>(const Ipp8u* pSrc, int srcStep, Ipp32f* pDst, int dstStep,
IppiSize roiSize, Ippi32f vMin, Ippi32f vMax);
```

Supported values for mod:

8u32f\_C1R  
8u32f\_C3R  
8u32f\_C4R  
8u32f\_AC4R

### Case 3: Scaling of integer data with conversion to reduced bit depth

```
IppStatus ippiScale_<mod> (const Ipp<srcDatatype>* pSrc, int srcStep, Ipp<dstDatatype>*
pDst, int dstStep, IppiSize roiSize, IppHintAlgorithm hint);
```

Supported values for mod:

16u8u_C1R	16s8u_C1R	32s8u_C1R
16u8u_C3R	16s8u_C3R	32s8u_C3R
16u8u_C4R	16s8u_C4R	32s8u_C4R
16u8u_AC4R	16s8u_AC4R	32s8u_AC4R

### Case 4: Scaling of floating-point data with conversion to integer data type

```
IppStatus ippiScale_<mod>(const Ipp32f* pSrc, int srcStep, Ipp8u* pDst, int dstStep,
IppiSize roiSize, Ippi32f vMin, Ippi32f vMax);
```

Supported values for mod:

32f8u\_C1R  
32f8u\_C3R  
32f8u\_C4R  
32f8u\_AC4R

## Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>roiSize</i>	Size of the source and destination ROI in pixels.
<i>vMin, vMax</i>	Minimum and maximum values of the input data.
<i>hint</i>	Option to select the algorithmic implementation of the function.

## Description

The function `ippiScale` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts pixel values of a source image ROI `pSrc` to the destination data type, using a linear mapping. The computation algorithm is specified by the `hint` argument (see [Table Hint Arguments for Image Moment Functions](#)). For conversion between integer data types, the whole range `[src_Min..src_Max]` of the input data type is mapped onto the range `[dst_Min..dst_Max]` of the output data type (see [Intel® Integrated Performance Primitives Concepts](#) for more information on data types and ranges).

The following scaling formula to map the source pixel  $p$  to the destination pixel  $p$  is used:

$$p = dst\_Min + k * (p - src\_Min)$$

where  $k = (dst\_Max - dst\_Min) / (src\_Max - src\_Min)$ .

For conversions to and from floating-point data type, the user-defined floating-point data range `[vMin..vMax]` is mapped onto the source or destination data type range.

If the conversion is from `Ipp32f` type and some of the input floating-point values are outside the specified input data range `[vMin..vMax]`, the corresponding output values will saturate. To determine the actual floating-point data range in your image, use the function `ippiMinMax`.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error when any of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <code>roiSize</code> has a field with a zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <code>srcStep</code> or <code>dstStep</code> has a zero or negative value.
<code>ippStsScaleRangeErr</code>	Indicates an error condition if the input data bounds are incorrect, that is, <code>vMax</code> is less than or equal to <code>vMin</code> .

## Example

The code example below shows how to use scaling to preserve the data range.

```

IppStatus scale( void ) {
    IppiSize roi = {5,4};
    Ipp32f x[5*4];
    Ipp8u y[5*4];
    ippiSet_32f_C1R( -1.0f, x, 5*sizeof(Ipp32f), roi );
    x[1] = 300; x[2] = 150;
    return ippiScale_32f8u_C1R( x, 5*sizeof(Ipp32f), y, 5, roi, -1, 300 );
}

```

The code example below shows how to use scaling to preserve the data range.

The destination image *y* contains:

```
00 FF 80 00 00
00 00 00 00 00
00 00 00 00 00
00 00 00 00 00
```

## Set

*Sets an array of pixels to a value.*

---

### Syntax

#### Case 1: Setting one-channel data to a value

```
IppStatus ippiSet_<mod>(Ipp<datatype> value, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);
```

Supported values for mod:

8u_C1R	16u_C1R	16s_C1R	32s_C1R	32f_C1R
--------	---------	---------	---------	---------

#### Case 2: Setting each color channel to a specified value

```
IppStatus ippiSet_<mod>(const Ipp<datatype>value[3], Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);
```

Supported values for mod:

8u_C3R	16u_C3R	16s_C3R	32s_C3R	32f_C3R
8u_AC4R	16u_AC4R	16s_AC4R	32s_AC4R	32f_AC4R

#### Case 3: Setting color channels and alpha channel to specified values

```
IppStatus ippiSet_<mod>(const Ipp<datatype>value[4], Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);
```

Supported values for mod:

8u_C4R	16u_C4R	16s_C4R	32s_C4R	32f_C4R
--------	---------	---------	---------	---------

#### Case 4: Setting masked one-channel data to a value

```
IppStatus ippiSet_<mod>(Ipp<datatype> value, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, const Ipp8u* pMask, int maskStep);
```

Supported values for mod:

8u_C1MR	16u_C1MR	16s_C1MR	32s_C1MR	32f_C1MR
---------	----------	----------	----------	----------

**Case 5: Setting color channels of masked multi-channel data to specified values**

```
IppStatus ippiset_<mod>(const Ipp<datatype>value[3], Ipp<datatype>* pDst, int dstStep,
IppiSize roiSize, const Ipp8u* pMask, int maskStep);
```

Supported values for mod:

8u_C3MR	16u_C3MR	16s_C3MR	32s_C3MR	32f_C3MR
8u_AC4MR	16u_AC4MR	16s_AC4MR	32s_AC4MR	32f_AC4MR

**Case 6: Setting all channels of masked multi-channel data to specified values**

```
IppStatus ippiset_<mod>(const Ipp<datatype>value[4], Ipp<datatype>* pDst, int dstStep,
IppiSize roiSize, const Ipp8u* pMask, int maskStep);
```

Supported values for mod:

8u_C4MR	16u_C4MR	16s_C4MR	32s_C4MR	32f_C4MR
---------	----------	----------	----------	----------

**Case 7: Setting selected channel of multi-channel data to a value**

```
IppStatus ippiset_<mod>(Ipp<datatype> value, Ipp<datatype>* pDst, int dstStep, IppiSize
roiSize);
```

Supported values for mod:

8u_C3CR	16u_C3CR	16s_C3CR	32s_C3CR	32f_C3CR
8u_C4CR	16u_C4CR	16s_C4CR	32s_C4CR	32f_C4CR

**Parameters**

<i>value</i>	Constant value assigned to each pixel in the destination image ROI.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>roiSize</i>	Size of the image ROI in pixels.
<i>pMask</i>	Pointer to the mask image buffer.
<i>maskStep</i>	Distance in bytes between starts of consecutive lines in the mask image buffer.

**Description**

The function `ippiset` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function sets pixels in the destination image ROI *pDst* to a constant *value*. Either all pixels in a rectangular ROI, or only those selected by the specified mask *pMask*, can be set to a value. In case of masked operation, the function sets pixel values in the destination buffer only if the spatially corresponding mask array value is non-zero. When a channel of interest is selected, that is only one channel of a multi-channel image must be set (see **Case 7**), the *pDst* pointer points to the start of ROI buffer in the required channel. If alpha channel is present in the source image data, the alpha components may be either skipped, or set to a value, depending on the chosen `ippiset` function flavor.



## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error when any of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <code>roiSize</code> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <code>dstStep</code> or <code>maskStep</code> has a zero or negative value.

## Example

The code example below shows how to use the function `ippiSet_8u_C1R`.

```
void func_set()
{
    IppiSize roi = {5,4};
    Ipp8u x[8*4] = {0};

    ippiSet_8u_C1R(1, x, 8, roi);
}
```

Result:

```
01 01 01 01 01 00 00 00
01 01 01 01 01 00 00 00
01 01 01 01 01 00 00 00
01 01 01 01 01 00 00 00
```

## Copy

*Copies pixel values between two buffers.*

### Syntax

#### Case 1: Copying all pixels of all color channels

```
IppStatus ippiCopy_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pDst, int
dstStep, IppiSize roiSize);
```

Supported values for `mod`:

<code>8u_C1R</code>	<code>16u_C1R</code>	<code>16s_C1R</code>	<code>32s_C1R</code>	<code>32f_C1R</code>
<code>8u_C3R</code>	<code>16u_C3R</code>	<code>16s_C3R</code>	<code>32s_C3R</code>	<code>32f_C3R</code>
<code>8u_C4R</code>	<code>16u_C4R</code>	<code>16s_C4R</code>	<code>32s_C4R</code>	<code>32f_C4R</code>
<code>8u_AC4R</code>	<code>16u_AC4R</code>	<code>16s_AC4R</code>	<code>32s_AC4R</code>	<code>32f_AC4R</code>
<code>8u_C3AC4R</code>	<code>16u_C3AC4R</code>	<code>16s_C3AC4R</code>	<code>32s_C3AC4R</code>	<code>32f_C3AC4R</code>
<code>8u_AC4C3R</code>	<code>16u_AC4C3R</code>	<code>16s_AC4C3R</code>	<code>32s_AC4C3R</code>	<code>32f_AC4C3R</code>

**Case 2: Operation on masked pixels only**

```
IppStatus ippiCopy_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pDst, int
dstStep, IppiSize roiSize, const Ipp8u* pMask, int maskStep);
```

Supported values for mod:

8u_C1MR	16u_C1MR	16s_C1MR	32s_C1MR	32f_C1MR
8u_C3MR	16u_C3MR	16s_C3MR	32s_C3MR	32f_C3MR
8u_C4MR	16u_C4MR	16s_C4MR	32s_C4MR	32f_C4MR
8u_ACM4R	16u_ACM4R	16s_ACM4R	32s_ACM4R	32f_ACM4R

**Case 3: Copying of a selected channel in a multi-channel image**

```
IppStatus ippiCopy_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pDst, int
dstStep, IppiSize roiSize);
```

Supported values for mod:

8u_C3CR	16u_C3CR	16s_C3CR	32s_C3CR	32f_C3CR
8u_C4CR	16u_C4CR	16s_C4CR	32s_C4CR	32f_C4CR

**Case 4: Copying of a selected channel to a one-channel image**

```
IppStatus ippiCopy_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pDst, int
dstStep, IppiSize roiSize);
```

Supported values for mod:

8u_C3C1R	16u_C3C1R	16s_C3C1R	32s_C3C1R	32f_C3C1R
8u_C4C1R	16u_C4C1R	16s_C4C1R	32s_C4C1R	32f_C4C1R

**Case 5: Copying a one-channel image to a multi-channel image**

```
IppStatus ippiCopy_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pDst, int
dstStep, IppiSize roiSize);
```

Supported values for mod:

8u_C1C3R	16u_C1C3R	16s_C1C3R	32s_C1C3R	32f_C1C3R
8u_C1C4R	16u_C1C4R	16s_C1C4R	32s_C1C4R	32f_C1C4R

**Case 6: Splitting color image into separate planes**

```
IppStatus ippiCopy_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* const
pDst[3], int dstStep, IppiSize roiSize);
```

Supported values for mod:

8u_C3P3R	16u_C3P3R	16s_C3P3R	32s_C3P3R	32f_C3P3R
----------	-----------	-----------	-----------	-----------

```
IppStatus ippiCopy_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* const
pDst[4], int dstStep, IppiSize roiSize);
```

Supported values for mod:

8u\_C4P4R      16u\_C4P4R      16s\_C4P4R      32s\_C4P4R      32f\_C4P4R

### Case 7: Composing color image from separate planes

```
IppStatus ippiCopy_<mod>(const Ipp<datatype>* const pSrc[3], int srcStep, Ipp<datatype>*
pDst, int dstStep, IppiSize roiSize);
```

Supported values for mod:

8u\_P3C3R      16u\_P3C3R      16s\_P3C3R      32s\_P3C3R      32f\_P3C3R

```
IppStatus ippiCopy_<mod>(const Ipp<datatype>* const pSrc[4], int srcStep, Ipp<datatype>*
pDst, int dstStep, IppiSize roiSize);
```

Supported values for mod:

8u\_P4C4R      16u\_P4C4R      16s\_P4C4R      32s\_P4C4R      32f\_P4C4R

## Parameters

<i>pSrc</i>	Pointer to the source image ROI. The array storing pointers to the color planes of the source planar image.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI. The array storing pointers to the color planes of the destination planar image.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>roiSize</i>	Size of the source and destination ROI in pixels.
<i>pMask</i>	Pointer to the mask image buffer.
<i>maskStep</i>	Distance in bytes between starts of consecutive lines in the mask image buffer.

## Description

The function `ippiCopy` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function copies data from a source image *pSrc* to the destination image *pDst*. Copying of pixels selected by a mask *pMask* is supported as well.

For masked operation (**Case 2**), the function writes pixel values in the destination buffer only if the spatially corresponding mask array value is non-zero (as illustrated in the code example below).

Function flavors operating with the channel of interest (descriptor *C*) copy only one specified channel of a source multi-channel image to the channel of another multi-channel image (see **Case 3**). For these functions, the *pSrc* and *pDst* pointers point to the starts of ROI buffers in the specified channels of source and destination images, respectively.

Some function flavors add alpha channel to the 3-channel source image (flavors with the `_C3AC4R` descriptor), or discard alpha channel from the source image (flavors with the `_AC4C3R` descriptor) - see **Case 1**.

Special function flavors copy data from only one specified channel `pSrc` of a multi-channel image to a one-channel image `pDst` (see **Case 4**), as well as to copy data from a one-channel image `pSrc` to only one specified channel of a multi-channel image `pDst` (see **Case 5**).

The function `ippiCopy` can also be used to convert the interleaved color image into separate planes and vice versa (see **Case 6** and **Case 7**).

The function `ippiCopy_8u_C1R` is used in H.264 decoder and encoder included into Intel® IPP Samples downloadable from <http://www.intel.com/cd/software/products/asmo-na/eng/perflib/ipp/index.htm>

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error when any of the specified pointers is <code>NULL</code> , with the exception of second mode in <b>Case 4</b> .
<code>ippStsSizeErr</code>	Indicates an error condition if <code>roiSize</code> has a field with a zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <code>srcStep</code> or <code>dstStep</code> is less than <code>roiSize.width * &lt;pixelSize&gt;</code> for <b>Cases 4</b> and <b>5</b> .

## Example

The code example below shows how to copy masked data.

```

IppStatus copyWithMask( void ) {
    Ipp8u mask[3*3], x[5*4], y[5*4]={0};
    IppiSize imgroi={5,4}, mskroi={3,3};
    ippiSet_8u_C1R( 3, x, 5, imgroi );
    /// set mask with a hole in upper left corner
    ippiSet_8u_C1R( 1, mask, 3, mskroi );
    mask[0] = 0;
    /// copy roi with mask
    return ippiCopy_8u_C1MR( x, 5, y, 5, mskroi, mask, 3 );
}

```

The destination image `y` contains:

```

00 03 03 00 00
03 03 03 00 00
03 03 03 00 00
00 00 00 00 00

```

## CopyManaged

*Copies pixel values between two images in accordance with the specified type of copying.*

---

### Syntax

```

IppStatus ippiCopyManaged_8u_C1R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst, int dstStep,
IppiSize roiSize, int flags);

```

## Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>roiSize</i>	Size of the image ROI in pixels.
<i>flags</i>	Specifies the type of copying. Possible values are: <ul style="list-style-type: none"> <li>IPP_TEMPORAL_COPY - standard copying</li> <li>IPP_NONTEMPORAL_STORE - copying without caching the destination image.</li> <li>IPP_NONTEMPORAL_LOAD - processor uses non-temporal load instruction</li> </ul>

## Description

The function `ippiCopyManaged` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function copies data from a source image ROI *pSrc* to the destination image ROI *pDst*. The parameter *flags* specifies the type of copying that the function performs. When *flags* is set to `IPP_TEMPORAL_COPY`, the function is identical to the function `ippiCopy_8u_C1R`. When *flags* is set to `IPP_NONTEMPORAL_STORE`, the processor uses non-temporal store instructions, and operation is performed without caching the data of the destination image.

To achieve maximum performance, align data to the 64-byte boundary.

### Optimization Notice

Intel's compilers may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include SSE2, SSE3, and SSSE3 instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on microprocessors not manufactured by Intel. Microprocessor-dependent optimizations in this product are intended for use with Intel microprocessors. Certain optimizations not specific to Intel microarchitecture are reserved for Intel microprocessors. Please refer to the applicable product User and Reference Guides for more information regarding the specific instruction sets covered by this notice.

Notice revision #20110804

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition one of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with a zero or negative value.

## CopyConstBorder

*Copies pixels values between two images and adds the border pixels with a constant value.*

---

### Syntax

#### Case 1: Operation on one-channel data

```
IppStatus ippiCopyConstBorder_<mod>(const Ipp<datatype>* pSrc, int srcStep, IppiSize srcRoiSize, Ipp<datatype>* pDst, int dstStep, IppiSize dstRoiSize, int topBorderHeight, int leftBorderWidth, Ipp<datatype> value);
```

Supported values for mod:

8u_C1R	16u_C1R	16s_C1R	32s_C1R	32f_C1R
--------	---------	---------	---------	---------

#### Case 2: Operation on multi-channel data

```
IppStatus ippiCopyConstBorder_<mod>(const Ipp<datatype>* pSrc, int srcStep, IppiSize srcRoiSize, Ipp<datatype>* pDst, int dstStep, IppiSize dstRoiSize, int topBorderHeight, int leftBorderWidth, const Ipp<datatype> value[3]);
```

Supported values for mod:

8u_C3R	16u_C3R	16s_C3R	32s_C3R	32f_C3R
8u_AC4R	16u_AC4R	16s_AC4R	32s_AC4R	32f_AC4R

```
IppStatus ippiCopyConstBorder_<mod>(const Ipp<datatype>* pSrc, int srcStep, IppiSize srcRoiSize, Ipp<datatype>* pDst, int dstStep, IppiSize dstRoiSize, int topBorderHeight, int leftBorderWidth, const Ipp<datatype> value[4]);
```

Supported values for mod:

8u_C4R	16u_C4R	16s_C4R	32s_C4R	32f_C4R
--------	---------	---------	---------	---------

### Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>srcRoiSize</i>	Size of the source ROI in pixels.
<i>dstRoiSize</i>	Size of the destination ROI in pixels.
<i>topBorderHeight</i>	Height of the top border in pixels.
<i>leftBorderWidth</i>	Width of the left border in pixels.
<i>value</i>	The constant value to assign to the border pixels (constant vector in case of multi-channel images).

## Description

The function `ippiCopyConstBorder` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function copies the source image *pSrc* to the destination image *pDst* and creates border outside the copied area; pixel values of the border are set to the specified constant value that is passed by the *value* argument. The parameters *topBorderHeight* and *leftBorderWidth* specify the position of the first pixel of the source ROI in the destination image ROI (see Figure Creating a Border of Pixels with Constant Value.) Squares marked in red correspond to pixels copied from the source image, that is, the source image ROI.

## Creating a Border of Pixels with Constant Value

v	v	v	v	v	v	v	v	v
v	v	v	v	v	v	v	v	v
v	v	1	2	3	4	5	v	v
v	v	6	7	8	9	10	v	v
v	v	11	12	13	14	15	v	v
v	v	16	17	18	19	20	v	v
v	v	v	v	v	v	v	v	v

*topBorderHeight=2*  
*leftBorderWidth=2*

The height (width) of the destination ROI cannot be less than the sum of the height (width) of source ROI and the *topBorderHeight* (*leftBorderWidth*) parameter.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error when any of the specified pointers is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>srcRoiSize</i> or <i>dstRoiSize</i> has a field with a zero or negative value, or <i>topBorderHeight</i> or <i>leftBorderWidth</i> is less than zero, or <i>dstRoiSize.width</i> < <i>srcRoiSize.width</i> + <i>leftBorderWidth</i> , or <i>dstRoiSize.height</i> < <i>srcRoiSize.height</i> + <i>topBorderHeight</i> .

`ippiStsStepErr` Indicates an error condition if `srcStep` or `dstStep` has a zero or negative value.

## Example

The code example below shows how to use the function `ippiCopyConstBorder_8u_C1R`.

```

Ipp8u src[8*4] = {3, 3, 3, 3, 3, 8, 8, 8,
                  3, 2, 1, 2, 3, 8, 8, 8,
                  3, 2, 1, 2, 3, 8, 8, 8,
                  3, 3, 3, 3, 3, 8, 8, 8};

Ipp8u dst[8*6];
IppiSize srcRoi = { 5, 4 };
IppiSize dstRoi = { 7, 6 };
int borderWidth = 1;
int borderHeight = 1;
int borderVal = 0;

ippiCopyConstBorder_8u_C1R(src, 8, srcRoi, dst, 8, dstRoi, borderHeight, borderWidth, borderVal);

```

*Results*

source image:

```

3 3 3 3 3 8 8 8
3 2 1 2 3 8 8 8
3 2 1 2 3 8 8 8   src
3 3 3 3 3 8 8 8

```

destination image:

```

0 0 0 0 0 0 0
0 3 3 3 3 3 0
0 3 2 1 2 3 0
0 3 2 1 2 3 0   dst
0 3 3 3 3 3 0
0 0 0 0 0 0 0

```

## CopyReplicateBorder

*Copies pixels values between two images and adds the replicated border pixels.*

---

### Syntax

#### Case 1: Not-in-place operation

```

IppStatus ippiCopyReplicateBorder_<mod>(const Ipp<datatype>* pSrc, int srcStep, IppiSize
srcRoiSize, Ipp<datatype>* pDst, int dstStep, IppiSize dstRoiSize, int topBorderHeight,
int leftBorderWidth);

```

Supported values for `mod`:

8u_C1R	16u_C1R	16s_C1R	32s_C1R	32f_C1R
8u_C3R	16u_C3R	16s_C3R	32s_C3R	32f_C3R
8u_C4R	16u_C4R	16s_C4R	32s_C4R	32f_C4R
8u_AC4R	16u_AC4R	16s_AC4R	32s_AC4R	32f_AC4R



## Case 2: In-place operation

```
IppStatus ippiCopyReplicateBorder_<mod>(const Ipp<datatype>* pSrc, int srcDstStep, IppiSize srcRoiSize, IppiSize dstRoiSize, int topBorderHeight, int leftBorderWidth);
```

Supported values for mod:

8u_C1IR	16u_C1IR	16s_C1IR	32s_C1IR	32f_C1IR
8u_C3IR	16u_C3IR	16s_C3IR	32s_C3IR	32f_C3IR
8u_C4IR	16u_C4IR	16s_C4IR	32s_C4IR	32f_C4IR
8u_AC4IR	16u_AC4IR	16s_AC4IR	32s_AC4IR	32f_AC4IR

## Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in destination image.
<i>srcDstStep</i>	Distance in bytes between starts of consecutive lines in the source and the destination image.
<i>srcRoiSize</i>	Size of the source ROI in pixels.
<i>dstRoiSize</i>	Size of the destination ROI in pixels.
<i>topBorderHeight</i>	Height of the top border in pixels.
<i>leftBorderWidth</i>	Width of the left border in pixels.

## Description

The function `ippiCopyReplicateBorder` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function copies the source image *pSrc* to the destination image *pDst* and fills pixels ("border") outside the copied area in the destination image with the values of the source image pixels according to the scheme illustrated in Figure Creating a Replicated Border. Squares marked in red correspond to pixels copied from the source image, that is, the source image ROI.

Note that the in-place function flavor actually adds border pixels to the source image ROI, thus a destination image ROI is larger than the initial image.

The parameters *topBorderHeight* and *leftBorderWidth* specify the position of the first pixel of the source ROI in the destination image ROI.

### Creating a Replicated Border

1	1	1	2	3	4	5	5	5
1	1	1	2	3	4	5	5	5
1	1	1	2	3	4	5	5	5
6	6	6	7	8	9	10	10	10
11	11	11	12	13	14	15	15	15
16	16	16	17	18	19	20	20	20
16	16	16	17	18	19	20	20	20

*topBorderHeight=2*  
*leftBorderWidth=2*

The height (width) of the destination ROI cannot be less than the sum of the height (width) of source ROI and the *topBorderHeight* (*leftBorderWidth*) parameter.

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error when any of the specified pointers is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>srcRoiSize</i> or <i>dstRoiSize</i> has a field with a zero or negative value, or <i>topBorderHeight</i> or <i>leftBorderWidth</i> is less than zero, or <i>dstRoiSize.width</i> < <i>srcRoiSize.width</i> + <i>leftBorderWidth</i> , or <i>dstRoiSize.height</i> < <i>srcRoiSize.height</i> + <i>topBorderHeight</i> .
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> or <i>dstStep</i> has a zero or negative value.

### Example

The code example below shows how to use the `ippiCopyReplicateBorder_8u_C1R` function.

```

Ipp8u src[8*4] = {5, 4, 3, 4, 5, 8, 8, 8,
                  3, 2, 1, 2, 3, 8, 8, 8,
                  3, 2, 1, 2, 3, 8, 8, 8,
                  5, 4, 3, 4, 5, 8, 8, 8};
Ipp8u dst[9*8];
IppiSize srcRoi = { 5, 4 };
IppiSize dstRoi = { 9, 8 };
int topborderHeight = 2;

```

```
int leftborderWidth = 2;

ippiCopyReplicateBorder_8u_C1R(src, 8, srcRoi, dst, 9, dstRoi, topBorderHeight, leftBorderWidth);
```

*Results*

source image:

```
5 4 3 4 5 8 8 8
3 2 1 2 3 8 8 8
3 2 1 2 3 8 8 8
5 4 3 4 5 8 8 8
```

destination image:

```
5 5 5 4 3 4 5 5 5
5 5 5 4 3 4 5 5 5
5 5 5 4 3 4 5 5 5
3 3 3 2 1 2 3 3 3
3 3 3 2 1 2 3 3 3
5 5 5 4 3 4 5 5 5
5 5 5 4 3 4 5 5 5
5 5 5 4 3 4 5 5 5
```

## CopyWrapBorder

*Copies pixels values between two images and adds the border pixels.*

---

### Syntax

#### Case 1: Not-in-place operation

```
IppStatus ippiCopyWrapBorder_32s_C1R(const Ipp32s* pSrc, int srcStep, IppiSize srcRoiSize,
Ipp32s* pDst, int dstStep, IppiSize dstRoiSize, int topBorderHeight, int leftBorderWidth);

IppStatus ippiCopyWrapBorder_32f_C1R(const Ipp32f* pSrc, int srcStep, IppiSize srcRoiSize,
Ipp32f* pDst, int dstStep, IppiSize dstRoiSize, int topBorderHeight, int leftBorderWidth);
```

#### Case 2: In-place operation

```
IppStatus ippiCopyWrapBorder_32s_C1IR(const Ipp32s* pSrc, int srcDstStep, IppiSize
srcRoiSize, IppiSize dstRoiSize, int topBorderHeight, int leftBorderWidth);

IppStatus ippiCopyWrapBorder_32f_C1IR(const Ipp32f* pSrc, int srcDstStep, IppiSize
srcRoiSize, IppiSize dstRoiSize, int topBorderHeight, int leftBorderWidth);
```

### Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>srcDstStep</i>	Distance in bytes between starts of consecutive lines in the source and destination image for in-place flavor.

<i>srcRoiSize</i>	Size of the source ROI in pixels.
<i>dstRoiSize</i>	Size of the destination ROI in pixels.
<i>topBorderHeight</i>	Height of the top border in pixels.
<i>leftBorderWidth</i>	Width of the left border in pixels.

## Description

The function `ippiCopyWrapBorder` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function copies the source image *pSrc* to the destination image *pDst* and fills pixels ("border") outside the copied area in the destination image with the values of the source image pixels according to the scheme illustrated in Figure Creating a Border of Pixels by `ippiCopyWrapBorder` Function. Squares marked in red correspond to pixels copied from the source image.

Note that the in-place function flavor actually adds border pixels to the source image ROI, thus a destination image ROI is larger than the initial image.

The parameters *topBorderHeight* and *leftBorderWidth* specify the position of the first pixel of the source ROI in the destination image ROI.

## Creating a Border of Pixels by `ippiCopyWrapBorder` Function

14	15	11	12	13	14	15	11	12	13	14	15	11
19	20	16	17	18	19	20	16	17	18	19	20	16
4	5	1	2	3	4	5	1	2	3	4	5	1
9	10	6	7	8	9	10	6	7	8	9	10	6
14	15	11	12	13	14	15	11	12	13	14	15	11
19	20	16	17	18	19	20	16	17	18	19	20	16
4	5	1	2	3	4	5	1	2	3	4	5	1
9	10	6	7	8	9	10	6	7	8	9	10	6
14	15	11	12	13	14	15	11	12	13	14	15	11
19	20	16	17	18	19	20	16	17	18	19	20	16
4	5	1	2	3	4	5	1	2	3	4	5	1
9	10	6	7	8	9	10	6	7	8	9	10	6

*topBorderHeight*=2  
*leftBorderWidth*=2

The height (width) of the destination ROI cannot be less than the sum of the height (width) of source ROI and the *topBorderHeight* (*leftBorderWidth*) parameter.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error when any of the specified pointers is <code>NULL</code> .

<code>ippStsSizeErr</code>	Indicates an error condition if <code>srcRoiSize</code> or <code>dstRoiSize</code> has a field with a zero or negative value, or <code>topBorderHeight</code> or <code>leftBorderWidth</code> is less than zero, or <code>dstRoiSize.width &lt; srcRoiSize.width + leftBorderWidth</code> , or <code>dstRoiSize.height &lt; srcRoiSize.height + topBorderHeight</code> .
<code>ippStsStepErr</code>	Indicates an error condition if <code>srcStep</code> or <code>dstStep</code> has a zero or negative value.
<code>ippStsNotEvenStepErr</code>	Indicates an error condition if one of the step values is not divisible by 4 for floating point images, or by 2 for short integer images.

### Example

The code example below shows how to use the `ippiCopyWrapBorder_32s_C1R` function.

```

Ipp32s src[8*4] = {
    5, 4, 3, 4, 5, 8, 8, 8,
    3, 2, 1, 2, 3, 8, 8, 8,
    3, 2, 1, 2, 3, 8, 8, 8,
    5, 4, 3, 4, 5, 8, 8, 8
};
Ipp32s dst[9*8];
IppiSize srcRoi = { 5, 4 };
IppiSize dstRoi = { 9, 8 };
int topborderHeight = 2;
int leftborderWidth = 2;

ippiCopyWrapBorder_32s_C1R (src, 8*sizeof(Ipp32s), srcRoi, dst,
                             9*sizeof(Ipp32s), dstRoi, topborderHeight, leftborderWidth);

```

#### Results

source image:

```

5 4 3 4 5 8 8 8
3 2 1 2 3 8 8 8
3 2 1 2 3 8 8 8
5 4 3 4 5 8 8 8

```

destination image:

```

2 3 3 2 1 2 3 3 2
4 5 5 4 3 4 5 5 4
4 5 5 4 3 4 5 5 4
2 3 3 2 1 2 3 3 2
2 3 3 2 1 2 3 3 2
4 5 5 4 3 4 5 5 4
4 5 5 4 3 4 5 5 4
2 3 3 2 1 2 3 3 2

```

## CopySubpix

*Copies pixel values between two images with subpixel precision.*

---

### Syntax

#### Case 1: Copying without conversion or with conversion to floating point data

```
IppStatus ippiCopySubpix_<mod>(const Ipp<srcDatatype>* pSrc, int srcStep, Ipp<dstDatatype>* pDst, int dstStep, IppiSize roiSize, Ipp32f dx, Ipp32f dy);
```

Supported values for mod:

8u_C1R	16u_C1R	32f_C1R
8u32f_C1R	16u32f_C1R	

#### Case 2: Copying with conversion to integer data

```
IppStatus ippiCopySubpix_8u16u_C1R_Sfs(const Ipp8u* pSrc, int srcStep, Ipp16u* pDst, int dstStep, IppiSize roiSize, Ipp32f dx, Ipp32f dy, int scaleFactor);
```

### Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>roiSize</i>	Size of the source and destination ROI in pixels.
<i>dx</i>	Fractional part of the <i>x</i> -coordinate in the source image.
<i>dy</i>	Fractional part of the <i>y</i> -coordinate in the source image.
<i>scaleFactor</i>	Scale factor (see <a href="#">Integer Result Scaling</a> ).

### Description

The function `ippiCopySubpix` is declared in the `ippcv.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function computes the pixel value of the destination image using linear interpolation (see [Linear Interpolation in Appendix B](#)) in accordance with the following formula:

$$pDst_{j,i} = pSrc_{j+dx,i+dy}$$

where  $i = 0, \dots, roiSize.height - 1$ ,  $j = 0, \dots, roiSize.width - 1$ .

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error when any of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <code>roiSize</code> has a field with a zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <code>srcStep</code> or <code>dstStep</code> is less than <code>roiSize.width * &lt;pixelSize&gt;</code> .
<code>ippStsNotEvenStepErr</code>	Indicates an error condition if one of <code>srcStep</code> or <code>dstStep</code> is not divisible by 4 for floating point images, or by 2 for short integer images.

## CopySubpixIntersect

*Copies pixel values of the intersection with specified window with subpixel precision.*

### Syntax

#### Case 1: Copying without conversion or with conversion to floating point data

```
IppStatus ippCopySubpixIntersect_<mod>(const Ipp<srcDatatype>* pSrc, int srcStep, IppiSize srcRoiSize, Ipp<dstDatatype>* pDst, int dstStep, IppiSize dstRoiSize, IppiPoint_32f point, IppiPoint* pMin, IppiPoint* pMax);
```

Supported values for `mod`:

<code>8u_C1R</code>	<code>16u_C1R</code>	<code>32f_C1R</code>
<code>8u32f_C1R</code>	<code>16u32f_C1R</code>	

#### Case 2: Copying with conversion to integer data

```
IppStatus ippCopySubpixIntersect_8u16u_C1R_Sfs(const Ipp8u* pSrc, int srcStep, IppiSize srcRoiSize, Ipp16u* pDst, int dstStep, IppiSize dstRoiSize, IppiPoint_32f point, IppiPoint* pMin, IppiPoint* pMax, int scaleFactor);
```

### Parameters

<code>pSrc</code>	Pointer to the source image ROI.
<code>srcStep</code>	Distance in bytes between starts of consecutive lines in the source image.
<code>srcRoiSize</code>	Size of the source image ROI in pixels.
<code>pDst</code>	Pointer to the destination image ROI.
<code>dstStep</code>	Distance in bytes between starts of consecutive lines in the source and destination image.
<code>dstRoiSize</code>	Size of the destination ROI in pixels.
<code>point</code>	Center point of the window.
<code>pMin</code>	Pointer to coordinates of the top left pixel of the intersection in the destination image.

*pMax* Pointer to coordinates of the bottom right pixel of the intersection in the destination image.

*scaleFactor* Scale factor (see [Integer Result Scaling](#)).

## Description

The function `ippiCopySubpixIntersect` is declared in the `ippcv.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function determines the intersection of the source image and the window of size `dstRoiSize` centered in point `point`. The corresponding pixels of the destination image are calculated using linear interpolation (see [Linear Interpolation](#) in Appendix B) in accordance with the following formula:

$$pDst_{j,i} = pSrc_{Xsubpix(j),Ysubpix(i)}$$

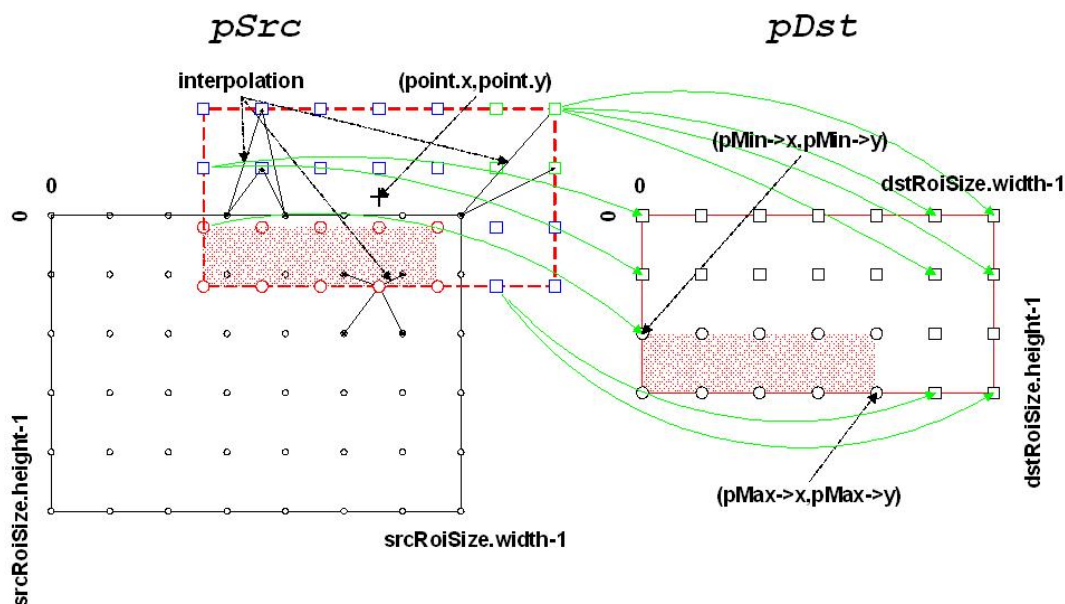
where  $Xsubpix(j) = \min(\max(\text{point.x} + j - 0.5 * (\text{dstRoiSize.width} - 1), 0), \text{srcRoiSize.width} - 1)$ ,

$Ysubpix(i) = \min(\max(\text{point.y} + i - 0.5 * (\text{dstRoiSize.height} - 1), 0), \text{srcRoiSize.height} - 1)$ ,

$i = 0, \dots, \text{dstRoiSize.height} - 1, j = 0, \dots, \text{dstRoiSize.width} - 1$ .

Minimal values of  $j$  and  $i$  (coordinates of the top left calculated destination pixel) are assigned to `pMin.x` and `pMin.y`, maximal values (coordinates of the top left calculated destination pixel) - to `pMin.x` and `pMin.y`. (See Figure Image Copying with Subpixel Precision Using `ippiCopySubpixIntersect` Function.)

## Image Copying with Subpixel Precision Using `ippiCopySubpixIntersect` Function





The height (width) of the destination ROI cannot be less than the sum of the height (width) of source ROI and the *topBorderHeight* (*leftBorderWidth*) parameter.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error when any of the specified pointers is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>srcRoiSize</i> or <i>dstRoiSize</i> has a field with a zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> is less than <i>srcRoiSize.width</i> * <i>&lt;pixelSize&gt;</i> , or <i>dstStep</i> is less than <i>dstRoiSize.width</i> * <i>&lt;pixelSize&gt;</i> .

## Example

The code example below shows how to use the function `ippiCopySubpixIntersect_8u_C1R`.

```

Ipp8u src[8*6] = {
    7, 7, 6, 6, 6, 6, 7, 7,
    6, 5, 5, 5, 5, 5, 5, 6,
    6, 5, 4, 3, 3, 4, 5, 6,
    6, 5, 4, 3, 3, 4, 5, 6,
    6, 5, 5, 5, 5, 5, 5, 6,
    6, 6, 6, 6, 6, 6, 6, 6
};
Ipp8u dst[7*4];
IppiSize srcRoi = { 8, 6 };
IppiSize dstRoi = { 7, 4 };
IppiPoint_32f point = { 4, 1 };
IppiPoint min;
IppiPoint max;

ippiCopySubpixIntersect_8u_C1R (src, 8, srcRoi, dst, 7, dstRoi, point, &min, &max );

```

*Results*

source image:

```

7 7 6 6 6 6 7 7
6 5 5 5 5 5 5 6
6 5 4 3 3 4 5 6
6 5 4 3 3 4 5 6
6 5 5 5 5 5 5 6
6 6 6 6 6 6 6 6

```

destination image:

```

7 6 6 6 6 7 7
6 6 6 6 6 6 7
5 5 4 4 5 5 6
5 4 3 3 4 5 6

```

min = { 0, 1 }  
max = { 5, 3 }

## Dup

*Copies a gray scale image to all channels of the color image.*

---

### Syntax

```
IppStatus ippiDup_8u_C1C3R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst, int dstStep, IppiSize roiSize);
```

```
IppStatus ippiDup_8u_C1C4R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst, int dstStep, IppiSize roiSize);
```

### Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>roiSize</i>	Size of the source and destination ROI in pixels.

### Description

The function `ippiDup` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)). This function copies a one-channel (gray scale) image *pSrc* to each channel of the multi-channel image *pDst*.

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error when any of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <i>RoiSize</i> has a field with a zero or negative value.

## Transpose

*Transposes a source image.*

---

### Syntax

#### Case 1: Not-in-place operation

```
IppStatus ippiTranspose_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);
```

Supported values for `mod`:

8u_C1R	16u_C1R	16s_C1R	32s_C1R	32f_C1R
8u_C3R	16u_C3R	16s_C3R	32s_C3R	32f_C3R

8u_C4R	16u_C4R	16s_C4R	32s_C4R	32f_C4R
--------	---------	---------	---------	---------

### Case 2: In-place operation

```
IppStatus ippiTranspose_<mod>(const Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize);
```

Supported values for mod:

8u_C1IR	16u_C1IR	16s_C1IR	32s_C1IR	32f_C1IR
8u_C3IR	16u_C3IR	16s_C3IR	32s_C3IR	32f_C3IR
8u_C4IR	16u_C4IR	16s_C4IR	32s_C4IR	32f_C4IR

### Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>pSrcDst</i>	Pointer to the source and destination ROI for in-place operation.
<i>srcDstStep</i>	Distance in bytes between starts of consecutive lines in the source and destination image buffer for the in-place operation.
<i>roiSize</i>	Size of the source ROI in pixels.

### Description

The function `ippiTranspose` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function transposes the source image *pSrc* (*pSrcDst* for in-place flavors) and stores the result in *pDst* (*pSrcDst*). The destination image is obtained from the source image by transforming the columns to the rows, that is,  $pDst(x,y) = pSrc(y,x)$ .

The parameter *roiSize* is specified for the source image; therefore *roiSize.width* for the destination image is equal to *roiSize.height* for the source image, and *roiSize.height* for the destination image is equal to *roiSize.width* for the source image.



**NOTE.** For in-place operations, *roiSize.width* must be equal to *roiSize.height*.

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error when any of the specified pointers is <code>NULL</code> , with the exception of second mode in Case 4.

`ippStsSizeErr`

Indicates an error condition if *roiSize* has a field with a zero or negative value, or *roiSize.width* is not equal to *roiSize.height* for in-place flavors.

## Example

The code example below shows how to use the function `ippiTranspose_8u_C1R`.

```

Ipp8u src[8*4] = {1, 2, 3, 4, 8, 8, 8, 8,
                  1, 2, 3, 4, 8, 8, 8, 8,
                  1, 2, 3, 4, 8, 8, 8, 8,
                  1, 2, 3, 4, 8, 8, 8, 8};
Ipp8u dst[4*4];
IppiSize srcRoi = { 4, 4 };

ippiTranspose_8u_C1R ( src, 8, dst, 4, srcRoi );

```

Result:

```

1 2 3 4 8 8 8 8
1 2 3 4 8 8 8 8   src
1 2 3 4 8 8 8 8

```

```

1 1 1 1
2 2 2 2
3 3 3 3   dst
4 4 4 4

```

## SwapChannels

*Copies channels of the source image to the destination image.*

---

### Syntax

#### Case 1: Not-in-place operation

```

IppStatus ippiSwapChannels_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>*
pDst, int dstStep, IppiSize roiSize, const int dstOrder[3]);

```

Supported values for *mod*:

8u_C3R	16u_C3R	16s_C3R	32s_C3R	32f_C3R
8u_AC4R	16u_AC4R	16s_AC4R	32s_AC4R	32f_AC4R

#### Case 2: In-place operation

```

IppStatus ippiSwapChannels_8u_C3IR(Ipp8u* pSrcDst, int srcDstStep, IppiSize roiSize, const
int dstOrder[3]);

```

```

IppStatus ippiSwapChannels_8u_C4IR(Ipp8u* pSrcDst, int srcDstStep, IppiSize roiSize, const
int dstOrder[4]);

```

**Case 3: Operation with converting 3-channel image to the 4-channel image**

```
IppStatus ippiSwapChannels_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>*
pDst, int dstStep, IppiSize roiSize, const int dstOrder[4], Ipp<datatype> val);
```

Supported values for mod:

8u\_C3C4R      16u\_C3C4R      16s\_C3C4R      32s\_C3C4R      32f\_C3C4R

**Case 4: Operation with converting 4-channel image to the 3-channel image**

```
IppStatus ippiSwapChannels_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>*
pDst, int dstStep, IppiSize roiSize, const int dstOrder[3]);
```

Supported values for mod:

8u\_C4C3R      16u\_C4C3R      16s\_C4C3R      32s\_C4C3R      32f\_C4C3R

**Parameters**

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>pSrcDst</i>	Pointer to the source and destination ROI for in-place operation.
<i>srcDstStep</i>	Distance in bytes between starts of consecutive lines in the source and destination image for the in-place operation.
<i>roiSize</i>	Size of the source and destination ROI in pixels.
<i>dstOrder</i>	Order of channels in the destination image.
<i>val</i>	Constant value.

**Description**

The function `ippiSwapChannels` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function copies the data from specified channels of the source image ROI *pSrc* to the specified channels of the destination image ROI *pDst*.

The first channel in the destination image is determined by the first component of *dstOrder*. Its value lies in the range [0..2] for a 3-channel image, and [0..3] for a 4-channel image, and indicates the corresponding channel number of the source image. Other channels are specified in the similar way. For example, if the sequence of channels in the source 3-channel image is *A*, *B*, *C*, and *dstOrder*[0]=2, *dstOrder*[1]=0, *dstOrder*[2]=1, then the order of channels in the 3-channel destination image is *C*, *A*, *B*. Some or all components of *dstOrder* may have the same values. It means that data from a certain channel of the source image may be copied to several channels of the destination image.

Some functions flavors convert a 3-channel source image to the 4-channel destination image (see [Case 3](#)). In this case an additional channel contains data from any specified source channel, or its pixel values are set to the specified constant value *val* (corresponding component *dstOrder*[*n*] should be set to 3), or its pixel values

are not set at all (corresponding component *dstOrder[n]* should be set to an arbitrary value greater than 3). For example, the sequence of channels in the source 3-channel image is *A, B, C*, if *dstOrder[0]=1*, *dstOrder[1]=0*, *dstOrder[2]=1*, *dstOrder[3]=2*, then the order of channels in the 4-channel destination image will be *B, A, B, C*; if *dstOrder[0]=4*, *dstOrder[1]=0*, *dstOrder[2]=1*, *dstOrder[3]=2*, then the order of channels in the 4-channel destination image will be *D, A, B, C*, where *D* is a channel whose pixel values are not set.

The function flavors that support image with the alpha channel do not perform operation on it.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error when any of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <i>dstStep</i> or <i>maskStep</i> has a zero or negative value.

## Example

The code example below shows how to use the function `ippiSwapChannels_8u_C3R`.

```

Ipp8u src[12*3] = { 255, 0, 0, 255, 0, 0, 255, 0, 0, 255, 0, 0,
                   0, 255, 0, 0, 255, 0, 0, 255, 0, 0, 255, 0,
                   0, 0, 255, 0, 0, 255, 0, 0, 255, 0, 0, 255};

Ipp8u dst[12*3];
IppiSize roiSize = { 4, 3 };
int order[3] = { 2, 1, 0 };

ippiSwapChannels_8u_C3R ( src, 12, dst, 12, roiSize, order );

```

Result:

src [rgb]

```

255 0 0 255 0 0 255 0 0 255 0 0
0 255 0 0 255 0 0 255 0 0 255 0
0 0 255 0 0 255 0 0 255 0 0 255

```

dst [bgr]

```

0 0 255 0 0 255 0 0 255 0 0 255
0 255 0 0 255 0 0 255 0 0 255 0
255 0 0 255 0 0 255 0 0 255 0 0

```

## AddRandUniform\_Direct

*Generates random samples with uniform distribution and adds them to an image data.*

### Syntax

```
IppStatus ippiAddRandUniform_Direct_<mod>(Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize, Ipp<datatype> low, Ipp<datatype> high, unsigned int* pSeed);
```

Supported values for mod:

8u_C1IR	16u_C1IR	16s_C1IR	32f_C1IR
8u_C3IR	16u_C3IR	16s_C3IR	32f_C3IR
8u_C4IR	16u_C4IR	16s_C4IR	32f_C4IR
8u_AC4IR	16u_AC4IR	16s_AC4IR	32f_AC4IR

### Parameters

<i>pSrcDst</i>	Pointer to the source and destination image ROI.
<i>srcDstStep</i>	Distance in bytes between starts of consecutive lines in the source and destination image.
<i>roiSize</i>	Size of the image ROI in pixels.
<i>low</i>	The lower bound for the range of uniformly distributed values.
<i>high</i>	The upper bound for the range of uniformly distributed values.
<i>pSeed</i>	The initial seed value for the pseudo-random number generator.

### Description

The function `ippiAddRandUniform_Direct` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

The function generates samples with uniform distribution over the range  $[low, high]$  and adds them to a source image pointed to by *pSrcDst*.

The resulting pixel values that exceed the image data range are saturated to the respective data-range limits. To obtain an image that contains pure noise with uniform distribution, call `ippiAddRandUniform_Direct` using a source image with zero data as input.

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error when any of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcDstStep</i> has a zero or negative value.

## Example

The code example below shows data conversion without scaling.

```

IppStatus randUniform( void ) {
    unsigned int seed = 123456;
    Ipp8u img[2048], mn, mx;
    IppiSize roi={2048,1};
    Ipp64f mean;
    IppStatus st;
    ippiSet_8u_C1R( 0, img, 2048, roi );
    st = ippiAddRandUniform_Direct_8u_C1R(img, 2048, roi, 0, 255, &seed);
    ippiMean_8u_C1R( img, 2048, roi, &mean );
    ippiMinMax_8u_C1R( img, 2048, roi, &mn, &mx );
    printf( "[%d..%d], mean=%.3f\n", mn, mx, mean );
    return st;
}

```

## AddRandGauss\_Direct

*Generates random samples with Gaussian distribution and adds them to an image data.*

---

### Syntax

```

IppStatus ippiAddRandGauss_Direct_<mod>(Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize
roiSize, Ipp<datatype> mean, Ipp<datatype> stDev, unsigned int* pSeed);

```

Supported values for mod:

8u_C1IR	16u_C1IR	16s_C1IR	32f_C1IR
8u_C3IR	16u_C3IR	16s_C3IR	32f_C3IR
8u_C4IR	16u_C4IR	16s_C4IR	32f_C4IR
8u_AC4IR	16u_AC4IR	16s_AC4IR	32f_AC4IR

### Parameters

<i>pSrcDst</i>	Pointer to the source and destination image ROI.
<i>srcDstStep</i>	Distance in bytes between starts of consecutive lines in the source and destination image.
<i>roiSize</i>	Size of the image ROI in pixels.
<i>mean</i>	The mean of the Gaussian distribution.
<i>stDev</i>	The standard deviation of the Gaussian distribution.
<i>pSeed</i>	The initial seed value for the pseudo-random number generator.

### Description

The function `ippiAddRandGauss_Direct` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

The function generates samples with Gaussian distribution that have the mean value *mean* and standard deviation *stdev* and adds them to a source image ROI pointed to by *pSrcDst*.



The resulting pixel values that exceed the image data range are saturated to the respective data-range limits. To obtain an image which contains pure noise with Gaussian distribution, call `ippiAddRandGauss_Direct` using a source image with zero data as input.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <code>pSrcDst</code> or <code>pSeed</code> pointer is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <code>roiSize</code> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <code>srcDstStep</code> has a zero or negative value.

## ImageJaehne

*Creates Jaehne test image.*

## Syntax

```
IppStatus ippiImageJaehne_<mod>(Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);
```

Supported values for `mod`:

<code>8u_C1R</code>	<code>8s_C1R</code>	<code>16u_C1R</code>	<code>16s_C1R</code>	<code>32s_C1R</code>	<code>32f_C1R</code>
<code>8u_C3R</code>	<code>8s_C3R</code>	<code>16u_C3R</code>	<code>16s_C3R</code>	<code>32s_C3R</code>	<code>32f_C3R</code>
<code>8u_C4R</code>	<code>8s_C4R</code>	<code>16u_C4R</code>	<code>16s_C4R</code>	<code>32s_C4R</code>	<code>32f_C4R</code>
<code>8u_AC4R</code>	<code>8s_AC4R</code>	<code>16u_AC4R</code>	<code>16s_AC4R</code>	<code>32s_AC4R</code>	<code>32f_AC4R</code>

## Parameters

<code>pDst</code>	Pointer to the destination image ROI.
<code>dstStep</code>	Distance in bytes between starts of consecutive lines in the destination image.
<code>roiSize</code>	Size of the destination image ROI in pixels.

## Description

The function `ippiImageJaehne` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function creates a specific one- or three-channel test image that has been first introduced to digital image processing by B.Jaehne (see [Jae95]).

The destination image pixel values are computed according to the following formula:

$$\text{Dst}(x,y) = A \cdot \sin(0.5 \cdot \text{IPP\_PI} \cdot (x_2^2 + y_2^2) / \text{roiSize.height}),$$

where  $x, y$  are pixel coordinates varying in the range

$$0 \leq x \leq \text{roiSize.width}-1, 0 \leq y \leq \text{roiSize.height}-1;$$

`IPP_PI` is the library constant that stands for  $\pi$  value.

$$x_2 = (x - \text{roiSize.width} + 1) / 2.0,$$

```
y2 = (y-roiSize.height+1)/2.0,
```

*A* is the constant value that depends upon the image type being created.

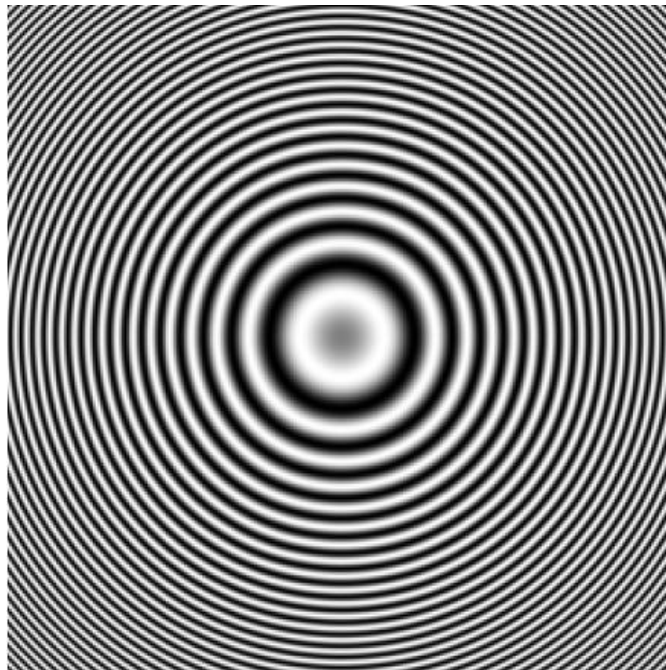
For the `32f` floating point data, the pixel values in the created image can vary in the range between 0 (inclusive) and 1 (exclusive).

Figure Example of a Generated Jaehne's Test Image illustrates an example of a test image generated by the `ippiImageJaehne` function.

---

#### Example of a Generated Jaehne's Test Image

---




---

These test images can be effectively used when you need to visualize and interpret the results of applying filtering functions, similarly to what is proposed in [Jae95].

#### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if the <i>pDst</i> pointer is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with a zero or negative value, or if <i>dstStep</i> is less than or equal to zero.

## ImageRamp

*Creates a test image that has an intensity ramp.*

### Syntax

```
IppStatus ippiImageRamp_<mod>(Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, float offset, float slope, IppiAxis axis);
```

Supported values for mod:

8u_C1R	8s_C1R	16u_C1R	16s_C1R	32s_C1R	32f_C1R
8u_C3R	8s_C3R	16u_C3R	16s_C3R	32s_C3R	32f_C3R
8u_C4R	8s_C4R	16u_C4R	16s_C4R	32s_C4R	32f_C4R
8u_AC4R	8s_AC4R	16u_AC4R	16s_AC4R	32s_AC4R	32f_AC4R

### Parameters

<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>roiSize</i>	Size of the destination image ROI in pixels.
<i>offset</i>	Offset value.
<i>slope</i>	Slope coefficient.
<i>axis</i>	Specifies the direction of the image intensity ramp; can be one of the following:
<code>ippAxsHorizontal</code>	for the ramp in <i>x</i> -direction,
<code>ippAxsVertical</code>	for the ramp in <i>y</i> -direction,
<code>ippAxsBoth</code>	for the ramp in both <i>x</i> and <i>y</i> -directions.

### Description

The function `ippiImageRamp` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

The function creates a one- or three-channel image that can be used as a test image to examine the effect of applying different image processing functions.

The destination image pixel values are computed according to one of the following formulas:

$\text{dst}(x,y) = \text{offset} + \text{slope} * x$ , if  $\text{axis} = \text{ippAxsHorizontal}$ ,

$\text{dst}(x,y) = \text{offset} + \text{slope} * y$ , if  $\text{axis} = \text{ippAxsVertical}$ ,

$\text{dst}(x,y) = \text{offset} + \text{slope} * x * y$ , if  $\text{axis} = \text{ippAxsBoth}$ ,

where  $x, y$  are pixel coordinates varying in the range

$0 \leq x \leq \text{roiSize.width}-1, 0 \leq y \leq \text{roiSize.height}-1$ ;

Note that linear transform coefficients *offset* and *slope* have floating-point values for all function flavors. The computed pixel values that exceed the image data range are saturated to the respective data-range limits.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if the <i>pDst</i> pointer is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with a zero or negative value, or if <i>dstStep</i> is less than or equal to zero.

## Example

The code example below illustrates how to use the `ippiImageRamp` function.

```
IppStatus ramp( void ){
    Ipp8u dst[8*4];
    IppiSize roiSize = { 8, 4 };
    return ippiImageRamp_8u_C1R( dst, 8, roiSize, 0.0f, 256.0f/7, ippAxsHorizontal);
}
```

The destination image contains the following data:

```
00 25 49 6E 92 B7 DB FF
00 25 49 6E 92 B7 DB FF
00 25 49 6E 92 B7 DB FF
00 25 49 6E 92 B7 DB FF
```

## SampleLine

*Stores a raster line into buffer.*

---

### Syntax

```
IppStatus ippiSampleLine_<mod>(const Ipp<datatype>* pSrc, int srcStep, IppiSize roiSize,
Ipp<datatype>* pDst, IppiPoint pt1, IppiPoint pt2);
```

Supported values for *mod*:

<code>8u_C1R</code>	<code>16u_C1R</code>	<code>32f_C1R</code>
<code>8u_C3R</code>	<code>16u_C3R</code>	<code>32f_C3R</code>

### Parameters

<i>pSrc</i>	Pointer to the ROI in the source raster image.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the raster image.
<i>roiSize</i>	Size of the image ROI in pixels.
<i>pDst</i>	Pointer to the destination buffer. The buffer is to store at least $\max( pt2.x - pt1.x  + 1,  pt2.y - pt1.y  + 1)$ points.
<i>pt1</i>	Starting point of the line.
<i>pt2</i>	Ending point of the line.

## Description

The function `ippiSampleLine` is declared in the `ippcv.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function iterates through the points that belong to the raster line using the 8-point connected Bresenham algorithm, and stores the resulting pixels in the destination buffer.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if any of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <code>roiSize.width</code> or <code>roiSize.height</code> is less than or equal to zero.
<code>ippStsStepErr</code>	Indicates an error condition if <code>srcStep</code> is less than <code>roiSize.width * &lt;pixelSize&gt;</code> .
<code>ippStsNotEvenStepErr</code>	Indicates an error when the step for the floating-point image cannot be divided by 4.
<code>ippStsOutOfRangeErr</code>	Indicates an error when any of the line ending points is outside the image.

## Example

The code example below shows how to use the function `ippiSampleLine_8u_C1R`.

```
void func_sampleline()
{
    Ipp8u pSrc[5*4] = { 0, 1, 2, 3, 4,
                       5, 6, 7, 8, 9,
                       0, 9, 8, 7, 6,
                       5, 4, 3, 2, 1 };
    IppiSize roiSize = {5, 4};
    IppiPoint pt1 = {1, 1};
    IppiPoint pt2 = {2, 3};
    Ipp8u pDst[3];
    int srcStep = 8*sizeof(Ipp8u);

    ippiSampleLine_8u_C1R( pSrc, srcStep, roiSize, pDst, pt1, pt2 );
}
```

Result: 9 3 18

## ZigzagFwd8x8

Converts a conventional order to the zigzag order.

## Syntax

```
IppStatus ippiZigzagFwd8x8_16s_C1(const Ipp16s* pSrc, Ipp16s* pDst);
```

## Parameters

<code>pSrc</code>	Pointer to the source data.
<code>pDst</code>	Pointer to the destination data.

## Description

The function `ippiZigzagFwd8x8` is declared in the `ippi.h` file. This function rearranges data in an 8x8 block from a conventional order (left-to-right, top-to-bottom) to the zigzag sequence.

Figure Zigzag Sequence specifies the resulting zigzag sequence.

## Zigzag Sequence

---

0	1	5	6	14	15	27	28
2	4	7	13	16	26	29	42
3	8	12	17	25	30	41	43
9	11	18	24	31	40	44	53
10	19	23	32	39	45	52	54
20	22	33	38	46	51	55	60
21	34	37	47	50	56	59	61
35	36	48	49	57	58	62	63

---

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when any of the specified pointers is <code>NULL</code> .

## Example

The code example below shows how to use the `ippiZigzagFwd8x8_16s_C1` function.

```

Ipp16s src[8*8] = {
    0, 1, 5, 7, 9, 2, 4, 1,
    5, 4, 8, 6, 3, 8, 0, 3,
    6, 2, 6, 8, 1, 4, 2, 8,
    4, 3, 2, 9, 3, 0, 6, 6,
    7, 7, 3, 0, 4, 1, 0, 9,
    5, 1, 9, 2, 5, 7, 1, 7,
    0, 3, 5, 0, 7, 5, 9, 8,
    2, 9, 1, 4, 6, 8, 2, 3
};

Ipp16s dst[8*8];

ippiZigzagFwd8x8_16s_C1 ( src, dst );

Result:
0 1 5 7 9 2 4 1
5 4 8 6 3 8 0 3
6 2 6 8 1 4 2 8
4 3 2 9 3 0 6 6      src //conventional order
7 7 3 0 4 1 0 9
5 1 9 2 5 7 1 7
0 3 5 0 7 5 9 8
2 9 1 4 6 8 2 3

```

```

0 1 5 6 4 5 7 8
2 4 7 3 6 6 9 2
3 8 2 7 5 0 1 3
9 1 8 4 1 0 4 3    dst  //zigzag order
0 9 3 2 9 5 2 4
0 2 3 8 6 1 5 0
1 4 7 7 0 6 9 1
5 6 8 9 7 8 2 3

```

## ZigzagInv8x8

*Converts a zigzag order to the conventional order.*

### Syntax

```
IppStatus ippiZigzagInv8x8_16s_C1(const Ipp16s* pSrc, Ipp16s* pDst);
```

### Parameters

*pSrc*                                      Pointer to the source data.  
*pDst*                                      Pointer to the destination data.

### Description

The function `ippiZigzagInv8x8` is declared in the `ippi.h` file. This function rearranges data in an 8x8 block from a zigzag sequence to the conventional order (left-to-right, top-to-bottom).

Figure "Zigzag Sequence" specifies the resulting zigzag sequence.

### Return Values

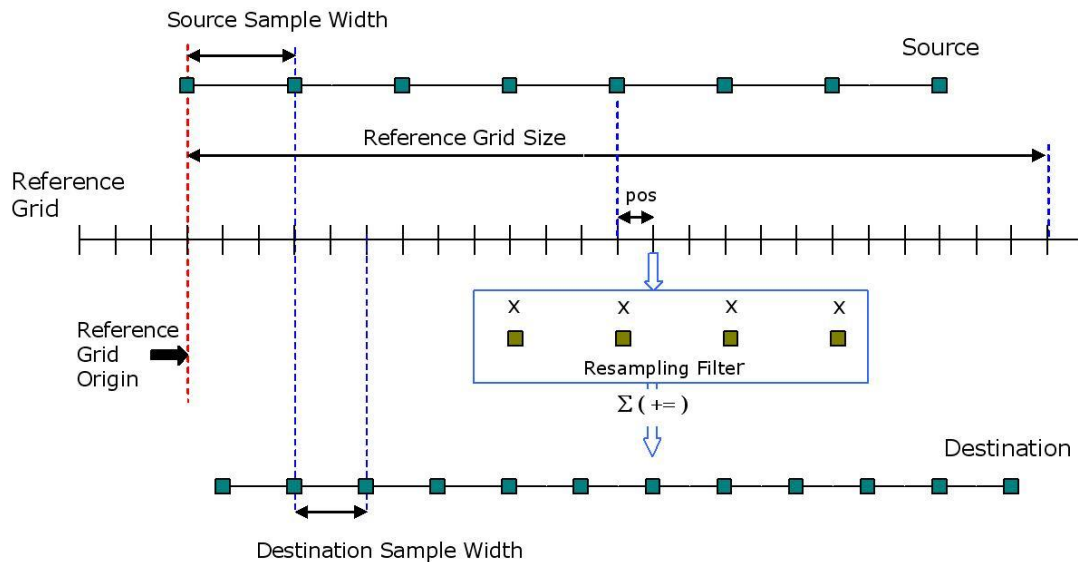
`ippStsNoErr`                                      Indicates no error.  
`ippStsNullPtrErr`                                      Indicates an error when any of the specified pointers is `NULL`.

## Row Oriented Resampling

This section describes the low-level high-optimized functions for image resampling. They can be used for resizing of images with arbitrary rational resizing factor. The functions use anti-aliasing filters in effective polyphase implementation. (The similar implementation is used in signal-processing multi-rate filters.)

For the perfect unshifted block-by-block or parallelized image data processing, the source and destination images are defined with respect to the high-resolution reference grid (see Figure Row-Oriented Resampling of an Image). Required borders around the source image may either be created manually by the user or automatically by the special flavor of the resampling function.

### Row-Oriented Resampling of an Image



### ResampleRowGetSize

Computes the sizes of the resample structure and the work buffer.

#### Syntax

```
IppStatus ippiResampleRowGetSize_32f(int srcSampleWidth, int dstSampleWidth, int* pSpecSize,
int* pBufSize);
```

#### Parameters

<i>srcSampleWidth</i>	Distance between neighbor pixels of the source row on the reference grid.
<i>dstSampleWidth</i>	Distance between neighbor pixels of the destination row on the reference grid.
<i>pSpecSize</i>	Pointer to the computed size of the structure.
<i>pBufSize</i>	Pointer to the computed size of the buffer.

#### Description

The function `ippiResampleRowGetSize` is declared in the `ippi.h` file.



This function calculates sizes of the resampling parameters structure `IppiResampleRowSpec` structure and the work buffer utilized by the functions `ippiResampleRow` and `ippiResampleRowReplicateBorder`. These values are stored in the `pSpecSize` and `pBufSize` respectively. Calculations are performed in accordance with the parameters of resampling: widths of source and destination samples. The structure and the buffer must be created by the user. Use the function `ippiResampleRowInit` to initialize the `IppiResampleRowSpec` structure.

This function computes the sizes of the row resampling structure `IppiResampleRowSpec` and of the work buffer.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointers is <code>NULL</code> .
<code>ippStsOutOfRangeErr</code>	Indicates an error condition if any of the <code>srcSampleWidth</code> or <code>dstSampleWidth</code> parameters is not valid.

## ResampleRowInit

*Initializes the row resampling structure.*

## Syntax

```
IppStatus ippiResampleRowInit_32f(IppiResampleRowSpec_32f* pRowSpec, IppiResampleType filterType, int srcSampleWidth, int dstSampleWidth);
```

## Parameters

<code>srcSampleWidth</code>	Distance between neighbor pixels of the source row on the reference grid.
<code>dstSampleWidth</code>	Distance between neighbor pixels of the destination row on the reference grid.
<code>pRowSpec</code>	Pointer to the initialized structure.
<code>filterType</code>	Specifies the type of the interpolation filter that is used for resampling. Possible value is <code>ippResampleCatmullRom</code> for the Catmull-Rom interpolation.

## Description

The function `ippiResampleRowInit` is declared in the `ippi.h` file.

This function initializes in the external buffer the resampling specification structure `IppiResampleRowSpec` created by the user. The buffer size should be computed previously by the function `ippiResampleRowGetSize`. This function is necessary for the functions `ippiResampleRow` and `ippiResampleRowReplicateBorder`.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointers is <code>NULL</code> .
<code>ippStsBadArgErr</code>	Indicates an error condition if the specified filter type is incorrect or unsupported.

`ippStsOutOfRangeErr` Indicates an error condition if any of the `srcSampleWidth` or `dstSampleWidth` parameters is not valid.

## ResampleRowGetBorderWidth

*Computes the width of the left and right borders.*

---

### Syntax

```
IppStatus ippResampleRowGetBorderWidth_32f(const IppiResampleRowSpec_32f* pRowSpec, int* pLeftWidth, int* pRightWidth);
```

### Parameters

<code>pRowSpec</code>	Pointer to the initialized row resampling structure.
<code>pLeftWidth</code>	Pointer to the size (in pixels) of the left border width.
<code>pRightWidth</code>	Pointer to the size (in pixels) of the right border width.

### Description

The function `ippResampleRowGetBorderWidth` is declared in the `ippi.h` file.

This function computes the width of the right and left borders (in pixels) that must be added to the source buffer before resampling with the function [ippiResampleRowReplicateBorder](#). The results are stored in the `pRightWidth` and `pLeftWidth` respectively. The width of each border depends on parameters of resampling determined in the `IppiResampleRowSpec` structure.

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if the <code>pRowSpec</code> pointer is NULL.
<code>ippStsBadArgErr</code>	Indicates an error condition if the specified filter type is incorrect or unsupported.

## ResampleRow, ResampleRowReplicateBorder

*Performs row-oriented resampling of the image.*

---

### Syntax

```
IppStatus ippResampleRow_32f_C1(const Ipp32f* pSrc, int srcLineStep, Ipp32f* pDst, int dstLineStep, int xOriginRefGrid, IppiSize sizeRefGrid, const IppiResampleRowSpec_32f* pRowSpec, Ipp8u* pBuffer);
```

```
IppStatus ippResampleRowReplicateBorder_32f_C1R(const Ipp32f* pSrc, int srcLineStep, Ipp32f* pDst, int dstLineStep, int xOriginRefGrid, IppiSize sizeRefGrid, const IppiResampleRowSpec_32f* pRowSpec, Ipp8u* pBuffer);
```

### Parameters

<code>pSrc</code>	Pointer to the source row origin.
-------------------	-----------------------------------

<i>srcLineStep</i>	Distance in bytes between starts of consecutive lines in the source image buffer.
<i>pDst</i>	Pointer to the destination row origin.
<i>dstLineStep</i>	Distance in bytes between starts of consecutive lines in the destination image buffer.
<i>xOriginRefGrid</i>	<i>x</i> -coordinate of the image origin in the reference grid.
<i>sizeRefGrid</i>	Size of the image in terms of the reference grid.
<i>pRowSpec</i>	Pointer to the initialized row resampling structure.
<i>pBuffer</i>	Pointer to the work buffer.

## Description

The functions `ippiResampleRow` and `ippiResampleRowReplicateBorder` are declared in the `ippi.h` file.

These functions resample one or several rows of the source image *pSrc*. Parameters of resampling (including filter taps) are defined by the `IppiResampleRowSpec` structure. Reference grid is determined by its origin (the *xOriginRefGrid* parameter) and size (the *sizeRefGrid* parameter). Height of the *sizeRefGrid* structure defines number of rows to be resampled. The result is written to the destination image *pDst*.

Function `ippiResampleRow` automatically creates replicated borders around the source image.

Function `ippiResampleRowReplicateBorder` requires left and right replicate borders created by the user around the source image for each row. Use the `ippiResampleRowGetBorderWidth` function to compute width of the borders.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointer is <code>NULL</code> .
<code>ippStsBadArgErr</code>	Indicates an error condition if the structure is corrupted.
<code>ippStsOutOfRangeErr</code>	Indicates an error condition if the <i>xOriginRefGrid</i> is out of the range.
<code>ippStsSizeErr</code>	Indicates an error condition if the <i>sizeRefGrid</i> is not valid.



# Image Arithmetic and Logical Operations

## 5

This chapter describes Intel IPP functions that modify pixel values of an image buffer using arithmetic or logical operations. It also includes functions that perform image compositing based on opacity (alpha-blending).

The table below provides a list of arithmetic and logical functions:

### Arithmetic and Logical Functions

Function Base Name	Operation
<b>Arithmetic Functions</b>	
Add	Adds pixel values of two image buffers
AddC	Adds a constant to pixel values of an image buffer
AddSquare	Adds squared pixel values of a source image to floating-point pixel values of an accumulator image
AddProduct	Adds product of pixel values of two source images to floating-point pixel values of an accumulator image
AddWeighted	Adds pixel values of a source image multiplied by factor $\alpha$ to floating-point pixel values of an accumulator image multiplied by factor $(1-\alpha)$
Mul	Multiplies pixel values of two image buffers
MulC	Multiplies pixel values of an image buffer by a constant
MulScale	Multiplies pixel values of two image buffers and scales the products
MulCScale	Multiplies pixel values of an image buffer by a constant and scales the products
Sub	Subtracts pixel values of two image buffers
SubC	Subtracts a constant from pixel values of an image buffer
Div	Divides pixel values of two image buffers
Div_Round	Divides pixel values of an image by pixel values of another image with different rounding modes

Function Base Name	Operation
<a href="#">DivC</a>	Divides pixel values of an image buffer by a constant
<a href="#">Abs</a>	Computes absolute pixel values of a source image and places them into the destination image
<a href="#">AbsDiff</a>	Finds the absolute difference between two images.
<a href="#">AbsDiffC</a>	Finds the absolute difference between an image and a scalar value.
<a href="#">Sqr</a>	Squares pixel values of an image and writes them into the destination image
<a href="#">Sqrt</a>	Computes square roots of pixel values of a source image and writes them into the destination image
<a href="#">Ln</a>	Computes the natural logarithm of pixel values in a source image and writes the results into the destination image
<a href="#">Exp</a>	Computes the exponential of pixel values in a source image and writes the results into the destination image
<a href="#">Complement</a>	Converts negative number from the complement to direct code.
<a href="#">DotProd</a>	Computes the dot product of pixel values of two source images.
<a href="#">DotProdCol</a>	Calculates the dot product of taps vector and columns of the specified set of rows.
<b>Logical Functions</b>	
<a href="#">And</a>	Combines corresponding pixels of two image buffers by a bitwise AND operation.
<a href="#">AndC</a>	Performs a bitwise AND operation on each pixel with a constant.
<a href="#">Or</a>	Combines corresponding pixels of two image buffers by a bitwise OR operation.
<a href="#">OrC</a>	Performs a bitwise OR operation on each pixel with a constant.
<a href="#">Xor</a>	Combines corresponding pixels of two image buffers by a bitwise XOR operation.
<a href="#">XorC</a>	Performs a bitwise OR operation on each pixel with a constant.
<a href="#">Not</a>	Performs a bitwise NOT operation on each pixel
<a href="#">RShiftC</a>	Divides pixel values by a constant power of 2 by shifting bits logically or arithmetically to the right.
<a href="#">LShiftC</a>	Shifts bits in pixel values to the left.

Function Base Name	Operation
<b>Alpha-Blending Functions</b>	
<a href="#">AlphaComp</a>	Combines two image buffers using alpha (opacity) values.
<a href="#">AlphaCompC</a>	Combines two image buffers using constant alpha values
<a href="#">AlphaPremul</a>	Pre-multiplies pixel values of an image buffer by its alpha values.
<a href="#">AlphaPremulC</a>	Pre-multiplies pixel values of an image buffer using constant alpha values.

An additional suffix *C*, if present in the function name, indicates operation with a constant. Arithmetic functions that operate on integer data perform fixed scaling of the internally computed results. In case of overflow the result value is saturated to the destination data type range.



**NOTE.** Most arithmetic and logical functions support data with 1-, 3-, or 4-channel pixel values. In the alpha channel case (AC4), the alpha channels are not processed.

Many solutions and hints for use of these functions can be found in Intel IPP Samples. See *Intel® IPP Image Processing and Media Processing Samples* downloadable from <http://www.intel.com/cd/software/products/asmo-na/eng/220046.htm>.

## Arithmetic Operations

Functions described in this section perform arithmetic operations on pixel values. Arithmetic operations include addition, multiplication, subtraction, and division of pixel values of two images as well as similar operations on a single image and a constant. Computation of an absolute value, square, square root, exponential, and natural logarithm of pixels in an image buffer is also supported. Functions of this group perform operations on each pixel in the source buffer(s), and write the results into the destination buffer. Some functions also support processing of images with complex data.

### Add

*Adds pixel values of two images.*

#### Syntax

##### Case 1: Not-in-place operation on integer or complex data

```
IppStatus ippiAdd_<mod>(const Ipp<datatype>* pSrc1, int src1Step, const Ipp<datatype>* pSrc2, int src2Step, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, int scaleFactor);
```

Supported values for *mod*:

8u_C1RSfs	16u_C1RSfs	16s_C1RSfs	16sc_C1RSfs	32sc_C1RSfs
8u_C3RSfs	16u_C3RSfs	16s_C3RSfs	16sc_C3RSfs	32sc_C3RSfs

8u_AC4RSfs	16u_AC4RSfs	16s_AC4RSfs	16sc_AC4RSfs	32sc_AC4RSfs
8u_C4RSfs	16u_C4RSfs	16s_C4RSfs		

**Case 2: Not-in-place operation on floating-point or complex data**

```
IppStatus ippiAdd_<mod>(const Ipp<datatype>* pSrc1, int src1Step, const Ipp<datatype>*
pSrc2, int src2Step, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);
```

Supported values for mod:

32f_C1R	32fc_C1R
32f_C3R	32fc_C3R
32f_AC4R	32fc_AC4R
32f_C4R	

**Case 3: In-place operation on integer or complex data**

```
IppStatus ippiAdd_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pSrcDst,
int srcDstStep, IppiSize roiSize, int scaleFactor);
```

Supported values for mod:

8u_C1IRSfs	16u_C1IRSfs	16s_C1IRSfs	16sc_C1IRSfs	32sc_C1IRSfs
8u_C3IRSfs	16u_C3IRSfs	16s_C3IRSfs	16sc_C3IRSfs	32sc_C3IRSfs
8u_AC4IRSfs	16u_AC4IRSfs	16s_AC4IRSfs	16sc_AC4IRSfs	32sc_AC4IRSfs
8u_C4IRSfs	16u_C4IRSfs	16s_C4IRSfs		

**Case 4: In-place operation on floating-point or complex data**

```
IppStatus ippiAdd_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pSrcDst,
int srcDstStep, IppiSize roiSize);
```

Supported values for mod:

32f_C1IR	32fc_C1IR
32f_C3IR	32fc_C3IR
32f_AC4IR	32fc_AC4IR
32f_C4IR	



**Case 5: In-place operation using a floating-point accumulator image**

```
IppStatus ippiAdd_<mod>(const Ipp<srcDatatype>* pSrc, int srcStep, Ipp32f* pSrcDst, int srcDstStep, IppiSize roiSize);
```

Supported values for `mod`:

8u32f\_C1IR 8s32f\_C1IR 16u32f\_C1IR

**Case 6: Masked in-place operation using a floating-point accumulator image**

```
IppStatus ippiAdd_<mod>(const Ipp<srcDatatype>* pSrc, int srcStep, const Ipp8u* pMask, int maskStep, Ipp32f* pSrcDst, int srcDstStep, IppiSize roiSize);
```

Supported values for `mod`:

8u32f\_C1IMR 8s32f\_C1IMR 16u32f\_C1IMR 32f\_C1IMR

**Parameters**

<i>pSrc1, pSrc2</i>	Pointer to the ROI in the source images.
<i>src1Step, src2Step</i>	Distance in bytes between starts of consecutive lines in the source images.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>pSrc</i>	Pointer to the first source image ROI for the in-place operation.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the first source image for the in-place operation.
<i>pSrcDst</i>	Pointer to the second source and destination image ROI for the in-place operation.
<i>srcDstStep</i>	Distance in bytes between starts of consecutive lines in the source and destination image for the in-place operation.
<i>pMask</i>	Pointer to the mask image ROI for the masked operation.
<i>maskStep</i>	Distance in bytes between starts of consecutive lines in the mask image for the masked operation.
<i>roiSize</i>	Size of the source and destination ROI in pixels.
<i>scaleFactor</i>	Scale factor (see <a href="#">Integer Result Scaling</a> ).

**Description**

The function `ippiAdd` is declared in the `ippi.h` file with the exception of the function flavors in Case 5 and Case 6, which are declared in the `ippcv.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function adds corresponding pixel values of two source image buffers and places the results in a destination buffer. In case of operations on integer data, the resulting values are scaled by *scaleFactor*. For complex data, the function processes both real and imaginary parts of pixel values. Some function flavors add 8u, 8s, 16u, or

32f source image pixel values to a floating point accumulator image in-place. Addition of pixel values in case of a masked operation is performed only if the respective mask value is nonzero; otherwise, the accumulator pixel value remains unchanged.



**NOTE.** Step values must be positive for functions that operate on complex data, and no less than `roiSize.width*<pixelSize>` for functions using an accumulator image.

Functions with AC4 descriptor do not process alpha channels.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if any of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <code>roiSize</code> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition in the following cases: for functions that operate on complex data, if any of the specified step values is zero or negative; for functions using an accumulator image, if any of the specified step values is less than <code>roiSize.width * &lt;pixelSize&gt;</code> .
<code>ippStsNotEvenStepErr</code>	Indicates an error condition if one of step values for floating-point images cannot be divided by 4.

## Example

The code example below shows how to use the function `ippiAdd_8u_C1RSfs`.

```

Ipp8u src1[8*4] = {8, 4, 2, 1, 0, 0, 0, 0,
                   8, 4, 2, 1, 0, 0, 0, 0,
                   8, 4, 2, 1, 0, 0, 0, 0,
                   8, 4, 2, 1, 0, 0, 0, 0};
Ipp8u src2[8*4] = {4, 3, 2, 1, 0, 0, 0, 0,
                   4, 3, 2, 1, 0, 0, 0, 0,
                   4, 3, 2, 1, 0, 0, 0, 0,
                   4, 3, 2, 1, 0, 0, 0, 0};
Ipp8u dst[8*4];
IppiSize srcRoi = { 4, 4 };
Int scaleFactor = 1;    // later examples for 2 and -2 values

ippiAdd_8u_C1RSfs (src1, 8, src2, 8, dst, 4, srcRoi, scaleFactor );

```

Result:

src1	src2
8 4 2 1 0 0 0 0	4 3 2 1 0 0 0 0
8 4 2 1 0 0 0 0	4 3 2 1 0 0 0 0
8 4 2 1 0 0 0 0	4 3 2 1 0 0 0 0
8 4 2 1 0 0 0 0	4 3 2 1 0 0 0 0

dst >>	scaleFactor = 1	scaleFactor = 2	ScaleFactor = -2
	6 4 2 1	3 2 1 0	48 28 16 8
	6 4 2 1	3 2 1 0	48 28 16 8
	6 4 2 1	3 2 1 0	48 28 16 8
	6 4 2 1	3 2 1 0	48 28 16 8

## AddC

*Adds a constant to pixel values of an image.*

---

### Syntax

#### Case 1: Not-in-place operation on one-channel integer or complex data

```
IppStatus ippiAddC_<mod>(const Ipp<datatype>* pSrc, int srcStep, const Ipp<datatype>* value, const Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, int scaleFactor);
```

Supported values for mod:

8u\_C1RSfs      16u\_C1RSfs      16s\_C1RSfs      16sc\_C1RSfs      32sc\_C1RSfs

#### Case 2: Not-in-place operation on multi-channel integer or complex data

```
IppStatus ippiAddC_<mod>(const Ipp<datatype>* pSrc, int srcStep, const Ipp<datatype>* value[3], Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, int scaleFactor);
```

Supported values for mod:

8u\_C3RSfs      16u\_C3RSfs      16s\_C3RSfs      16sc\_C3RSfs      32sc\_C3RSfs  
8u\_AC4RSfs      16u\_AC4RSfs      16s\_AC4RSfs      16sc\_AC4RSfs      32sc\_AC4RSfs

```
IppStatus ippiAddC_<mod>(const Ipp<datatype>* pSrc, int srcStep, const Ipp<datatype>* value[4], Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, int scaleFactor);
```

Supported values for mod:

8u\_C4RSfs      16u\_C4RSfs      16s\_C4RSfs

#### Case 3: Not-in-place operation on one-channel floating-point or complex data

```
IppStatus ippiAddC_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* value, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);
```

Supported values for mod:

32f\_C1R      32fc\_C1R

#### Case 4: Not-in-place operation on multi-channel floating-point or complex data

```
IppStatus ippiAddC_<mod>(const Ipp<datatype>* pSrc, int srcStep, const Ipp<datatype>* value[3], Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);
```

Supported values for mod:

32f\_C3R      32fc\_C3R  
32f\_AC4R      32fc\_AC4R

```
IppStatus ippiAddC_32f_C4R(const Ipp32f* pSrc, int srcStep, const Ipp32f* value[4], Ipp32f* pDst, int dstStep, IppiSize roiSize);
```

#### Case 5: In-place operation on one-channel integer or complex data

```
IppStatus ippiAddC_<mod>(Ipp<datatype>* value, Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize, int scaleFactor);
```

Supported values for mod:

8u_C1IRSfs	16u_C1IRSfs	16s_C1IRSfs	16sc_C1IRSfs	32sc_C1IRSfs
------------	-------------	-------------	--------------	--------------

#### Case 6: In-place operation on multi-channel integer or complex data

```
IppStatus ippiAddC_<mod>(const Ipp<datatype>* value[3], Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize, int scaleFactor);
```

Supported values for mod:

8u_C3IRSfs	16u_C3IRSfs	16s_C3IRSfs	16sc_C3IRSfs	32sc_C3IRSfs
8u_AC4IRSfs	16u_AC4IRSfs	16s_AC4IRSfs	16sc_AC4IRSfs	32sc_AC4IRSfs

```
IppStatus ippiAddC_<mod>(const Ipp<datatype>* value[4], Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize, int scaleFactor);
```

Supported values for mod:

8u_C4IRSfs	16u_C4IRSfs	16s_C4IRSfs
------------	-------------	-------------

#### Case 7: In-place operation on multi-channel floating-point or complex data

```
IppStatus ippiAddC_<mod>(const Ipp<datatype>* value, Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize);
```

Supported values for mod:

32f_C1IR	32fc_C1IR
----------	-----------

#### Case 8: In-place operation on multi-channel floating-point or complex data

```
IppStatus ippiAddC_<mod>(const Ipp<datatype>* value[3], Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize);
```

Supported values for mod:

32f_C3IR	32fc_C3IR
32f_AC4IR	32fc_AC4IR

```
IppStatus ippiAddC_32f_C4IR(const Ipp32f value[4], Ipp32f* pSrcDst, int srcDstStep, IppiSize roiSize);
```

## Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>value</i>	The constant value to add to image pixel values (constant vector in case of multi-channel images).
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>pSrcDst</i>	Pointer to the source and destination image ROI for the in-place operation.
<i>srcDstStep</i>	Distance in bytes between starts of consecutive lines in the source and destination image buffer for the in-place operation.
<i>roiSize</i>	Size of the source and destination ROI in pixels.
<i>scaleFactor</i>	Scale factor (see <a href="#">Integer Result Scaling</a> ).

## Description

The function `ippiAddC` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function changes the image intensity by adding *value* to image pixel values. For one-channel images, a positive *value* brightens the image (increases the intensity); a negative value darkens the image (decreases the intensity). For multi-channel images, the components of a constant vector *value* are added to pixel channel values. For complex data, the function processes both real and imaginary parts of pixel values.



**NOTE.** Step values must be positive for functions that operate on complex data.

In case of operations on integer data, the resulting values are scaled by *scaleFactor*.

Note that the functions with `AC4` descriptor do not process alpha channels.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if any of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if any of the specified buffer step values is zero or negative for functions that operate on complex data.

## AddSquare

*Adds squared pixel values of a source image to floating-point pixel values of an accumulator image.*

---

### Syntax

#### Case 1: In-place operation

```
IppStatus ippiAddSquare_<mod>(const Ipp<srcDatatype>* pSrc, int srcStep, Ipp32f* pSrcDst,
int srcDstStep, IppiSize roiSize);
```

Supported values for mod:

8u32f\_C1IR      8s32f\_C1IR      16u32f\_C1IR      32f\_C1IR

#### Case 2: Masked in-place operation

```
IppStatus ippiAddSquare_<mod>(const Ipp<srcDatatype>* pSrc, int srcStep, const Ipp8u*
pMask, int maskStep, Ipp32f* pSrcDst, int srcDstStep, IppiSize roiSize);
```

Supported values for mod:

8u32f\_C1IMR      8s32f\_C1IMR      16u32f\_C1IMR      32f\_C1IMR

### Parameters

<i>pSrc1</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pMask</i>	Pointer to the mask image.
<i>maskStep</i>	Distance in bytes between starts of consecutive lines in the mask image.
<i>pSrcDst</i>	Pointer to the destination (accumulator) image ROI.
<i>srcDstStep</i>	Distance in bytes between starts of consecutive lines in the accumulator image.
<i>roiSize</i>	Size of the image ROI in pixels.

### Description

The function `ippiAddSquare` is declared in the `ippcv.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function adds squared pixel values of the source image *pSrc* to floating-point pixel values of the accumulator image *pSrcDst* as follows:

$$pSrcDst(x,y) = pSrcDst(x,y) + pSrc(x,y)^2$$

Addition of the squared pixel values in case of a masked operation is performed only if the respective mask value is nonzero; otherwise, the accumulator pixel value remains unchanged.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if any of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error when <code>roiSize.width</code> or <code>roiSize.height</code> is negative.
<code>ippStsStepErr</code>	Indicates an error when <code>srcStep</code> , <code>maskStep</code> , or <code>srcDstStep</code> is less than <code>roiSize.width * &lt;pixelSize&gt;</code> .
<code>ippStsNotEvenStepErr</code>	Indicates an error condition if one of step values for floating-point images cannot be divided by 4.

## AddProduct

*Adds product of pixel values of two source images to floating-point pixel values of an accumulator image.*

### Syntax

#### Case 1: In-place operation

```
IppStatus ippiAddProduct_<mod>(const Ipp<srcDatatype>* pSrc1, int src1Step, const
Ipp<srcDatatype>* pSrc2, int src2Step, Ipp32f* pSrcDst, int srcDstStep, IppiSize roiSize);
```

Supported values for `mod`:

8u32f\_C1IR      8s32f\_C1IR      16u32f\_C1IR      32f\_C1IR

#### Case 2: Masked in-place operation

```
IppStatus ippiAddProduct_<mod>(const Ipp<srcDatatype>* pSrc1, int src1Step, const
Ipp<srcDatatype>* pSrc2, int src2Step, const Ipp8u* pMask, int maskStep, Ipp32f* pSrcDst,
int srcDstStep, IppiSize roiSize);
```

Supported values for `mod`:

8u32f\_C1IMR      8s32f\_C1IMR      16u32f\_C1IMR      32f\_C1IMR

### Parameters

<code>pSrc1, pSrc2</code>	Pointers to the source images ROI.
<code>src1Step, src2Step</code>	Distances in bytes between starts of consecutive lines in the source images.
<code>pMask</code>	Pointer to the mask image.
<code>maskStep</code>	Distance in bytes between starts of consecutive lines in the mask image.
<code>pSrcDst</code>	Pointer to the destination (accumulator) image ROI.
<code>srcDstStep</code>	Distance in bytes between starts of consecutive lines in the accumulator image.

*roiSize*                                      Size of the image ROI in pixels.

## Description

The function `ippiAddProduct` is declared in the `ippcv.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function adds the product of pixel values of two source images *pSrc1* and *pSrc2* to floating-point pixel values of the accumulator image *pSrcDst* as given by:

$$pSrcDst(x,y) = pSrcDst(x,y) + pSrc1(x,y) * pSrc2(x,y)$$

The products of pixel values in case of a masked operation are added only if the respective mask value is nonzero; otherwise, the accumulator pixel value remains unchanged.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when any of the specified pointers is <i>null</i> .
<code>ippStsSizeErr</code>	Indicates an error when <i>roiSize.width</i> or <i>roiSize.height</i> is negative.
<code>ippStsStepErr</code>	Indicates an error if <i>src1Step</i> , <i>src2Step</i> , <i>maskStep</i> , or <i>srcDdstStep</i> is less than <i>roiSize.width</i> * <i>&lt;pixelSize&gt;</i> .
<code>ippStsNotEvenStepErr</code>	Indicates an error condition if one of step values for floating-point images cannot be divided by 4.

## AddWeighted

*Adds weighted pixel values of a source image to floating-point pixel values of an accumulator image.*

---

### Syntax

#### Case 1: In-place operation

```
ippStatus ippiAddWeighted_<mod>(const Ipp<srcDatatype>* pSrc, int srcStep, Ipp32f* pSrcDst,
int srcDstStep, IppiSize roiSize, Ipp32f alpha);
```

Supported values for *mod*:

8u32f\_C1IR      8s32f\_C1IR      16u32f\_C1IR      32f\_C1IR

#### Case 2: Masked in-place operation

```
IppStatus ippiAddWeighted_<mod>(const Ipp<srcDatatype>* pSrc, int srcStep, const Ipp8u*
pMask, int maskStep, Ipp32f* pSrcDst, int srcDstStep, IppiSize roiSize, Ipp32f alpha);
```

Supported values for *mod*:

8u32f\_C1IMR      8s32f\_C1IMR      16u32f\_C1IMR      32f\_C1IMR



**Case 3: Not-in-place operation**

```
ippStatus ippiAddWeighted_32f_C1R(const Ipp32f* pSrc1, int src1Step, const Ipp32f* pSrc2,
int src2Step, Ipp32f* pDst, int dstStep, IppiSize roiSize, Ipp32f alpha);
```

**Parameters**

<i>pSrc</i>	Pointer to the source image ROI for the in-place operation.
<i>pSrc1, pSrc2</i>	Pointers to the ROI in the source images.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image for the in-place operation.
<i>src1Step, src2Step</i>	Distance in bytes between starts of consecutive lines in the source images.
<i>pMask</i>	Pointer to the mask image.
<i>maskStep</i>	Distance in bytes between starts of consecutive lines in the mask image.
<i>pSrcDst</i>	Pointer to the destination (accumulator) image ROI for the in-place operation.
<i>srcDstStep</i>	Distance in bytes between starts of consecutive lines in the accumulator image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>roiSize</i>	Size of the image ROI in pixels.
<i>alpha</i>	Weight $\alpha$ of the source image.

**Description**

The function `ippiAddWeighted` is declared in the `ippcv.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function adds pixel values of the source image *pSrc1* multiplied by a weight factor *alpha* to pixel values of the image *pSrc2* multiplied by  $(1-\alpha)$  and stores result in the *pDst* as follows:

$$pDst(x,y) = pSrc1(x,y) * \alpha + pSrc2(x,y) * (1 - \alpha).$$

The in-place flavors of the function adds pixel values of the source image *pSrc* multiplied by a weight factor *alpha* to floating-point pixel values of the accumulator image *pSrcDst* multiplied by  $(1-\alpha)$  as follows:

$$pSrcDst(x,y) = pSrcDst(x,y) * (1-\alpha) + pSrc(x,y) * \alpha$$

Addition of the weighted pixel values in case of a masked operation is performed only if the respective mask value is nonzero; otherwise, the accumulator pixel value remains unchanged.

**Return Values**

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when any of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error when <i>roiSize.width</i> or <i>roiSize.height</i> is equal to 0 or negative.

<code>ippStsStepErr</code>	Indicates an error when one of the step values is equal to zero, or is less than <code>roiSize.width * &lt;pixelSize&gt;</code> .
<code>ippStsNotEvenStepErr</code>	Indicates an error when one of step values for floating-point images cannot be divided by 4.

## Mul

*Multiplies pixel values of two images.*

---

### Syntax

#### Case 1: Not-in-place operation on integer or complex data

```
IppStatus ippMul_<mod>(const Ipp<datatype>* pSrc1, int src1Step, const Ipp<datatype>* pSrc2, int src2Step, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, int scaleFactor);
```

Supported values for mod:

<code>8u_C1RSfs</code>	<code>16u_C1RSfs</code>	<code>16s_C1RSfs</code>	<code>16sc_C1RSfs</code>	<code>32sc_C1RSfs</code>
<code>8u_C3RSfs</code>	<code>16u_C3RSfs</code>	<code>16s_C3RSfs</code>	<code>16sc_C3RSfs</code>	<code>32sc_C3RSfs</code>
<code>8u_AC4RSfs</code>	<code>16u_AC4RSfs</code>	<code>16s_AC4RSfs</code>	<code>16sc_AC4RSfs</code>	<code>32sc_AC4RSfs</code>
<code>8u_C4RSfs</code>	<code>16u_C4RSfs</code>	<code>16s_C4RSfs</code>		

#### Case 2: Not-in-place operation on floating-point or complex data

```
IppStatus ippMul_<mod>(const Ipp<datatype>* pSrc1, int src1Step, const Ipp<datatype>* pSrc2, int src2Step, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);
```

Supported values for mod:

<code>32f_C1R</code>	<code>32fc_C1R</code>
<code>32f_C3R</code>	<code>32fc_C3R</code>
<code>32f_AC4R</code>	<code>32fc_AC4R</code>
<code>32f_C3R</code>	

#### Case 3: In-place operation on integer or complex data

```
IppStatus ippMul_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize, int scaleFactor);
```

Supported values for mod:

<code>8u_C1IRSfs</code>	<code>16u_C1IRSfs</code>	<code>16s_C1IRSfs</code>	<code>16sc_C1IRSfs</code>	<code>32sc_C1IRSfs</code>
<code>8u_C3IRSfs</code>	<code>16u_C3IRSfs</code>	<code>16s_C3IRSfs</code>	<code>16sc_C3IRSfs</code>	<code>32sc_C3IRSfs</code>

8u_AC4IRSfs	16u_AC4IRSfs	16s_AC4IRSfs	16sc_AC4IRSfs	32sc_AC4IRSfs
8u_C4IRSfs	16u_C4IRSfs	16s_C4IRSfs		

#### Case 4: In-place operation on floating-point or complex data

```
IppStatus ippiMul_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pSrcDst,
int srcDstStep, IppiSize roiSize);
```

Supported values for `mod`:

32f_C1IR	32fc_C1IR
32f_C3IR	32fc_C3IR
32f_AC4IR	32fc_AC4IR
32f_C4IR	

#### Parameters

<i>pSrc, pSrc1, pSrc2</i>	Pointers to the source images ROI.
<i>srcStep, src1Step, src2Step</i>	Distances in bytes between starts of consecutive lines in the source images.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>pSrcDst</i>	Pointer to the source and destination image ROI for the in-place operation.
<i>srcDstStep</i>	Distance in bytes between starts of consecutive lines in the source and destination image for the in-place operation.
<i>roiSize</i>	Size of the source and destination ROI in pixels.
<i>scaleFactor</i>	Scale factor (see <a href="#">Integer Result Scaling</a> ).

#### Description

The function `ippiMul` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function multiplies corresponding pixel values of two source image buffers and places the results in a destination buffer. In case of operations on integer data, the resulting values are scaled by *scaleFactor*.

For complex data, the function processes both real and imaginary parts of pixel values.



**NOTE.** Step values must be positive for functions that operate on complex data.

Note that the functions with `AC4` descriptor do not process alpha channels.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if any of the pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <code>roiSize</code> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if any of the specified buffer step values is zero or negative for functions that operate on complex data.

## MulC

*Multiplies pixel values of an image by a constant.*

---

### Syntax

#### Case 1: Not-in-place operation on one-channel integer or complex data

```
IppStatus ippMulC_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype> value,
Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, int scaleFactor);
```

Supported values for `mod`:

8u\_C1RSfs      16u\_C1RSfs      16s\_C1RSfs      16sc\_C1RSfs      32sc\_C1RSfs

#### Case 2: Not-in-place operation on multi-channel integer or complex data

```
IppStatus ippMulC_<mod>(const Ipp<datatype>* pSrc, int srcStep, const Ipp<datatype>
value[3], Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, int scaleFactor);
```

Supported values for `mod`:

8u\_C3RSfs      16u\_C3RSfs      16s\_C3RSfs      16sc\_C3RSfs      32sc\_C3RSfs

8u\_AC4RSfs      16u\_AC4RSfs      16s\_AC4RSfs      16sc\_AC4RSfs      32sc\_AC4RSfs

```
IppStatus ippMulC_<mod>(const Ipp<datatype>* pSrc, int srcStep, const Ipp<datatype>
value[4], Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, int scaleFactor);
```

Supported values for `mod`:

8u\_C4RSfs      16u\_C4RSfs      16s\_C4RSfs

#### Case 3: Not-in-place operation on one-channel floating-point or complex data

```
IppStatus ippMulC_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype> value,
Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);
```

Supported values for `mod`:

32f\_C1R      32fc\_C1R

**Case 4: Not-in-place operation on multi-channel floating-point or complex data**

```
IppStatus ippiMulC_<mod>(const Ipp<datatype>* pSrc, int srcStep, const Ipp<datatype>
value[3], Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);
```

Supported values for mod:

```
32f_C3R    32fc_C3R
32f_AC4R   32fc_AC4R
```

```
IppStatus ippiMulC_32f_C4R(const Ipp32f* pSrc, int srcStep, const Ipp32f value[4], Ipp32f*
pDst, int dstStep, IppiSize roiSize);
```

**Case 5: In-place operation on one-channel integer or complex data**

```
IppStatus ippiMulC_<mod>(Ipp<datatype> value, Ipp<datatype>* pSrcDst, int srcdstStep,
IppiSize roiSize, int scaleFactor);
```

Supported values for mod:

```
8u_C1IRSfs    16u_C1IRSfs    16s_C1IRSfs    16sc_C1IRSfs    32sc_C1IRSfs
```

**Case 6: In-place operation on multi-channel integer or complex data**

```
IppStatus ippiMulC_<mod>(const Ipp<datatype> value[3], Ipp<datatype>* pSrcDst, int
srcdstStep, IppiSize roiSize, int scaleFactor);
```

Supported values for mod:

```
8u_C3IRSfs    16u_C3IRSfs    16s_C3IRSfs    16sc_C3IRSfs    32sc_C3IRSfs
8u_AC4IRSfs    16u_AC4IRSfs    16s_AC4IRSfs    16sc_AC4IRSfs    32sc_AC4IRSfs
```

```
IppStatus ippiMulC_<mod>(const Ipp<datatype> value[4], Ipp<datatype>* pSrcDst, int
srcdstStep, IppiSize roiSize, int scaleFactor);
```

Supported values for mod:

```
8u_C4IRSfs 16u_C4IRSfs 16s_C4IRSfs
```

**Case 7: In-place operation on one-channel floating-point or complex data**

```
IppStatus ippiMulC_<mod>(Ipp<datatype> value, Ipp<datatype>* pSrcDst, int srcdstStep,
IppiSize roiSize);
```

Supported values for mod:

```
32f_C1IR    32fc_C1IR
```

**Case 8: In-place operation on multi-channel floating-point or complex data**

```
IppStatus ippiMulC_<mod>(const Ipp<datatype> value[3], Ipp<datatype>* pSrcDst, int
srcdstStep, IppiSize roiSize);
```

Supported values for `mod`:

```
32f_C3IR    32fc_C3IR
32f_AC4IR   32fc_AC4IR
```

```
IppStatus ippiMulC_32f_C4IR(const Ipp32f value[4], Ipp32f* pSrcDst, int srcdstStep, IppiSize
roiSize);
```

**Parameters**

<code>pSrc</code>	Pointer to the source image ROI.
<code>srcStep</code>	Distance in bytes between starts of consecutive lines in the source image.
<code>value</code>	The constant value to add to image pixel values (constant vector in case of multi-channel images).
<code>pDst</code>	Pointer to the destination image ROI.
<code>dstStep</code>	Distance in bytes between starts of consecutive lines in the destination image.
<code>pSrcDst</code>	Pointer to the source and destination image ROI for the in-place operation.
<code>srcdstStep</code>	Distance in bytes between starts of consecutive lines in the source and destination image for the in-place operation.
<code>roiSize</code>	Size of the source and destination ROI in pixels.
<code>scaleFactor</code>	Scale factor (see <a href="#">Integer Result Scaling</a> ).

**Description**

The function `ippiMulC` is declared in the `ippi.h` file. It operates with ROI (see Regions of [Interest in Intel IPP](#)).

This function multiplies pixel values of an image by a constant `value`. For multi-channel images, pixel channel values are multiplied by the components of a constant vector `value`. For complex data, the function processes both real and imaginary parts of pixel values.



**NOTE.** Step values must be positive for functions that operate on complex data.

---

In case of operations on integer data, the resulting values are scaled by `scaleFactor`.

Note that the functions with `AC4` descriptor do not process alpha channels.

**Return Values**

`ippStsNoErr` Indicates no error. Any other value indicates an error or a warning.

<code>ippStsNullPtrErr</code>	Indicates an error condition if any of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <code>roiSize</code> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if any of the specified buffer step values is zero or negative for functions that operate on complex data.

## MulScale

*Multiplies pixel values of two images and scales the products.*

---

### Syntax

#### Case 1: Not-in-place operation

```
IppStatus ippMulScale_<mod>(const Ipp<datatype>* pSrc1, int src1Step, const Ipp<datatype>* pSrc2, int src2Step, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);
```

Supported values for `mod`:

<code>8u_C1R</code>	<code>16u_C1R</code>
<code>8u_C3R</code>	<code>16u_C3R</code>
<code>8u_C4R</code>	<code>16u_C4R</code>
<code>8u_AC4R</code>	<code>16u_AC4R</code>

#### Case 2: In-place operation

```
IppStatus ippMulScale_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize);
```

Supported values for `mod`:

<code>8u_C1IR</code>	<code>16u_C1IR</code>
<code>8u_C3IR</code>	<code>16u_C3IR</code>
<code>8u_C4IR</code>	<code>16u_C4IR</code>
<code>8u_AC4IR</code>	<code>16u_AC4IR</code>

### Parameters

<code>pSrc, pSrc1, pSrc2</code>	Pointers to the source images ROI.
<code>srcStep, src1Step, src2Step</code>	Distances in bytes between starts of consecutive lines in the source images.
<code>pDst</code>	Pointer to the destination image ROI.

<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>pSrcDst</i>	Pointer to the source and destination image ROI for the in-place operation.
<i>srcDstStep</i>	Distance in bytes between starts of consecutive lines in the source and destination image for the in-place operation.
<i>roiSize</i>	Size of the source and destination ROI in pixels.

## Description

The function `ippiMulScale` is declared in the `ippi.h` file. It operates with ROI (see Regions of [Interest in Intel IPP](#)).

This function multiplies corresponding pixel values of two input buffers and scales the products using the following formula:

$$dst\_pixel = src1\_pixel * src2\_pixel / max\_val,$$

where *src1\_pixel* and *src2\_pixel* are pixel values of the source buffers, *dst\_pixel* is the resultant pixel value, and *max\_val* is the maximum value of the pixel data range (see [Table “Image Data Types and Ranges”](#) for details). The function is implemented for 8-bit and 16-bit unsigned data types only.

Note that the functions with AC4 descriptor do not process alpha channels.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if any of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if any of the specified buffer step values is zero or negative.

## MulCScale

*Multiplies pixel values of an image by a constant and scales the products.*

---

## Syntax

### Case 1: Not-in-place operation on one-channel data

```
ippStatus ippiMulCScale_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype> value,
Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);
```

Supported values for `mod`:

8u\_C1R      16u\_C1R



**Case 2: Not-in-place operation on multi-channel data**

```
IppStatus ippiMulCScale_<mod>(const Ipp<datatype>* pSrc, int srcStep, const Ipp<datatype>
value[3], Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);
```

Supported values for mod:

```
8u_C3R      16u_C3R
8u_AC4R     16u_AC4R
```

```
IppStatus ippiMulCScale_<mod>(const Ipp<datatype>* pSrc, int srcStep, const Ipp<datatype>
value[4], Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);
```

Supported values for mod:

```
8u_C4R      16u_C4R
```

**Case 3: In-place operation on one-channel data**

```
IppStatus ippiMulCScale_<mod>(Ipp<datatype> value, const Ipp<datatype>* pSrcDst, int
srcDstStep, IppiSize roiSize);
```

Supported values for mod:

```
8u_C1IR     16u_C1IR
```

**Case 4: In-place operation on multi-channel data**

```
IppStatus ippiMulCScale_<mod>(const Ipp<datatype> value[3], const Ipp<datatype>* pSrcDst,
int srcDstStep, IppiSize roiSize);
```

Supported values for mod:

```
8u_C3IR     16u_C3IR
8u_AC4IR    16u_AC4IR
```

```
IppStatus ippiMulCScale_<mod>(const Ipp<datatype> value[4], const Ipp<datatype>* pSrcDst,
int srcDstStep, IppiSize roiSize);
```

Supported values for mod:

```
8u_C4IR     16u_C4IR
```

**Parameters**

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.

<i>value</i>	The constant value to multiply each pixel value in a source image (constant vector in case of 3- or four-channel images).
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>pSrcDst</i>	Pointer to the source and destination image ROI for the in-place operation.
<i>srcDstStep</i>	Distance in bytes between starts of consecutive lines in the source and destination image for the in-place operation.
<i>roiSize</i>	Size of the source and destination ROI in pixels.

## Description

The function `ippiMulCScale` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function multiplies pixel values in the input buffer by a constant *value* and scales the products using the following formula:

$$dst\_pixel = src\_pixel * value / max\_val,$$

where *src\_pixel* is a pixel values of the source buffer, *dst\_pixel* is the resultant pixel value, and *max\_val* is the maximum value of the pixel data range (see [Table “Image Data Types and Ranges”](#) for details).

The function is implemented for 8-bit and 16-bit unsigned data types only. It can be used to multiply pixel values by a number between 0 and 1.

Note that the functions with AC4 descriptor do not process alpha channels.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if the <i>roiSize</i> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if any of the specified buffer step values is zero or negative.

## Example

The code example below shows how to use the function `ippiMulCScale_8u_C1R`.

```
void func_mulcscale()
{
    IppiSize ROI = {8,4};
    IppiSize ROI2 = {5,4};
    Ipp8u src[8*4];
    Ipp8u dst[8*4];
    Ipp8u v = 100;

    ippiSet_8u_C1R(100,src,8,ROI);
    ippiSet_8u_C1R(0,dst,8,ROI);
    ippiMulCScale_8u_C1R(src,8,v,dst,8,ROI2);
}
```

Result:

src1	dst
100 100 100 100 100 100 100 100	39 39 39 39 39 0 0 0
100 100 100 100 100 100 100 100	39 39 39 39 39 0 0 0
100 100 100 100 100 100 100 100	39 39 39 39 39 0 0 0
100 100 100 100 100 100 100 100	39 39 39 39 39 0 0 0

## Sub

*Subtracts pixel values of two images.*

### Syntax

#### Case 1: Not-in-place operation on integer or complex data

```
IppStatus ippiSub_<mod>(const Ipp<datatype>* pSrc1, int src1Step, const Ipp<datatype>*
pSrc2, int src2Step, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, int scaleFactor);
```

Supported values for mod

8u_C1RSfs	16u_C1RSfs	16s_C1RSfs	16sc_C1RSfs	32sc_C1RSfs
8u_C3RSfs	16u_C3RSfs	16s_C3RSfs	16sc_C3RSfs	32sc_C3RSfs
8u_AC4RSfs	16u_AC4RSfs	16s_AC4RSfs	16sc_AC4RSfs	32sc_AC4RSfs
8u_C4RSfs	16u_C4RSfs	16s_C4RSfs		

#### Case 2: Not-in-place operation on floating-point or complex data

```
IppStatus ippiSub_<mod>(const Ipp<datatype>* pSrc1, int src1Step, const Ipp<datatype>*
pSrc2, int src2Step, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);
```

Supported values for mod:

32f_C1R	32fc_C1R
32f_C3R	32fc_C3R
32f_AC4R	32fc_AC4R
32f_C4R	

#### Case 3: In-place operation on integer or complex data

```
IppStatus ippiSub_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pSrcDst,
int srcDstStep, IppiSize roiSize, int scaleFactor);
```

Supported values for mod:

8u_C1IRSfs	16u_C1IRSfs	16s_C1IRSfs	16sc_C1IRSfs	32sc_C1IRSfs
------------	-------------	-------------	--------------	--------------

8u_C3IRSfs	16u_C3IRSfs	16s_C3IRSfs	16sc_C3IRSfs	32sc_C3IRSfs
8u_AC4IRSfs	16u_AC4IRSfs	16s_AC4IRSfs	16sc_AC4IRSfs	32sc_AC4IRSfs
8u_C4IRSfs	16u_C4IRSfs	16s_C4IRSfs		

**Case 4: In-place operation on floating-point or complex data**

```
IppStatus ippiSub_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pSrcDst,
int srcDstStep, IppiSize roiSize);
```

Supported values for `mod`:

32f_C1IR	32fc_C1IR
32f_C3IR	32fc_C3IR
32f_AC4IR	32fc_AC4IR
32f_C4IR	

**Parameters**

<i>pSrc, pSrc1, pSrc2</i>	Pointers to the source images ROI.
<i>srcStep, src1Step, src2Step</i>	Distances in bytes between starts of consecutive lines in the source images.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>pSrcDst</i>	Pointer to the source and destination image ROI for the in-place operation.
<i>srcDstStep</i>	Distance in bytes between starts of consecutive lines in the source and destination image for the in-place operation.
<i>roiSize</i>	Size of the source and destination ROI in pixels.
<i>scaleFactor</i>	Scale factor (see <a href="#">Integer Result Scaling</a> ).

**Description**

The function `ippiSub` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function subtracts pixel values of the source buffer *pSrc1* from the corresponding pixel values of the buffer *pSrc2* and places the result in the destination buffer *pDst*. For in-place operations, the values in *pSrc* are subtracted from the values in *pSrcDst* and the results are placed into *pSrcDst*. For complex data, the function processes both real and imaginary parts of pixel values.



**NOTE.** Step values must be positive for functions that operate on complex data.

---

In case of operations on integer data, the resulting values are scaled by *scaleFactor*.

Note that the functions with AC4 descriptor do not process alpha channels.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <code>roiSize</code> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if any of the specified buffer step values is zero or negative for functions that operate on complex data.

## SubC

*Subtracts a constant from pixel values of an image.*

### Syntax

#### Case 1: Not-in-place operation on one-channel integer or complex data

```
IppStatus ippSubC_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype> value,
Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, int scaleFactor);
```

Supported values for `mod`:

8u\_C1RSfs      16u\_C1RSfs      16s\_C1RSfs      16sc\_C1RSfs      32sc\_C1RSfs

#### Case 2: Not-in-place operation on multi-channel integer or complex data

```
IppStatus ippSubC_<mod>(const Ipp<datatype>* pSrc, int srcStep, const Ipp<datatype>
value[3], Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, int scaleFactor);
```

Supported values for `mod`:

8u\_C3RSfs      16u\_C3RSfs      16s\_C3RSfs      16sc\_C3RSfs      32sc\_C3RSfs  
8u\_AC4RSfs      16u\_AC4RSfs      16s\_AC4RSfs      16sc\_AC4RSfs      32sc\_AC4RSfs

```
IppStatus ippSubC_<mod>(const Ipp<datatype>* pSrc, int srcStep, const Ipp<datatype>
value[4], Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, int scaleFactor);
```

Supported values for `mod`:

8u\_C4RSfs      16u\_C4RSfs      16s\_C4RSfs

**Case 3: Not-in-place operation on one-channel floating-point or complex data**

```
IppStatus ippiSubC_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype> value,
Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);
```

Supported values for mod:

32f\_C1R      32fc\_C1R

**Case 4: Not-in-place operation on multi-channel floating-point or complex data**

```
IppStatus ippiSubC_<mod>(const Ipp<datatype>* pSrc, int srcStep, const Ipp<datatype>
value[3], Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);
```

Supported values for mod:

32f\_C3R      32fc\_C3R  
32f\_AC4R      32fc\_AC4R

```
IppStatus ippiSubC_32f_C4R(const Ipp32f* pSrc, int srcStep, const Ipp32f value[4], Ipp32f*
pDst, int dstStep, IppiSize roiSize);
```

**Case 5: In-place operation on one-channel integer or complex data**

```
IppStatus ippiSubC_<mod>(Ipp<datatype> value, Ipp<datatype>* pSrcDst, int srcDstStep,
IppiSize roiSize, int scaleFactor);
```

Supported values for mod:

8u\_C1IRSfs      16u\_C1IRSfs      16s\_C1IRSfs      16sc\_C1IRSfs      32sc\_C1IRSfs

**Case 6: In-place operation on multi-channel integer or complex data**

```
IppStatus ippiSubC_<mod>(const Ipp<datatype> value[3], Ipp<datatype>* pSrcDst, int
srcDstStep, IppiSize roiSize, int scaleFactor);
```

Supported values for mod:

8u\_C3IRSfs      16u\_C3IRSfs      16s\_C3IRSfs      16sc\_C3IRSfs      32sc\_C3IRSfs  
8u\_AC4IRSfs      16u\_AC4IRSfs      16s\_AC4IRSfs      16sc\_AC4IRSfs      32sc\_AC4IRSfs

```
IppStatus ippiSubC_<mod>(const Ipp<datatype> value[4], Ipp<datatype>* pSrcDst, int
srcDstStep, IppiSize roiSize, int scaleFactor);
```

Supported values for mod:

8u\_C4IRSfs      16u\_C4IRSfs      16s\_C4IRSfs

**Case 7: In-place operation on one-channel floating-point or complex data**

```
IppStatus ippiSubC_<mod>(Ipp<datatype> value, Ipp<datatype>* pSrcDst, int srcDstStep,
IppiSize roiSize);
```

Supported values for mod:

```
32f_C1IR      32fc_C1IR
```

**Case 8: In-place operation on multi-channel floating-point or complex data**

```
IppStatus ippiSubC_<mod>(const Ipp<datatype> value[3], Ipp<datatype>* pSrcDst, int
srcDstStep, IppiSize roiSize);
```

Supported values for mod:

```
32f_C3IR      32fc_C3IR
32f_AC4IR      2fc_AC4IR
```

```
IppStatus ippiSubC_32f_C4IR(const Ipp32f value[4], Ipp32f* pSrcDst, int srcDstStep, IppiSize
roiSize);
```

**Parameters**

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>value</i>	The constant value to subtract from each pixel value in a source image (constant vector in case of multi-channel images).
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>pSrcDst</i>	Pointer to the source and destination image ROI for the in-place operation.
<i>srcDstStep</i>	Distance in bytes between starts of consecutive lines in the source and destination image for the in-place operation.
<i>roiSize</i>	Size of the source and destination ROI in pixels.
<i>scaleFactor</i>	Scale factor (see <a href="#">Integer Result Scaling</a> ).

**Description**

The function `ippiSubC` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function changes image intensity by subtracting the constant *value* from pixel values of an image buffer. For multi-channel images, the components of a constant vector *value* are subtracted from pixel channel values. For complex data, the function processes both real and imaginary parts of pixel values.



**NOTE.** Step values must be positive for functions that operate on complex data.

In case of operations on integer data, the resulting values are scaled by *scaleFactor*.

Note that the functions with the AC4 descriptor do not process alpha channels.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if any of the specified buffer step values is zero or negative for functions that operate on complex data.

## Div

*Divides pixel values of an image by pixel values of another image.*

---

## Syntax

### Case 1: Not-in-place operation on integer or complex data

```
IppStatus ippiDiv_<mod>(const Ipp<datatype>* pSrc1, int src1Step, const Ipp<datatype>* pSrc2, int src2Step, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, int scaleFactor);
```

Supported values for *mod*:

<code>8u_C1RSfs</code>	<code>16u_C1RSfs</code>	<code>16s_C1RSfs</code>	<code>16sc_C1RSfs</code>	<code>32sc_C1RSfs</code>
<code>8u_C3RSfs</code>	<code>16u_C3RSfs</code>	<code>16s_C3RSfs</code>	<code>16sc_C3RSfs</code>	<code>32sc_C3RSfs</code>
<code>8u_AC4RSfs</code>	<code>16u_AC4RSfs</code>	<code>16s_AC4RSfs</code>	<code>16sc_AC4RSfs</code>	<code>32sc_AC4RSfs</code>
<code>8u_C4RSfs</code>	<code>16u_C4RSfs</code>	<code>16s_C4RSfs</code>		

### Case 2: Not-in-place operation on floating-point or complex data

```
IppStatus ippiDiv_<mod>(const Ipp<datatype>* pSrc1, int src1Step, const Ipp<datatype>* pSrc2, int src2Step, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);
```

Supported values for *mod*:

<code>32f_C1R</code>	<code>32fc_C1R</code>
<code>32f_C3R</code>	<code>32fc_C3R</code>
<code>32f_AC4R</code>	<code>32fc_AC4R</code>
<code>32f_C4R</code>	



**Case 3: In-place operation on integer or complex data**

```
IppStatus ippiDiv_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pSrcDst,
int srcDstStep, IppiSize roiSize, int scaleFactor);
```

Supported values for mod:

8u_C1IRSfs	16u_C1IRSfs	16s_C1IRSfs	16sc_C1IRSfs	32sc_C1IRSfs
8u_C3IRSfs	16u_C3IRSfs	16s_C3IRSfs	16sc_C3IRSfs	32sc_C3IRSfs
8u_AC4IRSfs	16u_AC4IRSfs	16s_AC4IRSfs	16sc_AC4IRSfs	32sc_AC4IRSfs
8u_C4IRSfs	16u_C4IRSfs	16s_C4IRSfs		

**Case 4: In-place operation on floating-point or complex data**

```
IppStatus ippiDiv_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pSrcDst,
int srcDstStep, IppiSize roiSize);
```

Supported values for mod:

32f_C1IR	32fc_C1IR
32f_C3IR	32fc_C3IR
32f_AC4IR	32fc_AC4IR
32f_C4IR	

**Parameters**

<i>pSrc, pSrc1, pSrc2</i>	Pointers to the source images ROI.
<i>srcStep, src1Step, src2Step</i>	Distances in bytes between starts of consecutive lines in the source images.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>pSrcDst</i>	Pointer to the source and destination image ROI for the in-place operation.
<i>srcDstStep</i>	Distance in bytes between starts of consecutive lines in the source and destination image for the in-place operation.
<i>roiSize</i>	Size of the source and destination ROI in pixels.
<i>scaleFactor</i>	Scale factor (see <a href="#">Integer Result Scaling</a> ).

**Description**

The function `ippiDiv` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function divides pixel values of the source buffer *pSrc2* by the corresponding pixel values of the buffer *pSrc1* and places the result in a destination buffer *pDst*. For in-place operations, the values in *pSrcDst* are divided by the values in *pSrc* and placed into *pSrcDst*. For complex data, the function processes both real and imaginary parts of pixel values. In case of operations on integer data, the resulting values are scaled by *scaleFactor* and rounded (not truncated). When the function encounters a zero divisor value, the execution is not interrupted. The function returns the warning message and corresponding result value (see appendix A “[Handling of Special Cases](#)” for more information).

Note that the functions with the AC4 descriptor do not process alpha channels.

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if any of the specified buffer step values is zero or negative.
<code>ippStsDivByZero</code>	Indicates a warning that a divisor value is zero. The function execution is continued.

### Example

The code example below illustrates how the function `ippiDiv` can be used.

```
IppStatus div32f( void ) {
    Ipp32f a[4*3], b[4*3];
    IppiSize roi = {2,2};
    int i;
    for( i=0; i<4*3; ++i ) a[i] = b[i] = (float)i;
    return ippiDiv_32f_C1IR( a, 4*sizeof(Ipp32f), b,
                           4*sizeof(Ipp32f), roi );
}
```

The destination image *b* contains

```
-1.#IND  +1.000  +2.000  +3.000
+1.000   +1.000  +6.000  +7.000
+8.000   +9.000 +10.00  +11.00
```

Console output:

```
-- warning in div32f:
(6) Zero value(s) of divisor in the function Div
```

## Div\_Round

*Divides pixel values of an image by pixel values of another image with different rounding modes.*

### Syntax

#### Case 1: Not-in-place operation on integer data

```
IppStatus ippiDiv_Round_<mod>(const Ipp<datatype>* pSrc1, int src1Step, const Ipp<datatype>* pSrc2, int src2Step, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, IppRoundMode rndMode, int scaleFactor);
```

Supported values for mod:

8u_C1RSfs	16u_C1RSfs	16s_C1RSfs
8u_C3RSfs	16u_C3RSfs	16s_C3RSfs
8u_AC4RSfs	16u_AC4RSfs	16s_AC4RSfs
8u_C4RSfs	16u_C4RSfs	16s_C4RSfs

#### Case 2: In-place operation on integer data

```
IppStatus ippiDiv_Round_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize, IppRoundMode rndMode, int scaleFactor);
```

Supported values for mod:

8u_C1IRSfs	16u_C1IRSfs	16s_C1IRSfs
8u_C3IRSfs	16u_C3IRSfs	16s_C3IRSfs
8u_AC4IRSfs	16u_AC4IRSfs	16s_AC4IRSfs
8u_C4IRSfs	16u_C4IRSfs	16s_C4IRSfs

### Parameters

<i>pSrc, pSrc1, pSrc2</i>	Pointers to the source images ROI.
<i>srcStep, src1Step, src2Step</i>	Distances in bytes between starts of consecutive lines in the source images.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>pSrcDst</i>	Pointer to the source and destination image ROI for the in-place operation.
<i>srcDstStep</i>	Distance in bytes between starts of consecutive lines in the source and destination image for the in-place operation.

<i>roiSize</i>	Size of the source and destination ROI in pixels.						
<i>roundMode</i>	Rounding mode, the following values are possible: <table> <tr> <td><i>ippRndZero</i></td><td>specifies that floating-point values are truncated toward zero,</td></tr> <tr> <td><i>ippRndNear</i></td><td>specifies that floating-point values are rounded to the nearest even integer when the fractional part equals 0.5; otherwise they are rounded to the nearest integer,</td></tr> <tr> <td><i>ippRndFinancial</i></td><td>specifies that floating-point values are rounded down to the nearest integer when the fractional part is less than 0.5, or rounded up to the nearest integer if the fractional part is equal or greater than 0.5.</td></tr> </table>	<i>ippRndZero</i>	specifies that floating-point values are truncated toward zero,	<i>ippRndNear</i>	specifies that floating-point values are rounded to the nearest even integer when the fractional part equals 0.5; otherwise they are rounded to the nearest integer,	<i>ippRndFinancial</i>	specifies that floating-point values are rounded down to the nearest integer when the fractional part is less than 0.5, or rounded up to the nearest integer if the fractional part is equal or greater than 0.5.
<i>ippRndZero</i>	specifies that floating-point values are truncated toward zero,						
<i>ippRndNear</i>	specifies that floating-point values are rounded to the nearest even integer when the fractional part equals 0.5; otherwise they are rounded to the nearest integer,						
<i>ippRndFinancial</i>	specifies that floating-point values are rounded down to the nearest integer when the fractional part is less than 0.5, or rounded up to the nearest integer if the fractional part is equal or greater than 0.5.						

*scaleFactor*                      Scale factor (see [Integer Result Scaling](#)).

## Description

The function `ippiDiv_Round` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function divides pixel values of the source buffer *pSrc2* by the corresponding pixel values of the buffer *pSrc1* and places the result in a destination buffer *pDst*. For in-place operations, the values in *pSrcDst* are divided by the values in *pSrc* and placed into *pSrcDst*. The resulting values are scaled by *scaleFactor* and rounded using the rounding method specified by the parameter *roundMode*. When the function encounters a zero divisor value, the execution is not interrupted. The function returns the warning message and corresponding result value (see appendix A “[Handling of Special Cases](#)” for more information).

Note that the functions with the AC4 descriptor do not process alpha channelss.

## Return Values

<i>ippStsNoErr</i>	Indicates no error. Any other value indicates an error or a warning.
<i>ippStsNullPtrErr</i>	Indicates an error condition if one of the specified pointers is NULL.
<i>ippStsSizeErr</i>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.
<i>ippStsStepErr</i>	Indicates an error condition if any of the specified buffer step values is zero or negative.
<i>ippStsDivByZero</i>	Indicates a warning that a divisor value is zero. The function execution is continued.
<i>ippStsStsRoundModeNotSupportedErr</i>	Indicates an error condition if the <i>roundMode</i> has an illegal value.

## DivC

*Divides pixel values of an image by a constant.*

### Syntax

#### Case 1: Not-in-place operation on one-channel integer or complex data

```
IppStatus ippiDivC_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype> value,
Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, int scaleFactor);
```

Supported values for mod:

8u\_C1RSfs      16u\_C1RSfs      16s\_C1RSfs      16sc\_C1RSfs      32sc\_C1RSfs

#### Case 2: Not-in-place operation on multi-channel integer or complex data

```
IppStatus ippiDivC_<mod>(const Ipp<datatype>* pSrc, int srcStep, const Ipp<datatype>
value[3], Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, int scaleFactor);
```

Supported values for mod:

8u\_C3RSfs      16u\_C3RSfs      16s\_C3RSfs      16sc\_C3RSfs      32sc\_C3RSfs  
8u\_AC4RSfs      16u\_AC4RSfs      16s\_AC4RSfs      16sc\_AC4RSfs      32sc\_AC4RSfs

```
IppStatus ippiDivC_<mod>(const Ipp<datatype>* pSrc, int srcStep, const Ipp<datatype>
value[4], Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, int scaleFactor);
```

Supported values for mod:

8u\_C4RSfs    16u\_C4RSfs    16s\_C4RSfs

#### Case 3: Not-in-place operation on one-channel floating-point or complex data

```
IppStatus ippiDivC_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype> value,
Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);
```

Supported values for mod:

32f\_C1R      32fc\_C1R

#### Case 4: Not-in-place operation on multi-channel floating-point or complex data

```
IppStatus ippiDivC_<mod>(const Ipp<datatype>* pSrc, int srcStep, const Ipp<datatype>
value[3], Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);
```

Supported values for mod:

32f\_C3R      32fc\_C3R  
32f\_AC4R      32fc\_AC4R

```
IppStatus ippiDivC_32f_C4R(const Ipp32f* pSrc, int srcStep, const Ipp32f value[4], Ipp32f* pDst, int dstStep, IppiSize roiSize);
```

#### Case 5: In-place operation on one-channel integer or complex data

```
IppStatus ippiDivC_<mod>(Ipp<datatype> value, Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize, int scaleFactor);
```

Supported values for mod:

8u_C1IRSfs	16u_C1IRSfs	16s_C1IRSfs	16sc_C1IRSfs	32sc_C1IRSfs
------------	-------------	-------------	--------------	--------------

#### Case 6: In-place operation on multi-channel integer or complex data

```
IppStatus ippiDivC_<mod>(const Ipp<datatype> value[3], Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize, int scaleFactor);
```

Supported values for mod:

8u_C3IRSfs	16u_C3IRSfs	16s_C3IRSfs	16sc_C3IRSfs	32sc_C3IRSfs
8u_AC4IRSfs	16u_AC4IRSfs	16s_AC4IRSfs	16sc_AC4IRSfs	32sc_AC4IRSfs

```
IppStatus ippiDivC_<mod>(const Ipp<datatype> value[4], Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize, int scaleFactor);
```

Supported values for mod:

8u_C4IRSfs	16u_C4IRSfs	16s_C4IRSfs
------------	-------------	-------------

#### Case 7: In-place operation on one-channel floating-point or complex data

```
IppStatus ippiDivC_<mod>(Ipp<datatype> value, Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize);
```

Supported values for mod:

32f_C1IR	32fc_C1IR
----------	-----------

#### Case 8: In-place operation on multi-channel floating-point or complex data

```
IppStatus ippiDivC_<mod>(const Ipp<datatype> value[3], Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize);
```

Supported values for mod:

32f_C3IR	32fc_C3IR
32f_AC4IR	32fc_AC4IR

```
IppStatus ippiDivC_32f_C4IR(const Ipp32f value[4], Ipp32f* pSrcDst, int srcDstStep, IppiSize roiSize);
```

## Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>value</i>	The constant value to divide each pixel value in a source buffer (constant vector in case of 3- or four-channel images).
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>pSrcDst</i>	Pointer to the source and destination image ROI for the in-place operation.
<i>srcDstStep</i>	Distance in bytes between starts of consecutive lines in the source and destination image for the in-place operation.
<i>roiSize</i>	Size of the source and destination ROI in pixels.
<i>scaleFactor</i>	Scale factor (see <a href="#">Integer Result Scaling</a> ).

## Description

The function `ippiDiv` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function changes image intensity by dividing pixel values of an image buffer by the constant *value*. For multi-channel images, pixel channel values are divided by the components of a constant vector *value*. For complex data, the function processes both real and imaginary parts of pixel values. In case of operations on integer data, the resulting values are scaled by *scaleFactor* and rounded (not truncated).

When the divisor value is zero, the function execution is aborted and the `ippStsDivByZeroErr` error status is set. Note that in the alpha channel case (AC4), the alpha channels are not processed.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> , <i>dstStep</i> , or <i>srcDstStep</i> has a zero or negative value.
<code>ippStsDivByZeroErr</code>	Indicates an error condition if the divisor value is zero.

## Abs

*Computes absolute pixel values of a source image and places them into the destination image.*

---

### Syntax

#### Case 1: Not-in-place operation

```
IppStatus ippiAbs_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);
```

Supported values for mod:

16s_C1R	32f_C1R
16s_C3R	32f_C3R
16s_C4R	32f_C4R
16s_AC4R	32f_AC4R

#### Case 2: In-place operation

```
IppStatus ippiAbs_<mod>(Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize);
```

Supported values for mod:

16s_C1IR	32f_C1IR
16s_C3IR	32f_C3IR
16s_C4IR	32f_C4IR
16s_AC4IR	32f_AC4IR

### Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>pSrcDst</i>	Pointer to the source and destination image ROI for the in-place operation.
<i>srcDstStep</i>	Distance in bytes between starts of consecutive lines in the source and destination image for the in-place operation.
<i>roiSize</i>	Size of the source and destination ROI in pixels.



## Description

The function `ippiAbs` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function takes the absolute value of each channel in each pixel of the source image ROI and places the result into a destination image ROI. It operates on signed data only. Note that the functions with the `AC4` descriptor do not process alpha channels.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <code>roiSize</code> has a field with zero or negative value.

## AbsDiff

*Calculates absolute difference between two images.*

## Syntax

```
IppStatus ippiAbsDiff_<mod>(const Ipp<datatype>* pSrc1, int src1Step, const Ipp<datatype>* pSrc2, int src2Step, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);
```

Supported values for `mod`:

```
8u_C1R      16u_C1R   32f_C1R
8u_C3R
```

## Parameters

<code>pSrc1</code>	Pointer to the first source image.
<code>src1Step</code>	Distance in bytes between starts of consecutive lines in the first source image.
<code>pSrc2</code>	Pointer to second source image.
<code>src2Step</code>	Distance in bytes between starts of consecutive lines in the second source image.
<code>pDst</code>	Pointer to the destination image.
<code>dstStep</code>	Distance in bytes between starts of consecutive lines in the destination image.
<code>roiSize</code>	Size of the image ROI in pixels.

## Description

The function `ippiAbsDiff` is declared in the `ippcv.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function calculates the absolute pixel-wise difference between two images by the formula:

$pDst(x,y) = \text{abs}(pSrc1(x,y) - pSrc2(x,y)).$

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <code>roiSize</code> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error when <code>src1Step</code> , <code>src2Step</code> or <code>dstStep</code> is less than <code>roiSize.width * &lt;pixelSize&gt;</code> .
<code>ippStsNotEvenStepErr</code>	Indicates an error condition if one of step values for floating-point images cannot be divided by 4.

## AbsDiffC

*Calculates absolute difference between image and scalar value.*

---

### Syntax

```
ippStatus ippiAbsDiffC_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pDst,
int dstStep, IppiSize roiSize, int value);
```

Supported values for `mod`:

`8u_C1R`      `16u_C1R`

```
ippStatus ippiAbsDiffC_32f_C1R(const Ipp32f* pSrc, int srcStep, Ipp32f* pDst, int dstStep,
IppiSize roiSize, Ipp32f value);
```

### Parameters

<code>pSrc</code>	Pointer to the source image.
<code>srcStep</code>	Distance in bytes between starts of consecutive lines in the source image.
<code>pDst</code>	Pointer to the destination image.
<code>dstStep</code>	Distance in bytes between starts of consecutive lines in the destination image.
<code>roiSize</code>	Size of the image ROI in pixels.
<code>value</code>	Scalar value used to decrement each element of the source image.

### Description

The function `ippiAbsDiffC` is declared in the `ippcv.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function calculates the absolute pixel-wise difference between the source image `pSrc` and the scalar `value` by the formula:

$pDst(x,y) = \text{abs}(pSrc(x,y) - \text{value})$ .

The function clips the *value* to the range [0, 255] for the 8u data type, and to the range [0, 65535] for the 16u data type.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error when <i>srcStep</i> or <i>dstStep</i> is less than <i>roiSize.width</i> * <i>&lt;pixelSize&gt;</i> .
<code>ippStsNotEvenStepErr</code>	Indicates an error condition if one of step values for floating-point images cannot be divided by 4.

## Sqr

*Squares pixel values of an image and writes them into the destination image.*

## Syntax

### Case 1: Not-in-place operation on integer data

```
ippStatus ippISqr_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, int scaleFactor);
```

Supported values for mod:

8u\_C1RSfs 16u\_C1RSfs 16s\_C1RSfs

8u\_C3RSfs 16u\_C3RSfs 16s\_C3RSfs

8u\_C4RSfs 16u\_C4RSfs 16s\_C4RSfs

8u\_AC4RSfs 16u\_AC4RSfs 16s\_AC4RSfs

### Case 2: Not-in-place operation on floating-point data

```
ippStatus ippISqr_<mod>(const Ipp32f* pSrc, int srcStep, Ipp32f* pDst, int dstStep, IppiSize roiSize);
```

Supported values for mod:

32f\_C1R

32f\_C3R

32f\_C4R

32f\_AC4R

**Case 3: In-place operation on integer data**

```
IppStatus ippiSqr_<mod>(Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize, int
scaleFactor);
```

Supported values for mod:

```
8u_C1IRSfs 16u_C1IRSfs 16s_C1IRSfs
8u_C3IRSfs 16u_C3IRSfs 16s_C3IRSfs
8u_C4IRSfs 16u_C4IRSfs 16s_C4IRSfs
8u_AC4IRSfs 16u_AC4IRSfs 16s_AC4IRSfs
```

**Case 4: In-place operation on floating-point data**

```
IppStatus ippiSqr_<mod>(Ipp32f* pSrcDst, int srcDstStep, IppiSize roiSize);
```

Supported values for mod:

```
32f_C1IR
32f_C3IR
32f_C4IR
32f_AC4IR
```

**Parameters**

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>pSrcDst</i>	Pointer to the source and destination image ROI for the in-place operation.
<i>srcDstStep</i>	Distance in bytes between starts of consecutive lines in the source and destination image for the in-place operation.
<i>roiSize</i>	Size of the source and destination ROI in pixels.
<i>scaleFactor</i>	Scale factor (see <a href="#">Integer Result Scaling</a> ).

**Description**

The function `ippiSqr` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function squares pixel values of the source image ROI and writes them to the destination image ROI. The function flavors operating on integer data apply fixed scaling defined by *scaleFactor* to the internally computed values, and saturate the results before writing them to the destination image ROI.

Note that the functions with the AC4 descriptor do not process alpha channels.

## Return Values

<code>ippStsNoErr</code>	Indicates an error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <code>roiSize</code> has a field with zero or negative value.

## Sqrt

*Computes square roots of pixel values of a source image and writes them into the destination image.*

## Syntax

### Case 1: Not-in-place operation on integer data

```
ippStatus ippISqrt_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, int scaleFactor);
```

Supported values for mod:

`8u_C1RSfs 16u_C1RSfs 16s_C1RSfs`

`8u_C3RSfs 16u_C3RSfs 16s_C3RSfs`

`8u_AC4RSfs 16u_AC4RSfs 16s_AC4RSfs`

### Case 2: Not-in-place operation on floating-point data

```
ippStatus ippISqrt_<mod>(const Ipp32f* pSrc, int srcStep, Ipp32f* pDst, int dstStep, IppiSize roiSize);
```

Supported values for mod:

`32f_C1R`

`32f_C3R`

`32f_AC4R`

### Case 3: In-place operation on integer data

```
ippStatus ippISqrt_<mod>(Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize, int scaleFactor);
```

Supported values for mod:

`8u_C1IRSfs 16u_C1IRSfs 16s_C1IRSfs`

`8u_C3IRSfs 16u_C3IRSfs 16s_C3IRSfs`

8u\_AC4IRSfs 16u\_AC4IRSfs 16s\_AC4IRSfs

#### Case 4: In-place operation on floating-point data

```
IppStatus ippiSqrt_<mod>(Ipp32f* pSrcDst, int srcDstStep, IppiSize roiSize);
```

Supported values for mod:

32f\_C1IR

32f\_C3IR

32f\_C4IR

32f\_AC4IR

#### Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>pSrcDst</i>	Pointer to the source and destination image ROI for the in-place operation.
<i>srcDstStep</i>	Distance in bytes between starts of consecutive lines in the source and destination image for the in-place operation.
<i>roiSize</i>	Size of the source and destination ROI in pixels.
<i>scaleFactor</i>	Scale factor (see <a href="#">Integer Result Scaling</a> ).

#### Description

The function `ippiSqrt` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function computes square roots of pixel values of the source image ROI and writes them into the destination image ROI. The function flavors operating on integer data apply fixed scaling defined by *scaleFactor* to the internally computed values, and saturate the results before writing them to the destination image ROI. If a source pixel value is negative, the function issues a warning and continues execution with the corresponding result value (see appendix A “[Handling of Special Cases](#)” for more information ).

Note that the functions with the AC4 descriptor do not process alpha channelss.

#### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.

`ippStsSqrtNegArg`

Indicates a warning that a source pixel has a negative value.

## Ln

*Computes the natural logarithm of pixel values in a source image and writes the results into the destination image.*

---

### Syntax

#### Case 1: Not-in-place operation on integer data

```
IppStatus ippILn_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, int scaleFactor);
```

Supported values for `mod`:

<code>8u_C1RSfs</code>	<code>16u_C1RSfs</code>	<code>16s_C1RSfs</code>
<code>8u_C3RSfs</code>	<code>16u_C3RSfs</code>	<code>16s_C3RSfs</code>

#### Case 2: Not-in-place operation on floating-point data

```
IppStatus ippILn_<mod>(const Ipp32f* pSrc, int srcStep, Ipp32f* pDst, int dstStep, IppiSize roiSize);
```

Supported values for `mod`:

<code>32f_C1R</code>
<code>32f_C3R</code>

#### Case 3: In-place operation on integer data

```
IppStatus ippILn_<mod>(Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize, int scaleFactor);
```

Supported values for `mod`:

<code>8u_C1IRSfs</code>	<code>16u_C1IRSfs</code>	<code>16s_C1IRSfs</code>
<code>8u_C3IRSfs</code>	<code>16u_C3IRSfs</code>	<code>16s_C3IRSfs</code>

#### Case 4: In-place operation on floating-point data

```
IppStatus ippILn_<mod>(Ipp32f* pSrcDst, int srcDstStep, IppiSize roiSize);
```

Supported values for `mod`:

<code>32f_C1IR</code>
<code>32f_C3IR</code>

## Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>pSrcDst</i>	Pointer to the source and destination image ROI for the in-place operation.
<i>srcDstStep</i>	Distance in bytes between starts of consecutive lines in the source and destination image for the in-place operation.
<i>roiSize</i>	Size of the source and destination ROI in pixels.
<i>scaleFactor</i>	Scale factor (see <a href="#">Integer Result Scaling</a> ).

## Description

The function `ippiLn` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function computes natural logarithms of pixel values of the source image ROI and writes the resultant values to the destination image ROI. The function flavors operating on integer data apply fixed scaling defined by *scaleFactor* to the internally computed values, and saturate the results before writing them to the destination image ROI.

If a source pixel value is zero or negative, the function issues a corresponding warning and continues execution with the corresponding result value (see appendix A “[Handling of Special Cases](#)” for more information).

When several inputs have zero or negative value, the status code returned by the function corresponds to the first encountered case as illustrated in the code example below.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> , <i>dstStep</i> , or <i>srcDstStep</i> has a zero or negative value.
<code>ippStsLnZeroArg</code>	Indicates a warning that a source pixel has a zero value.
<code>ippStsLnNegArg</code>	Indicates a warning that a source pixel has a negative value.

## Example

The code example below shows how to use `Ln` function.

```
IppStatus ln( void ) {
    Ipp32f img[8*8];
    IppiSize roi = { 8, 8 };
    IppStatus st;
    ippiSet_32f_C1R( (float)IPP_E, img, 8*4, roi );
    img[0] = -0;
```



```

    img[1] = -1;
    st = ippiLn_32f_C1IR( img, 8*sizeof(Ipp32f), roi );
    printf( "%f %f %f\n", img[0], img[1], img[2] );
    return st;
}

```

Output values:

```
-1.#INF00 -1.#IND00 1.000000
```

Status value and message:

```
(7) Zero value(s) of argument in the Ln function
```

## Exp

*Computes the exponential of pixel values in a source image and writes the results into the destination image.*

### Syntax

#### Case 1: Not-in-place operation on integer data

```
IppStatus ippiExp_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize , int scaleFactor);
```

Supported values for mod:

```
8u_C1RSfs 16u_C1RSfs 16s_C1RSfs
```

```
8u_C3RSfs 16u_C3RSfs 16s_C3RSfs
```

#### Case 2: Not-in-place operation on floating-point data

```
IppStatus ippiExp_<mod>(const Ipp32f* pSrc, int srcStep, Ipp32f* pDst, int dstStep, IppiSize roiSize);
```

Supported values for mod:

```
32f_C1R
```

```
32f_C3R
```

#### Case 3: In-place operation on integer data

```
IppStatus ippiExp_<mod>(Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize, int scaleFactor);
```

Supported values for mod:

```
8u_C1IRSfs 16u_C1IRSfs 16s_C1IRSfs
```

```
8u_C3IRSfs 16u_C3IRSfs 16s_C3IRSfs
```

**Case 4: In-place operation on floating-point data**

```
IppStatus ippiExp_<mod>(Ipp32f* pSrcDst, int srcDstStep, IppiSize roiSize);
```

Supported values for `mod`:

`32f_C1IR`

`32f_C3IR`

**Parameters**

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>pSrcDst</i>	Pointer to the source and destination image ROI for the in-place operation.
<i>srcDstStep</i>	Distance in bytes between starts of consecutive lines in the source and destination image for the in-place operation.
<i>roiSize</i>	Size of the source and destination ROI in pixels.
<i>scaleFactor</i>	Scale factor (see <a href="#">Integer Result Scaling</a> ).

**Description**

The function `ippiExp` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function computes  $e$  to the power of pixel values of the source image ROI and writes the resultant values into the destination image ROI. The function flavors operating on integer data apply fixed scaling defined by *scaleFactor* to the internally computed values, and saturate the results before writing them to the destination image ROI.

When the overflow occurs, the resultant value is determined in accordance with the data type (see appendix A “[Handling of Special Cases](#)” for more information).

**Return Values**

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> , <i>dstStep</i> , or <i>srcDstStep</i> has a zero or negative value.

## Complement

*Converts negative number from the complement to direct code.*

---

### Syntax

```
IppStatus ippiComplement_32s_C1R(Ipp32s* pSrcDst, int srcDstStep, IppiSize roiSize);
```

### Parameters

<i>pSrcDst</i>	Pointer to the source and destination image ROI.
<i>srcDstStep</i>	Distance in bytes between starts of consecutive lines in the source and destination image.
<i>roiSize</i>	Size of the source and destination ROI in pixels.

### Description

The function `ippiComplement` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts a negative integer number from the complement to direct code reserving the sign in the most significant bit.

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if the <i>pSrcDst</i> pointer is <code>NULL</code> .
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcDstStep</i> has a zero or negative value.
<code>ippStsStrideErr</code>	Indicates an error condition if <i>srcDstStep</i> is less than the image width.

## DotProd

*Computes the dot product of pixel values of two source images.*

---

### Syntax

#### Case 1: Operation on one-channel integer data

```
IppStatus ippiDotProd_<mod>(const Ipp<srcDatatype>* pSrc1, int src1Step, const Ipp<srcDatatype>* pSrc2, int src2Step, IppiSize roiSize, Ipp64f* pDp);
```

Supported values for `mod`:

8u64f_C1R	8s64f_C1R	16u64f_C1R	16s64f_C1R	32u64f_C1R	32s64f_C1R
-----------	-----------	------------	------------	------------	------------

**Case 2: Operation on three-channel integer data**

```
IppStatus ippiDotProd_<mod>(const Ipp<srcDatatype>* pSrc1, int src1Step, const
Ipp<srcDatatype>* pSrc2, int src2Step, IppiSize roiSize, Ipp64f* pDp[3]);
```

Supported values for mod:

8u64f\_C3R      8s64f\_C3R      16u64f\_C3R      16s64f\_C3R      32u64f\_C3R      32s64f\_C3R

**Case 3: Operation on four-channel integer data**

```
IppStatus ippiDotProd_<mod>(const Ipp<srcDatatype>* pSrc1, int src1Step, const
Ipp<srcDatatype>* pSrc2, int src2Step, IppiSize roiSize, Ipp64f* pDp[4]);
```

Supported values for mod:

8u64f\_C4R      8s64f\_C4R      16u64f\_C4R      16s64f\_C4R      32u64f\_C4R      32s64f\_C4R  
8u64f\_AC4R      8s64f\_AC4R      16u64f\_AC4R      16s64f\_AC4R      32u64f\_AC4R      32s64f\_AC4R

**Case 4: Operation on floating-point data**

```
IppStatus ippiDotProd_32f64f_C1R(const Ipp32f* pSrc1, int src1Step, const Ipp32f* pSrc2,
int src2Step, IppiSize roiSize, Ipp64f* pDp, IppHintAlgorithm hint);
```

```
IppStatus ippiDotProd_32f64f_C3R(const Ipp32f* pSrc1, int src1Step, const Ipp32f* pSrc2,
int src2Step, IppiSize roiSize, Ipp64f* pDp[3], IppHintAlgorithm hint);
```

```
IppStatus ippiDotProd_32f64f_C4R(const Ipp32f* pSrc1, int src1Step, const Ipp32f* pSrc2,
int src2Step, IppiSize roiSize, Ipp64f* pDp[4], IppHintAlgorithm hint);
```

```
IppStatus ippiDotProd_32f64f_AC4R(const Ipp32f* pSrc1, int src1Step, const Ipp32f* pSrc2,
int src2Step, IppiSize roiSize, Ipp64f* pDp[4], IppHintAlgorithm hint);
```

**Parameters**

<i>pSrc1, pSrc2</i>	Pointer to the ROI in the source images.
<i>src1Step, src2Step</i>	Distance in bytes between starts of consecutive lines in the source images.
<i>pDp</i>	Pointer to the dot product or to the array containing the computed dot products for multi-channel images.
<i>hint</i>	Option to select the algorithmic implementation of the function, see Table “ <a href="#">Hint Arguments for Image Moment Functions</a> ”.

**Description**

The function `ippiDotProd` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function computes the dot product of pixel values of two source images *pSrc1* and *pSrc2* using algorithm indicated by the hint argument (see Table “[Hint Arguments for Image Moment Functions](#)”) and stores the result in *pDp*. In case of multi-channel images, the dot product is computed for each channel separately and stored in the array *pDp*.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <code>roiSize</code> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if one of the step values has zero or negative value.

## DotProdCol

*Calculates the dot product of taps vector and columns of the specified set of rows.*

---

### Syntax

```
IppStatus ippIDotProdCol_32f_L2(const Ipp32f* const ppSrcRow[], const Ipp32f* pTaps, int tapsLen, Ipp32f* pDst, int width);
```

```
IppStatus ippIDotProdCol_64f_L2(const Ipp64f* const ppSrcRow[], const Ipp64f* pTaps, int tapsLen, Ipp64f* pDst, int width);
```

### Parameters

<code>ppSrcRow</code>	Pointer to the set of rows.
<code>pTaps</code>	Pointer to the taps vector.
<code>tapsLen</code>	Length of taps vector, is equal to the number of rows.
<code>pDst</code>	Pointer to the destination row.
<code>width</code>	Width of the source and destination rows.

### Description

The function `ippIDotProdCol` is declared in the `ippi.h` file.

This function calculates the dot product of taps vector `pTaps` and columns of the specified set of the rows `ppSrcRow`. It is useful for external vertical filtering pipeline implementation.

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <code>width</code> is less than or equal to 0.

## Logical Operations

Functions described in this section perform bitwise operations on pixel values. The operations include logical AND, NOT, inclusive OR, exclusive OR, and bit shifts.

## And

*Performs a bitwise AND operation between corresponding pixels of two images.*

---

### Syntax

#### Case 1: Not-in-place operation

```
IppStatus ippiAnd_<mod>(const Ipp<datatype>* pSrc1, int src1Step, const Ipp<datatype>* pSrc2, int src2Step, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);
```

Supported values for mod:

8u_C1R	16u_C1R	32s_C1R
8u_C3R	16u_C3R	32s_C3R
8u_C4R	16u_C4R	32s_C4R
8u_AC4R	16u_AC4R	32s_AC4R

#### Case 2: In-place operation

```
IppStatus ippiAnd_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize);
```

Supported values for mod:

8u_C1IR	6u_C1IR	32s_C1IR
8u_C3IR	16u_C3IR	32s_C3IR
8u_C4IR	16u_C4IR	32s_C4IR
8u_AC4IR	16u_AC4IR	32s_AC4IR

### Parameters

<i>pSrc, pSrc1, pSrc2</i>	Pointers to the source images ROI.
<i>srcStep, src1Step, src2Step</i>	Distances in bytes between starts of consecutive lines in the source images.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>pSrcDst</i>	Pointer to the source and destination image ROI for the in-place operation.
<i>srcDstStep</i>	Distance in bytes between starts of consecutive lines in the source and destination imager for the in-place operation.
<i>roiSize</i>	Size of the source and destination ROI in pixels.

## Description

The function `ippiAnd` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function performs a bitwise AND operation between the values of corresponding pixels of two source image ROIs, and writes the result into a destination image ROI.

Note that the functions with the `AC4` descriptor do not process alpha channels.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <code>roiSize</code> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if any of the specified buffer step values is zero or negative.

## AndC

*Performs a bitwise AND operation of each pixel with a constant.*

---

## Syntax

### Case 1: Not-in-place operation on one-channel data

```
IppStatus ippiAndC<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype> value,
Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);
```

Supported values for `mod`:

`8u_C1R`      `16u_C1R`      `32s_C1R`

### Case 2: Not-in-place operation on multi-channel data

```
IppStatus ippiAndC<mod>(const Ipp<datatype>* pSrc, int srcStep, const Ipp<datatype>
value[3], Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);
```

Supported values for `mod`:

`8u_C3R`      `16u_C3R`      `32s_C3R`  
`8u_AC4R`      `16u_AC4R`      `32s_AC4R`

```
IppStatus ippiAndC<mod>(const Ipp<datatype>* pSrc, int srcStep, const Ipp<datatype>
value[4], Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);
```

Supported values for `mod`:

`8u_C4R`      `16u_C4R`      `32s_C4R`

**Case 3: In-place operation on one-channel data**

```
IppStatus ippiAndC_<mod>(Ipp<datatype> value, const Ipp<datatype>* pSrcDst, int srcDstStep,
IppiSize roiSize);
```

Supported values for mod:

8u\_C1IR            16u\_C1IR            32s\_C1IR

**Case 4: In-place operation on multi-channel data**

```
IppStatus ippiAndC_<mod>(const Ipp<datatype> value[3], const Ipp<datatype>* pSrcDst, int
srcDstStep, IppiSize roiSize);
```

Supported values for mod:

8u\_C3IR            16u\_C3IR            32s\_C3IR  
8u\_AC4IR           16u\_AC4IR           32s\_AC4IR

```
IppStatus ippiAndC_<mod>(const Ipp<datatype> value[4], const Ipp<datatype>* pSrcDst, int
srcDstStep, IppiSize roiSize);
```

Supported values for mod:

8u\_AC4IR           16u\_AC4IR           32s\_AC4IR

**Parameters**

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>value</i>	The constant value to perform the bitwise AND operation on each pixel of the source image ROI (constant vector in case of multi-channel images).
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>pSrcDst</i>	Pointer to the source and destination image ROI for the in-place operation.
<i>srcDstStep</i>	Distance in bytes between starts of consecutive lines in the source and destination image buffer for the in-place operation.
<i>roiSize</i>	Size of the source and destination ROI in pixels.

**Description**

The function `ippiAndC` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function performs a bitwise AND operation between each pixel value of a source image ROI and constant *value*.



Note that the functions with the AC4 descriptor do not process alpha channels.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <code>roiSize</code> has a field with a zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <code>srcStep</code> , <code>dstStep</code> , or <code>srcDstStep</code> has a zero or negative value.

## Or

*Performs bitwise inclusive OR operation between pixels of two source buffers.*

## Syntax

### Case 1: Not-in-place operation

```
IppStatus ippiOr_<mod>(const Ipp<datatype>* pSrc1, int src1Step, const Ipp<datatype>* pSrc2, int src2Step, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);
```

Supported values for mod:

<code>8u_C1R</code>	<code>16u_C1R</code>	<code>32s_C1R</code>
<code>8u_C3R</code>	<code>16u_C3R</code>	<code>32s_C3R</code>
<code>8u_C4R</code>	<code>16u_C4R</code>	<code>32s_C4R</code>
<code>8u_AC4R</code>	<code>16u_AC4R</code>	<code>32s_AC4R</code>

### Case 2: In-place operation

```
IppStatus ippiOr_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize);
```

Supported values for mod:

<code>8u_C1IR</code>	<code>16u_C1IR</code>	<code>32s_C1IR</code>
<code>8u_C3IR</code>	<code>16u_C3IR</code>	<code>32s_C3IR</code>
<code>8u_C4IR</code>	<code>16u_C4IR</code>	<code>32s_C4IR</code>
<code>8u_AC4IR</code>	<code>16u_AC4IR</code>	<code>32s_AC4IR</code>

## Parameters

`pSrc`, `pSrc1`, `pSrc2`      Pointers to the source images ROI.

<i>srcStep, src1Step, src2Step</i>	Distances in bytes between starts of consecutive lines in the source images.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>pSrcDst</i>	Pointer to the source and destination image ROI for the in-place operation.
<i>srcDstStep</i>	Distance in bytes between starts of consecutive lines in the source and destination image for the in-place operation.
<i>roiSize</i>	Size of the source and destination ROI in pixels.

## Description

The function `ippiOr` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function performs a bitwise inclusive OR operation between the values of corresponding pixels of two source image ROIs, and writes the result into a destination image ROI. Note that the functions with the `AC4` descriptor do not process alpha channels.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if any of the specified buffer step values is zero or negative.

## Example

The code example below show how to use the function `ippiOr_8u_C1R`.

```
void func_or()
{
    IppiSize Src1ROI = {8,4};
    IppiSize Src2ROI = {8,4};
    IppiSize DstROI = {5,4};
    Ipp8u src1[8*4];
    Ipp8u src2[8*4];
    Ipp8u dst[8*4];

    ippiSet_8u_C1R(0,dest,8,Src1ROI);
    ippiSet_8u_C1R(5,src1,8,Src1ROI);
    ippiSet_8u_C1R(6,src2,8,Src2ROI);
    ippiOr_8u_C1R(src1,8,src2,8,dst,8,DstROI);
}
```

Result:

src1	src2	dst
05 05 05 05 05 05 05 05	06 06 06 06 06 06 06 06	07 07 07 07 07 00 00 00
05 05 05 05 05 05 05 05	06 06 06 06 06 06 06 06	07 07 07 07 07 00 00 00

```

05 05 05 05 05 05 05 05    06 06 06 06 06 06 06 06    07 07 07 07 07 00 00 00
05 05 05 05 05 05 05 05    06 06 06 06 06 06 06 06    07 07 07 07 07 00 00 00

```

## OrC

*Performs a bitwise inclusive OR operation between each pixel of a buffer and a constant.*

### Syntax

#### Case 1: Not-in-place operation on one-channel data

```

IppStatus ippiOrC_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype> value,
Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);

```

Supported values for mod:

```

8u_C1R    16u_C1R    32s_C1R

```

#### Case 2: Not-in-place operation on multi-channel data

```

IppStatus ippiOrC_<mod>(const Ipp<datatype>* pSrc, int srcStep, const Ipp<datatype>
value[3], Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);

```

Supported values for mod:

```

8u_C3R    16u_C3R    32s_C3R

8u_AC4R    16u_AC4R    32s_AC4R

```

```

IppStatus ippiOrC_<mod>(const Ipp<datatype>* pSrc, int srcStep, const Ipp<datatype>
value[4], Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);

```

Supported values for mod:

```

8u_C4R    16u_C4R    32s_C4R

```

#### Case 3: In-place operation on one-channel data

```

IppStatus ippiOrC_<mod>(Ipp<datatype> value, const Ipp<datatype>* pSrcDst, int srcDstStep,
IppiSize roiSize);

```

Supported values for mod:

```

8u_C1IR    16u_C1IR    32s_C1IR

```

**Case 4: In-place operation on multi-channel data**

```
IppStatus ippiOrC_<mod>(const Ipp<datatype> value[3], const Ipp<datatype>* pSrcDst, int
srcDstStep, IppiSize roiSize);
```

Supported values for mod:

8u\_C3IR    16u\_C3IR    32s\_C3IR

8u\_AC4IR   16u\_AC4IR   32s\_AC4IR

```
IppStatus ippiOrC_<mod>(const Ipp<datatype> value[4], const Ipp<datatype>* pSrcDst, int
srcDstStep, IppiSize roiSize);
```

Supported values for mod:

8u\_C4IR

16u\_C4IR

32s\_C4IR

**Parameters**

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>value</i>	The constant value to perform the bitwise OR operation on each pixel of the source image (constant vector in case of multi-channel images).
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>pSrcDst</i>	Pointer to the source and destination image ROI for the in-place operation.
<i>srcDstStep</i>	Distance in bytes between starts of consecutive lines in the source and destination image for the in-place operation.
<i>roiSize</i>	Size of the source and destination ROI in pixels.

**Description**

The function `ippiOrC` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function performs a bitwise inclusive OR operation between each pixel value of a source image ROI and constant *value*.

Note that the functions with the AC4 descriptor do not process alpha channels.

**Return Values**

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointers is <code>NULL</code> .

<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> , <i>dstStep</i> , or <i>srcDstStep</i> has a zero or negative value.

## Xor

*Performs bitwise exclusive OR operation between pixels of two source buffers.*

---

### Syntax

#### Case 1: Not-in-place operation

```
IppStatus ippixor_<mod>(const Ipp<datatype>* pSrc1, int src1Step, const Ipp<datatype>* pSrc2, int src2Step, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);
```

Supported values for mod:

8u_C1R	16u_C1R	32s_C1R
8u_C3R	16u_C3R	32s_C3R
8u_C4R	16u_C4R	32s_C4R
8u_AC4R	16u_AC4R	32s_AC4R

#### Case 2: In-place operation

```
IppStatus ippixor_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize);
```

Supported values for mod:

8u_C1IR	16u_C1IR	32s_C1IR
8u_C3IR	16u_C3IR	32s_C3IR
8u_C4IR	16u_C4IR	32s_C4IR
8u_AC4IR	16u_AC4IR	32s_AC4IR

### Parameters

<i>pSrc</i> , <i>pSrc1</i> , <i>pSrc2</i>	Pointers to the source images ROI.
<i>srcStep</i> , <i>src1Step</i> , <i>src2Step</i>	Distances in bytes between starts of consecutive lines in the source images.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.

<i>pSrcDst</i>	Pointer to the source and destination image ROI for the in-place operation.
<i>srcDstStep</i>	Distance in bytes between starts of consecutive lines in the source and destination image for the in-place operation.
<i>roiSize</i>	Size of the source and destination ROI in pixels.

## Description

The function `ippiXor` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function performs a bitwise exclusive OR operation between the values of corresponding pixels of two source image ROIs, and writes the result into a destination image ROI.

Note that the functions with the AC4 descriptor do not process alpha channels.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if any of the specified buffer step values is zero or negative.

## XorC

*Performs a bitwise exclusive OR operation between each pixel of a buffer and a constant.*

---

## Syntax

### Case 1: Not-in-place operation on one-channel data

```
ippStatus ippiXorC_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype> value,
Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);
```

Supported values for `mod`:

8u\_C1R    16u\_C1R    32s\_C1R

### Case 2: Not-in-place operation on multi-channel data

```
IppStatus ippiXorC_<mod>(const Ipp<datatype>* pSrc, int srcStep, const Ipp<datatype>
value[3], Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);
```

Supported values for `mod`:

8u\_C3R    16u\_C3R    32s\_C3R  
8u\_AC4R    16u\_AC4R    32s\_AC4R

```
IppStatus ippiXorC_<mod>(const Ipp<datatype>* pSrc, int srcStep, const Ipp<datatype>
value[4], Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);
```

Supported values for mod:

8u\_C4R    16u\_C4R    32s\_C4R

### Case 3: In-place operation on one-channel data

```
IppStatus ippiXorC_<mod>(Ipp<datatype> value, const Ipp<datatype>* pSrcDst, int srcDstStep,
IppiSize roiSize);
```

Supported values for mod:

8u\_C1IR    16u\_C1IR    32s\_C1IR

### Case 4: In-place operation on multi-channel data

```
IppStatus ippiXorC_<mod>(const Ipp<datatype> value[3], const Ipp<datatype>* pSrcDst, int
srcDstStep, IppiSize roiSize);
```

Supported values for mod:

8u\_C3IR    16u\_C3IR    32s\_C3IR

8u\_AC4IR    16u\_AC4IR    32s\_AC4IR

```
IppStatus ippiXorC_<mod>(const Ipp<datatype> value[4], const Ipp<datatype>* pSrcDst, int
srcDstStep, IppiSize roiSize);
```

Supported values for mod:

8u\_C4IR    16u\_C4IR    32s\_C4IR

## Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>value</i>	The constant value to perform the bitwise XOR operation on each pixel of the source image ROI (constant vector in case of multi-channel images).
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>pSrcDst</i>	Pointer to the source and destination image ROI for the in-place operation.
<i>srcDstStep</i>	Distance in bytes between starts of consecutive lines in the source and destination image for the in-place operation.

*roiSize*

Size of the source and destination ROI in pixels.

## Description

The function `ippiXorC` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function performs a bitwise exclusive OR operation between each pixel value of a source image ROI and constant *value*.

Note that the functions with the AC4 descriptor do not process alpha channels.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> , <i>dstStep</i> , or <i>srcDstStep</i> has a zero or negative value.

## Not

*Performs a bitwise NOT operation on each pixel of a source buffer.*

---

## Syntax

### Case 1: Not-in-place operation

```
IppStatus ippiNot_<mod>(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst, int dstStep, IppiSize roiSize);
```

Supported values for *mod*:

8u\_C1R  
 8u\_C3R  
 8u\_C4R  
 8u\_AC4R

### Case 2: In-place operation

```
IppStatus ippiNot_<mod>(Ipp8u* pSrcDst, int srcDstStep, IppiSize roiSize);
```

Supported values for *mod*:

8u\_C1IR  
 8u\_C3IR  
 8u\_C4IR



8u\_AC4IR

## Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>pSrcDst</i>	Pointer to the source and destination image ROI for the in-place operation.
<i>srcDstStep</i>	Distance in bytes between starts of consecutive lines in the source and destination image for the in-place operation.
<i>roiSize</i>	Size of the source and destination ROI in pixels.

## Description

The function `ippiNot` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function performs a bitwise NOT operation on each pixel value of a source image ROI.

Note that the functions with the AC4 descriptor do not process alpha channels.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> , <i>dstStep</i> , or <i>srcDstStep</i> has a zero or negative value.

## RShiftC

*Shifts bits in pixel values to the right.*

## Syntax

### Case 1: Not-in-place operation on one-channel data

```
ippStatus ippiRShiftC_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp32u value,
Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);
```

Supported values for `mod`:

8u\_C1R                      8s\_C1R                      16u\_C1R                      16s\_C1R                      32s\_C1R

**Case 2: Not-in-place operation on multi-channel data**

```
IppStatus ippiRShiftC_<mod>(const Ipp<datatype>* pSrc, int srcStep, const Ipp32u value[3],
Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);
```

Supported values for mod:

8u_C3R	8s_C3R	16u_C3R	16s_C3R	32s_C3R
8u_AC4R	8s_AC4R	16u_AC4R	16s_AC4R	32s_AC4R

```
IppStatus ippiRShiftC_<mod>(const Ipp<datatype>* pSrc, int srcStep, const Ipp32u value[4],
Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);
```

Supported values for mod:

8u_C4R	8s_C4R	16u_C4R	16s_C4R	32s_C4R
--------	--------	---------	---------	---------

**Case 3: In-place operation on one-channel data**

```
IppStatus ippiRShiftC_<mod>(Ipp32u value, Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize
roiSize);
```

Supported values for mod:

8u_C1IR	8s_C1IR	16u_C1IR	16s_C1IR	32s_C1IR
---------	---------	----------	----------	----------

**Case 4: In-place operation on multi-channel data**

```
IppStatus ippiRShiftC_<mod>(const Ipp32u value[3], Ipp<datatype>* pSrcDst, int srcDstStep,
IppiSize roiSize);
```

Supported values for mod:

8u_C3IR	8s_C3IR	16u_C3IR	16s_C3IR	32s_C3IR
8u_AC4IR	8s_AC4IR	16u_AC4IR	16s_AC4IR	32s_AC4IR

```
IppStatus ippiRShiftC_<mod>(const Ipp32u value[4], Ipp<datatype>* pSrcDst, int srcDstStep,
IppiSize roiSize);
```

Supported values for mod:

8u_C4IR	8s_C4IR	16u_C4IR	16s_C4IR	32s_C4IR
---------	---------	----------	----------	----------

**Parameters**

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.

<i>value</i>	The number of bits to shift (constant vector in case of multi-channel images).
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>pSrcDst</i>	Pointer to the source and destination image ROI for the in-place operation.
<i>srcDstStep</i>	Distance in bytes between starts of consecutive lines in the source and destination image for the in-place operation.
<i>roiSize</i>	Size of the source and destination ROI in pixels.

## Description

The function `ippiRShiftC` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function decreases the intensity of pixels in the source image ROI by shifting the bits in each pixel value by *value* bits to the right. The positions vacated after shifting the bits are filled with the sign bit. In case of multi-channel data, each color channel can have its own shift value. This operation is equivalent to dividing the pixel values by a constant power of 2.

Note that the functions with the `AC4` descriptor do not process alpha channels.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with a zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> , <i>dstStep</i> , or <i>srcDstStep</i> has a zero or negative value.

## LShiftC

*Shifts bits in pixel values to the left.*

## Syntax

### Case 1: Not-in-place operation on one-channel data

```
ippStatus ippiLShiftC_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp32u value,
Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);
```

Supported values for `mod`:

8u\_C1R      16u\_C1R      32s\_C1R

**Case 2: Not-in-place operation on multi-channel data**

```
IppStatus ippiLShiftC_<mod>(const Ipp<datatype>* pSrc, int srcStep, const Ipp32u value[3],
Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);
```

Supported values for mod:

8u_C3R	16u_C3R	32s_C3R
8u_AC4R	16u_AC4R	32s_AC4R

```
IppStatus ippiLShiftC_<mod>(const Ipp<datatype>* pSrc, int srcStep, const Ipp32u value[4],
Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);
```

Supported values for mod:

8u_C4R	16u_C4R	32s_C4R
--------	---------	---------

**Case 3: In-place operation on one-channel data**

```
IppStatus ippiLShiftC_<mod>(Ipp32u value, Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize
roiSize);
```

Supported values for mod:

8u_C1IR	16u_C1IR	32s_C1IR
---------	----------	----------

**Case 4: In-place operation on multi-channel data**

```
IppStatus ippiLShiftC_<mod>(const Ipp32u value[3], Ipp<datatype>* pSrcDst, int srcDstStep,
IppiSize roiSize);
```

Supported values for mod:

8u_C3IR	16u_C3IR	32s_C3IR
8u_AC4IR	16u_AC4IR	32s_AC4IR

```
IppStatus ippiLShiftC_<mod>(const Ipp32u value[4], Ipp<datatype>* pSrcDst, int srcDstStep,
IppiSize roiSize);
```

Supported values for mod:

8u_C4IR	16u_C4IR	32s_C4IR
---------	----------	----------

**Parameters**

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.

<i>value</i>	The number of bits to shift (constant vector in case of multi-channel images).
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>pSrcDst</i>	Pointer to the source and destination image ROI for the in-place operation.
<i>srcDstStep</i>	Distance in bytes between starts of consecutive lines in the source and destination image for the in-place operation.
<i>roiSize</i>	Size of the source and destination ROI in pixels.

## Description

The function `ippiLShiftC` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function changes the intensity of pixels in the source image ROI by shifting the bits in each pixel value by *value* bits to the left. In case of multi-channel data, each color channel can have its own shift value. The positions vacated after shifting the bits are filled with zeros. Values obtained as a result of left shift operations are not saturated. To get saturated values, use multiplication functions instead.

Note that the functions with the `AC4` descriptor do not process alpha channels.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> , <i>dstStep</i> , or <i>srcDstStep</i> has a zero or negative value.

## Example

The code example below illustrates the use of left shift function.

```
IppStatus lshift( void ) {
    Ipp8u img[8*8] = { 1, 0x7F, 0xFE };
    IppiSize roi = { 8, 8 };
    IppStatus st = ippiLShiftC_8u_C1IR( 1, img, 8, roi );
    printf( "%02x %02x %02x\n", img[0], img[1], img[2] );
    return st;
}
```

Output values:

```
02 fe fc
```

## Alpha Composition

The Intel IPP provides functions that composite two image buffers using either the opacity (alpha) channel in the images or provided alpha values.

These functions operate on image buffers with 8-bit or 16-bit data in RGB or RGBA format. In all compositing operations a resultant pixel in destination buffer *pDst* is created by overlaying a pixel from the foreground image buffer *pSrc1* over a pixel from the background image buffer *pSrc2*. The supported types of images combining by using alpha values are listed in the table below.

### Types of Image Composing Operations

Type	Output Pixel		Description in Imaging Terms
	Color Components	Alpha value	
OVER	$\alpha_A * A + (1 - \alpha_A) * \alpha_B * B$	$\alpha_A + (1 - \alpha_A) * \alpha_B$	<i>A</i> occludes <i>B</i>
IN	$\alpha_A * A * \alpha_B$	$\alpha_A * \alpha_B$	<i>A</i> within <i>B</i> . <i>A</i> acts as a matte for <i>B</i> . <i>A</i> shows only where <i>B</i> is visible.
OUT	$\alpha_A * A * (1 - \alpha_B)$	$\alpha_A * (1 - \alpha_B)$	<i>A</i> outside <i>B</i> . NOT- <i>B</i> acts as a matte for <i>A</i> . <i>A</i> shows only where <i>B</i> is not visible.
ATOP	$\alpha_A * A * \alpha_B + (1 - \alpha_A) * \alpha_B * B$	$\alpha_A * \alpha_B + (1 - \alpha_A) * \alpha_B$	Combination of ( <i>A</i> IN <i>B</i> ) and ( <i>B</i> OUT <i>A</i> ). <i>B</i> is both back-ground and matte for <i>A</i> .
XOR	$\alpha_A * A * (1 - \alpha_B) + (1 - \alpha_A) * \alpha_B * B$	$\alpha_A * (1 - \alpha_B) + (1 - \alpha_A) * \alpha_B$	Combination of ( <i>A</i> OUT <i>B</i> ) and ( <i>B</i> OUT <i>A</i> ). <i>A</i> and <i>B</i> mutually exclude each other.
PLUS	$\alpha_A * A + \alpha_B * B$	$\alpha_A + \alpha_B$	Blend without precedence

In the formulas above, the input image buffers are denoted as *A* and *B* for simplicity. The Greek letter  $\alpha$  with subscripts denotes the normalized (scaled) alpha value in the range 0 to 1. It is related to the integer alpha value *alpha* as:

$$\alpha = \text{alpha} / \text{max\_val}$$

where *max\_val* is 255 for 8-bit or 65535 for 16-bit unsigned pixel data.

For the `ippiAlphaComp` function that operates on 4-channel RGBA buffers only,  $\alpha_A$  and  $\alpha_B$  are the normalized alpha values of the two input image buffers, respectively.

For the `ippiAlphaCompC` function,  $\alpha_A$  and  $\alpha_B$  are the normalized constant alpha values that are passed as parameters to the function.

Note that in formulas for computing the resultant color channel values, *A* and *B* stand for the pixel color components of the respective input image buffers.

To save a significant amount of computation for some of the alpha compositing operations, use functions [AlphaPremul](#), [AlphaPremulC](#) for pre-multiplying color channel values by the alpha values. This reduces the number of multiplications required in the compositing operations, which is especially efficient for repeated compositing of an image.

The type of composition operation that is performed by the function [AlphaComp](#) and [AlphaCompC](#) is specified by the parameter *alphaType*, the table below lists its possible values.

#### Possible Values of alphaType Parameter

Operation types		Parameter Value
OVER	<code>ippAlphaOver</code>	<code>ippAlphaOverPremul</code>
IN	<code>ippAlphaIn</code>	<code>ippAlphaInPremul</code>
OUT	<code>ippAlphaOut</code>	<code>ippAlphaOutPremul</code>
ATOP	<code>ippAlphaATop</code>	<code>ippAlphaATopPremul</code>
XOR	<code>ippAlphaXor</code>	<code>ippAlphaXorPremul</code>
PLUS	<code>ippAlphaPlus</code>	<code>ippAlphaPlusPremul</code>

## AlphaComp

*Combines two images using alpha (opacity) values of both images.*

### Syntax

```
IppStatus ippAlphaComp_<mod>(const Ipp<datatype>* pSrc1, int src1Step, const Ipp<datatype>*
pSrc2, int src2Step, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, IppiAlphaType
alphaType);
```

Supported values for mod:

```
8s_AC1R 16s_AC1R 32s_AC1R 32f_AC1R
8s_AC4R 16s_AC4R 32s_AC4R 32f_AC4R
```

```
IppStatus ippAlphaComp_<mod>(const Ipp<datatype>* const pSrc1[4], int src1Step, const
Ipp<datatype>* const pSrc2[4], int src2Step, Ipp<datatype>* const pDst[4], int dstStep,
IppiSize roiSize, IppiAlphaType alphaType);
```

Supported values for mod:

```
8u_AP4R 16u_AP4R
```

## Parameters

<i>pSrc1, pSrc2</i>	Pointers to the source image ROI for pixel-order data. An array of pointers to ROI in the separate source color planes in case of planar data.
<i>src1Step, src2Step</i>	Distances in bytes between starts of consecutive lines in the source images.
<i>pDst</i>	Pointer to the destination image ROI for pixel-order data. An array of pointers to ROI in the separate destination color planes in case of planar data.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>roiSize</i>	Size of the source and destination ROI in pixels.
<i>alphaType</i>	The composition type to perform. See <a href="#">Table “Possible Values of the Parameter alphaType”</a> for the type value and description.

## Description

The function `ippiAlphaComp` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function performs an image compositing operation on RGBA images using alpha values of both images. The compositing is done by overlaying pixels  $(r_A, g_A, b_A, \leq_A)$  from the foreground image *pSrc1* with pixels  $(r_B, g_B, b_B, \leq_B)$  from the background image *pSrc2* to produce pixels  $(r_C, g_C, b_C, \leq_C)$  in the resultant image *pDst*. The alpha values are assumed to be normalized to the range [0..1].

The type of the compositing operation is indicated by the *alphaType* parameter. Use [Table “Possible Values of the Parameter alphaType”](#) to choose a valid *alphaType* value depending on the required composition type. For example, the resulting pixel color components for the OVER operation (see [Table “Types of Image Composing Operations”](#)) are computed as follows:

$$r_C = \leq_A * r_A + (1 - \leq_A) * \leq_B * r_B$$

$$g_C = \leq_A * g_A + (1 - \leq_A) * \leq_B * g_B$$

$$b_C = \leq_A * b_A + (1 - \leq_A) * \leq_B * b_B$$

The resulting (normalized) alpha value is computed as

$$\leq_C = \leq_A + (1 - \leq_A) * \leq_B$$

This function can be used for unsigned pixel data only.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.



## AlphaCompC

*Combines two images using constant alpha values.*

### Syntax

```
IppStatus ippiAlphaCompC_<mod>(const Ipp<datatype>* pSrc1, int src1Step, Ipp<datatype>
alpha1, const Ipp<datatype>* pSrc2, int src2Step, Ipp<datatype> alpha2, Ipp<datatype>*
pDst, int dstStep, IppiSize roiSize, IppiAlphaType alphaType);
```

Supported values for mod:

8u\_C1R 8u\_C3R 8u\_C4R 8u\_AC4R

16u\_C1R 16u\_C3R 16u\_C4R 16u\_AC4R

8s\_C1R

16s\_C1R

32u\_C1R

32s\_C1R

32f\_C1R

```
IppStatus ippiAlphaCompC_<mod>(const Ipp<datatype>* constpSrc1[4], int src1Step,
Ipp<datatype> alpha1, const Ipp<datatype>* const pSrc2[4], int src2Step, Ipp<datatype>
alpha2, Ipp<datatype>* const pDst[4], int dstStep, IppiSize roiSize, IppiAlphaType
alphaType);
```

Supported values for mod:

8u\_AP4R 16u\_AP4R

### Parameters

<i>pSrc1, pSrc2</i>	Pointers to the source image ROI for pixel-order data. An array of pointers to ROI in the separate source color planes in case of planar data.
<i>src1Step, src2Step</i>	Distances in bytes between starts of consecutive lines in the source images.
<i>pDst</i>	Pointer to the destination image ROI for pixel-order data. An array of pointers to ROI in the separate destination color planes in case of planar data.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>roiSize</i>	Size of the source and destination ROI in pixels.
<i>alpha1, alpha2</i>	Constant alpha values to use for the compositing operation.

*alphaType*

The composition type to perform. See [Table “Possible Values of the Parameter alphaType”](#) for the type value and description.

## Description

The function `ippiAlphaCompC` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function performs an image compositing operation on one-channel image buffers, three-channel RGB and four-channel RGBA image buffers and on planar images, using constant alpha values *alpha1* and *alpha2*. These values are passed to the function as parameters.

The compositing is done by overlaying pixels from the foreground image ROI *pSrc1* with pixels from the background image ROI *pSrc2* to produce pixels in the resultant image ROI *pDst*. The alpha values are normalized to the range [0..1].

The type of the compositing operation is indicated by the *alphaType* parameter. Use [Table “Possible Values of the Parameter alphaType”](#) to choose a valid *alphaType* value depending on the required composition type. For example, the resulting pixel color components for the OVER operation (see [Table “Types of Image Composing Operations”](#)) are computed as follows:

$$r_C = \leq_1 * r_A + (1 - \leq_1) * \leq_2 * r_B$$

$$g_C = \leq_1 * g_A + (1 - \leq_1) * \leq_2 * g_B$$

$$b_C = \leq_1 * b_A + (1 - \leq_1) * \leq_2 * b_B$$

where  $\leq_1, \leq_2$  are the normalised alpha values *alpha1*, *alpha2*.

This function can be used for unsigned pixel data only.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.

## Example

The code example below shows how to use alpha composition function.

```
IppStatus acomp( void ) {
    Ipp8u imga[8*8], imgb[8*8], imgc[8*8];
    IppiSize roi = { 8, 8 };
    IppStatus st;
    ippiImageRamp_8u_C1R( imga, 8, roi, 0, 1, ippAxsHorizontal );
    ippiImageRamp_8u_C1R( imgb, 8, roi, 0, 2, ippAxsHorizontal );
    st = ippiAlphaCompC_8u_C1R( imga, 8, 255/3, imgb, 8, 255, imgc, 8, roi, ippAlphaOver );
    printf( "over: a=%d,A=255/3; b=%d,B=255; c=%d\n",
        c=a*A+b*(1-A)*B, imga[6], imgb[6], imgc[6] );
    return st;
}
```

**Output**

over: a=6,A=255/3; b=12,B=255; c=10 // c=a\*A+b\*B\*(1-A)

## AlphaPremul

*Pre-multiplies pixel values of an image by its alpha values.*

---

### Syntax

**Case 1: Not-in-place operation**

```
IppStatus ippiAlphaPremul_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);
```

Supported values for mod:

8u\_AC4R     16u\_AC4R

```
IppStatus ippiAlphaPremul_<mod>(const Ipp<datatype>* const pSrc[4], int srcStep, Ipp<datatype>* const pDst[4], int dstStep, IppiSize roiSize);
```

Supported values for mod:

8u\_AP4R     16u\_AP4R

**Case 2: In-place operation**

```
IppStatus ippiAlphaPremul_<mod>(Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize);
```

Supported values for mod:

8u\_AC4IR     16u\_AC4IR

```
IppStatus ippiAlphaPremul_<mod>(Ipp<datatype>* const pSrcDst[4], int srcDstStep, IppiSize roiSize);
```

Supported values for mod:

8u\_AP4IR     16u\_AP4IR

### Parameters

<i>pSrc</i>	Pointer to the source image ROI for pixel-order data. An array of pointers to ROI in the separate source color planes in case of planar data.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.

<i>pDst</i>	Pointer to the destination image ROI for pixel-order data. An array of pointers to ROI in the separate destination color planes in case of planar data.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>pSrcDst</i>	Pointer to the source and destination buffer or an array of pointers to separate source and destination color planes for the in-place operation.
<i>srcDstStep</i>	Distance in bytes between starts of consecutive lines in the source and destination image for the in-place operation.
<i>roiSize</i>	Size of the source and destination ROI in pixels.

## Description

The function `ippiAlphaPremul` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts a RGBA source image (pixel order or planar) to the pre-multiplied alpha form. If (r,g,b,a) are the red, green, blue, and alpha values of a pixel, then new pixel values are (r\* ≤, g\*≤, b\*≤, a) after execution of this function. Here ≤ is the pixel normalized alpha value in the range 0 to 1.

The function `ippiAlphaPremul` can be used for unsigned pixel data only.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.

## AlphaPremulC

*Pre-multiplies pixel values of an image using constant alpha (opacity) values.*

---

## Syntax

### Case 1: Not-in-place operation

```
ippStatus ippiAlphaPremulC_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>
alpha, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);
```

Supported values for mod :

8u_C1R	16u_C1R
8u_C3R	16u_C3R
8u_C4R	16u_C4R

8u\_AC4R     16u\_AC4R

```
IppStatus ippiAlphaPremulC_<mod>(const Ipp<datatype>* const pSrc[4], int srcStep,
Ipp<datatype> alpha, Ipp<datatype>* const pDst[4], int dstStep, IppiSize roiSize);
```

Supported values for mod :

8u\_AP4R     16u\_AP4R

### Case 2: In-place operation

```
IppStatus ippiAlphaPremulC_<mod>(Ipp<datatype> alpha, Ipp<datatype>* pSrcDst, int
srcDstStep, IppiSize roiSize);
```

Supported values for mod :

8u\_C1IR     16u\_C1IR

8u\_C3IR     16u\_C3IR

8u\_C4IR     16u\_C4IR

8u\_AC4IR     16u\_AC4IR

```
IppStatus ippiAlphaPremulC_<mod>(Ipp<datatype> alpha, Ipp<datatype>* const pSrcDst[4],
int srcDstStep, IppiSize roiSize);
```

Supported values for mod :

8u\_AP4IR     16u\_AP4IR

### Parameters

<i>pSrc</i>	Pointer to the source image ROI for pixel-order data. An array of pointers to ROI in the separate source color planes in case of planar data.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI for pixel-order data. An array of pointers to separate ROI in the destination color planes in case of planar data.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>pSrcDst</i>	Pointer to the source and destination image ROI or an array of pointers to ROI in the separate source and destination color planes for the in-place operation.
<i>srcDstStep</i>	Distance in bytes between starts of consecutive lines in the source and destination image for the in-place operation.

<code>roiSize</code>	Size of the source and destination ROI in pixels.
<code>alpha</code>	Global alpha value used for pre-multiplying pixel values.

### Description

The function `ippiAlphaPremulC` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts either a one-, three-, four-channel image or planar RGBA image to the pre-multiplied alpha form, using the global alpha value `alpha`. For one-, three-, four-channel image buffers, pixel values in each channel are multiplied by  $\leq$ ; for RGBA (pixel order and planar) images with (r,g,b,a) pixel values, new pixel values are (r\* $\leq$ , g\* $\leq$ , b\* $\leq$ , `alpha`) after execution of this function. Here  $\leq$  is the normalized `alpha` value in the range 0 to 1.

The function `ippiAlphaPremulC` can be used for unsigned pixel data only.

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <code>roiSize</code> has a field with zero or negative value.

# Image Color Conversion

This chapter describes the Intel® IPP image processing functions that perform different types of image color conversion. The Intel IPP software supports the following image color conversions:

- Color models conversion
- Conversion from color to gray scale and vice versa
- Different types of format conversion:
  - from pixel-order to planar format and vice versa
  - changing number of channels or planes
  - changing sampling format
  - altering order of samples or planes
- Gamma correction
- Reduction from high bit resolution color to low bit resolution color
- Intensity transformation using lookup tables
- Color twist
- Color keying

All Intel IPP color conversion functions perform point operations on pixels of the source image. For a given destination pixel, the resultant channel values are computed using channel values of the corresponding source pixel only, and not involving any neighborhood pixels. Thus, the rectangular region of interest (ROI, see [Regions of Interest in Intel IPP](#)) used in function operations may extend to the size of the whole image.

The table below lists the Intel IPP color space conversion functions.

## Color Conversion Functions

Function Base Name	Description
<b>Color Model Conversion</b>	
<a href="#">RGBToYUV</a> , <a href="#">YUVToRGB</a>	Convert RGB image to and from YUV color model.
<a href="#">RGBToYUV422</a> , <a href="#">YUV422ToRGB</a>	Convert RGB image to and from 4:2:2 YUV image.
<a href="#">RGB565ToYUV422</a>	Converts a 16-bit RGB image to the 4:2:2 YUV image
<a href="#">RGBToYUV420</a> , <a href="#">YUV420ToRGB</a>	Convert RGB images to and from 4:0:0 YUV image.
<a href="#">BGRToYUV420</a>	Converts a BGR image to the 4:2:0 YUV image.
<a href="#">YUV420ToBGR</a>	Converts a 4:2:0 YUV image to BGR image.
<a href="#">RGB565ToYUV420</a>	Converts a 16-bit RGB image to the 4:2:0 YUV image

Function Base Name	Description
<a href="#">YUV420ToRGB565</a> , <a href="#">YUV420ToRGB555</a> , <a href="#">YUV420ToRGB444</a>	Convert a 4:2:0 YUV image to the 16-bit RGB image.
<a href="#">YUV420ToRGB565Dither</a> , <a href="#">YUV420ToRGB555Dither</a> , <a href="#">YUV420ToRGB444Dither</a>	Convert a 4:2:0 YUV image to the 16-bit RGB image with dithering.
<a href="#">BGR565ToYUV420</a> , <a href="#">BGR555ToYUV420</a>	Convert a 16-bit BGR image to the 4:2:0 YUV image.
<a href="#">YUV420ToBGR565</a> , <a href="#">YUV420ToBGR555</a> , <a href="#">YUV420ToBGR444</a>	Convert a 4:2:0 YUV image to the 16-bit BGR image.
<a href="#">YUV420ToBGR565Dither</a> , <a href="#">YUV420ToBGR555Dither</a> , <a href="#">YUV420ToBGR444Dither</a>	Convert a 4:2:0 YUV image to the 16-bit BGR image with dithering.
<a href="#">RGBToYCbCr</a>	Converts RGB images to the YCbCr color model.
<a href="#">YCbCrToRGB</a>	Converts RGB images from the YCbCr color model.
<a href="#">YCbCrToBGR</a>	Converts a YCbCr image to the BGR color model.
<a href="#">YCbCrToBGR_709CSC</a>	Converts a YCbCr image to the BGR image for ITU-R BT.709 signal.
<a href="#">YCbCrToRGB565</a> , <a href="#">YCbCrToRGB555</a> , <a href="#">YCbCrToRGB444</a>	Convert a YCbCr image to the 16-bit per pixel RGB image.
<a href="#">YCbCrToRGB565Dither</a> , <a href="#">YCbCrToRGB555Dither</a> , <a href="#">YCbCrToRGB444Dither</a>	Convert a YCbCr image to the 16-bit per pixel RGB image with dithering.
<a href="#">YCbCrToBGR565</a> , <a href="#">YCbCrToBGR555</a> , <a href="#">YCbCrToBGR444</a>	Convert a YCbCr image to the 16-bit per pixel BGR image.
<a href="#">YCbCrToBGR565Dither</a> , <a href="#">YCbCrToBGR555Dither</a> , <a href="#">YCbCrToBGR444Dither</a>	Convert a YCbCr image to the 16-bit per pixel BGR image with dithering.
<a href="#">RGBToYCbCr422</a>	Converts an RGB image to a 4:2:2 YCbCr image.
<a href="#">YCbCr422ToRGB</a>	Converts an RGB image to from a 4:2:2 YCbCr image.
<a href="#">RGBToYCrCb422</a>	Converts an RGB image to a 4:2:2 YCrCb image.
<a href="#">YCrCb422ToRGB</a>	Converts an RGB image from a 4:2:2 YCrCb image.
<a href="#">BGRToYCbCr422</a>	Converts a 24-bit per pixel BGR image to a 16-bit per pixel YCbCr image.
<a href="#">YCbCr422ToBGR</a>	Converts a 16-bit per pixel YCbCr image to a 24-bit per pixel BGR image.
<a href="#">BGR565ToYCbCr422</a> , <a href="#">BGR555ToYCbCr422</a>	Convert a 16-bit per pixel BGR image to a 16-bit per pixel YCbCr image.



Function Base Name	Description
<a href="#">RGBToCbYCr422</a> , <a href="#">RGBToCbYCr422Gamma</a>	Convert RGB images to and from a 4:2:2 CbYCr image.
<a href="#">CbYCr422ToRGB</a>	Converts 16-bit per pixel CbYCr image to 24-bit per pixel RGB image.
<a href="#">BGRToCbYCr422</a>	Converts a 32-bit per pixel BGR image to a 16-bit per pixel CbYCr image.
<a href="#">BGRToCbYCr422_709HDTV</a>	Converts a BGR image to a 16-bit per pixel CbYCr image for an ITU-R BT.709 HDTV signal.
<a href="#">CbYCr422ToBGR</a>	Converts a 16-bit per pixel CbYCr image to a four channel BGR image.
<a href="#">CbYCr422ToBGR_709HDTV</a>	Converts a 16-bit per pixel CbYCr image to a BGR image for an ITU-R BT.709 HDTV signal.
<a href="#">YCbCr422ToRGB565</a> , <a href="#">YCbCr422ToRGB555</a> , <a href="#">YCbCr422ToRGB444</a>	Convert a 16-bit per pixel YCbCr image that has 4:2:2 sampling format to a 16-bit per pixel RGB image.
<a href="#">YCbCr422ToRGB565Dither</a> , <a href="#">YCbCr422ToRGB555Dither</a> , <a href="#">YCbCr422ToRGB444Dither</a>	Convert a 16-bit per pixel YCbCr image that has 4:2:2 sampling format to a 16-bit per pixel RGB image with dithering.
<a href="#">YCbCr422ToBGR565</a> , <a href="#">YCbCr422ToBGR555</a> , <a href="#">YCbCr422ToBGR444</a>	Convert a 16-bit per pixel YCbCr image that has 4:2:2 sampling format to a 16-bit per pixel BGR image.
<a href="#">YCbCr422ToBGR565Dither</a> , <a href="#">YCbCr422ToBGR555Dither</a> , <a href="#">YCbCr422ToBGR444Dither</a>	Convert a 16-bit per pixel YCbCr image that has 4:2:2 sampling format to a 16-bit per pixel BGR image with dithering.
<a href="#">RGBToYCbCr420</a>	Converts RGB images to a YCbCr color model with 4:2:0 sampling format.
<a href="#">YCbCr420ToRGB</a>	Converts RGB images from a YCbCr color model with 4:2:0 sampling format.
<a href="#">YCbCr420ToRGB565</a> , <a href="#">YCbCr420ToRGB555</a> , <a href="#">YCbCr420ToRGB444</a>	Convert a YCbCr image that has 4:2:0 sampling format to a 16-bit per pixel RGB image.
<a href="#">YCbCr420ToRGB565Dither</a> , <a href="#">YCbCr420ToRGB555Dither</a> , <a href="#">YCbCr420ToRGB444Dither</a>	Convert a YCbCr image that has 4:2:0 sampling format to a 16-bit per pixel RGB image with dithering.
<a href="#">RGBToYCrCb420</a>	Converts an RGB image to a YCrCb image with the 4:2:0 sampling.
<a href="#">YCrCb420ToRGB</a>	Converts a YCrCb image with the 4:2:0 sampling to an RGB image.
<a href="#">BGRToYCbCr420</a>	Converts a BGR image to a YCbCr image with 4:2:0 sampling format.

Function Base Name	Description
BGRToYCbCr420_709CSC	Converts a BGR image to a YCbCr image with 4:2:0 sampling for an ITU-R BT.709 CSC signal.
BGRToYCbCr420_709HDTV	Converts a BGR image to a YCbCr image with 4:2:0 sampling for an ITU-R BT.709 HDTV signal.
BGRToYCrCb420_709CSC	Converts a BGR image to a YCrCb image with 4:2:0 sampling for an ITU-R BT.709 signal.
YCbCr420ToBGR	Converts a YCbCr image with 4:2:0 sampling format to the RGB color model.
YCbCr420ToBGR_709CSC	Converts a YCbCr image with 4:2:0 sampling to a BGR image for an ITU-R BT.709 CSC signal.
YCbCr420ToBGR_709HDTV	Converts a YCbCr image with 4:2:0 sampling to a BGR image for an ITU-R BT.709 HDTV signal.
BGR565ToYCbCr420, BGR555ToYCbCr420	Convert a 16-bit per pixel BGR image to a YCbCr image that has 4:2:0 sampling format.
YCbCr420ToBGR565, YCbCr420ToBGR555, YCbCr420ToBGR444	Convert a YCbCr image with 4:2:0 sampling format to a 16-bit per pixel BGR image.
YCbCr420ToBGR565Dither, YCbCr420ToBGR555Dither, YCbCr420ToBGR444Dither	Convert a YCbCr image with 4:2:0 sampling format to a 16-bit per pixel BGR image with dithering.
BGRToYCrCb420	Converts a BGR image to a YCrCb image with 4:2:0 sampling format.
BGR565ToYCrCb420, BGR555ToYCrCb420	Convert a 16-bit per pixel BGR image to a YCrCb image with the 4:2:0 sampling format.
BGRToYCbCr411	Converts a BGR image to a YCbCr planar image that has the 4:1:1 sampling format.
YCbCr411ToBGR	Converts a YCbCr image with the 4:1:1 sampling format to the RGB color model.
BGR565ToYCbCr411, BGR555ToYCbCr411	Convert a 16-bit per pixel BGR image to a YCbCr image that has the 4:1:1 sampling format.
YCbCr411ToBGR565, YCbCr411ToBGR555	Convert a YCbCr image that has the 4:1:1 sampling format to a 16-bit per pixel BGR image.
RGBToXYZ, XYZToRGB	Convert RGB images to and from the XYZ color model
RGBToLUV, LUVToRGB	Convert RGB images to and from the CIE LUV color model.
BGRToLab, LabToBGR	Convert BGR images to and from the CIE Lab color model.
RGBToYCC, YCCToRGB	Convert RGB images to and from the YCC color model.

Function Base Name	Description
<a href="#">RGBToHLS, HLSToRGB</a>	Convert RGB images to and from the HLS color model.
<a href="#">BGRToHLS</a>	Converts BGR images to the HLS color model.
<a href="#">HLSToBGR</a>	Converts HLS images to the BGR color model.
<a href="#">RGBToHSV, HSVToRGB</a>	Convert RGB images to and from the HSV color model.
<a href="#">RGBToYCoCg, YCoCgToRGB</a>	Convert RGB images to and from the YCoCg color model.
<a href="#">BGRToYCoCg</a>	Converts a 24-bit BGR image to the YCoCg color model.
<a href="#">SBGRToYCoCg</a>	Converts a 48-bit BGR image to the YCoCg color model.
<a href="#">YCoCgToBGR</a>	Converts a YCoCg image to the 24-bit BGR image.
<a href="#">YCoCgToSBGR</a>	Converts a YCoCg image to the 48-bit BGR image.
<a href="#">BGRToYCoCg_Rev</a>	Converts a 24-bit BGR image to the YCoCg-R color model.
<a href="#">SBGRToYCoCg_Rev</a>	Converts a 48-bit BGR image to the YCoCg-R color model.
<a href="#">YCoCgToBGR_Rev</a>	Converts a YCoCg-R image to the 24-bit BGR image.
<a href="#">YCoCgToSBGR_Rev</a>	Converts a YCoCg-R image to the 48-bit BGR image.
<b>Color - Gray Scale Conversion</b>	
<a href="#">RGBToGray</a>	Converts an RGB image to gray scale using fixed transform coefficients.
<a href="#">ColorToGray</a>	Converts an RGB image to gray scale using custom transform coefficients.
<a href="#">CFAToRGB</a>	Restores an RGB image from a gray-scale CFA image
<a href="#">DemosaicAHD</a>	Restores an RGB image from a gray-scale CFA image using the AHD algorithm.
<b>Format Conversion</b>	
<a href="#">RGBToRGB565, BGRToBGR565</a>	Convert a 24-bit per pixel image to a 16-bit image.
<a href="#">RGB565ToRGB, BGR565ToBGR</a>	Convert a 16-bit per pixel image to a 24-bit image.
<a href="#">YCbCr422</a>	Converts a 4:2:2 YCbCr image.
<a href="#">YCbCr422ToYCrCb422</a>	Converts a 4:2:2 YCbCr image to a 4:2:2 YCrCb image.
<a href="#">YCbCr422ToCbYCr422</a>	Converts a 4:2:2 YCbCr image to a 4:2:2 CbYCr image.
<a href="#">YCbCr422ToYCbCr420</a>	Converts a 4:2:2 YCbCr image to a 4:2:0 YCbCr image.

Function Base Name	Description
<a href="#">YCbCr422To420_Interlace</a>	Converts an interlaced YCbCr image from the 4:2:2 sampling format to the 4:2:0 format.
<a href="#">YCbCr422ToYCrCb420</a>	Converts a 4:2:2 YCbCr image to a 4:2:0 YCrCb image.
<a href="#">YCbCr422ToYCbCr411</a>	Converts a 4:2:2 YCbCr image to a 4:1:1 YCbCr image.
<a href="#">YCrCb422ToYCrCb422</a>	Converts a 4:2:2 YCrCb image to a 4:2:2 YCbCr image.
<a href="#">YCrCb422ToYCbCr420</a>	Converts a 4:2:2 YCrCb image to a 4:2:0 YCbCr image.
<a href="#">YCrCb422ToYCbCr411</a>	Converts a 4:2:2 YCrCb image to a 4:1:1 YCbCr image.
<a href="#">CbYCr422ToYCbCr422</a>	Converts a 4:2:2 CbYCr image to a 4:2:2 YCbCr image.
<a href="#">CbYCr422ToYCbCr420</a>	Converts a 4:2:2 CbYCr image to a 4:2:0 YCbCr image.
<a href="#">CbYCr422ToYCbCr420_Interlace</a>	Converts an interlaced 4:2:2 CbYCr image to a 4:2:0 YCbCr image.
<a href="#">CbYCr422ToYCrCb420</a>	Converts a 4:2:2 CbYCr image to a 4:2:0 YCrCb image.
<a href="#">CbYCr422ToYCbCr411</a>	Converts a 4:2:2 CbYCr image to a 4:1:1 YCbCr image.
<a href="#">YCbCr420</a>	Converts a 4:2:0 YCbCr image.
<a href="#">YCbCr420ToYCbCr422</a>	Converts a 4:2:0 YCbCr image to a 4:2:2 YCbCr image.
<a href="#">YCbCr420ToYCbCr422_Filter</a>	Converts a 4:2:0 YCbCr image to a 4:2:2 YCbCr image with additional filtering.
<a href="#">YCbCr420To422_Interlace</a>	Converts an interlaced 4:2:0 YCbCr image to a 4:2:2 YCbCr image.
<a href="#">YCbCr420ToCbYCr422</a>	Converts a 4:2:0 YCbCr image to a 4:2:2 CbYCr image.
<a href="#">YCbCr420ToCbYCr422_Interlace</a>	Converts an interlaced 4:2:0 YCbCr image to a 4:2:2 CbYCr image.
<a href="#">YCbCr420ToYCrCb420</a>	Converts a 4:2:0 YCbCr image to a 4:2:0 YCrCb image.
<a href="#">YCbCr420ToYCrCb420_Filter</a>	Converts a 4:2:0 YCbCr image to a 4:2:0 YCrCb image with deinterlace filtering.
<a href="#">YCbCr420ToYCbCr411</a>	Converts a 4:2:0 YCbCr image to a 4:1:1 YCbCr image.
<a href="#">YCrCb420ToYCbCr422</a>	Converts a 4:2:0 YCrCb image to a 4:2:2 YCbCr image.
<a href="#">YCrCb420ToYCbCr422_Filter</a>	Converts a 4:2:0 YCrCb image to a 4:2:2 YCbCr image with additional filtering.
<a href="#">YCrCb420ToCbYCr422</a>	Converts a 4:2:0 YCrCb image to a 4:2:2 CbYCr image.

Function Base Name	Description
<a href="#">YCrCb420ToYCbCr420</a>	Converts a 4:2:0 YCrCb image to a 4:2:0 YCbCr image.
<a href="#">YCrCb420ToYCbCr411</a>	Converts a 4:2:0 YCrCb image to a 4:1:1 YCbCr image.
<a href="#">YCbCr411</a>	Converts a 4:1:1 YCbCr image.
<a href="#">YCbCr411ToYCbCr422</a>	Converts a 4:1:1 YCbCr image to a 4:2:2 YCbCr image.
<a href="#">YCbCr411ToYCrCb422</a>	Converts a 4:1:1 YCbCr image to a 4:2:2 YCrCb image.
<a href="#">YCbCr411ToYCbCr420</a>	Converts a 4:1:1 YCbCr image to a 4:2:0 YCbCr image.
<a href="#">YCbCr411ToYCrCb420</a>	Converts a 4:1:1 YCbCr image to a 4:2:0 YCrCb image.
<b>Color Twist</b>	
<a href="#">ColorTwist</a>	Applies a color-twist matrix to an image with integer pixel values.
<a href="#">ColorTwist32f</a>	Applies a color-twist matrix to an image with floating-point pixel values.
<b>Color Keying</b>	
<a href="#">CompColorKey</a>	Performs color keying of two images.
<a href="#">AlphaCompColorKey</a>	Performs color keying and alpha composition of two images.
<b>Gamma Correction</b>	
<a href="#">GammaFwd</a>	Performs gamma-correction of the source RGB image.
<a href="#">GammaInv</a>	Converts a gamma-corrected R'G'B' image back to the original RGB image.
<b>Intensity Transformation</b>	
<a href="#">ReduceBits</a>	Reduces the bit resolution of an image.
<a href="#">LUT</a>	Maps an image by applying intensity transformation.
<a href="#">LUT_Linear</a>	Maps an image by applying intensity transformation with linear interpolation.
<a href="#">LUT_Cubic</a>	Maps an image by applying intensity transformation with cubic interpolation.
<a href="#">LUTPalette</a> , <a href="#">LUTPaletteSwap</a>	Map an image by applying intensity transformation in accordance with a palette table.
<a href="#">ToneMapLinear</a> , <a href="#">ToneMapMean</a>	Map an HDRI image to an LDRI image.

This chapter starts with introductory material that discusses color space models essential for understanding of the Intel IPP color conversion functions.

For more information about color spaces and color conversion techniques, see [Jack01], [Rogers85], and [Foley90].

## Gamma Correction

The luminance intensity generated by most displays is not a linear function of the applied signal but is proportional to some power (referred to as *gamma*) of the signal voltage. As a result, high intensity ranges are expanded and low intensity ranges are compressed. This nonlinearity must be compensated to achieve correct color reproduction. To do this, luminance of each of the linear red, green, and blue components is reduced to a nonlinear form using an inverse transformation. This process is called "gamma correction".

The Intel IPP functions use the following basic equations to convert an RGB image to a gamma-corrected R'G'B' image:

for  $R, G, B < 0.018$

$$R' = 4.5R$$

$$G' = 4.5G$$

$$B' = 4.5B$$

for  $R, G, B \geq 0.018$

$$R' = 1.099R^{0.45} - 0.099$$

$$G' = 1.099G^{0.45} - 0.099$$

$$B' = 1.099B^{0.45} - 0.099$$

Note that the channel intensity values are normalized to fit in the range [0..1]. The gamma value is equal to  $1/0.45 = 2.22$  in conformity with ITU Rec.709 specification (see [ITU709]).

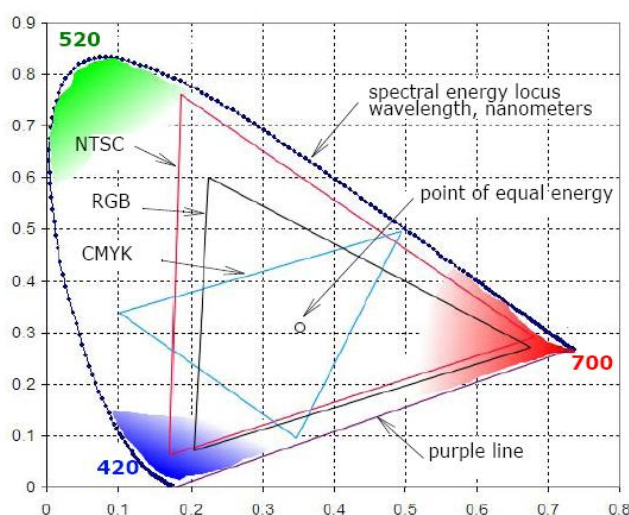
## CIE Chromaticity Diagram and Color Gamut

Figure CIE  $xyY$  Chromaticity Diagram and Color Gamut presents a diagram of all visible colors. It is called a chromaticity diagram and was developed as a result of the experimental investigations performed by CIE (International Commission on Illumination, <http://members.eunet.at/cie>). The diagram presents visible colors as a function of  $x$  (red) and  $y$  (green) components called *chromaticity coordinates*. Positions of various spectrum colors (from violet to red) are indicated as the points of a tongue-shaped curve called *spectrum locus*. The straight line connecting the ends of the curve is called the *purple line*. The point of equal energy represents the CIE standard for white light. Any point within the diagram represents some mixture of spectrum colors. The pure or fully saturated colors lie on the spectrum locus. A straight-line segment joining any two points in the diagram defines all color variations that can be obtained by additively combining these two colors. A triangle with vertices at any three points determine the gamut of colors that can be obtained by combining corresponding three colors.

The structure of the human eye that distinguishes three different stimuli, establishes the three-dimensional nature of color. The color may be described with a set of three parameters called tristimulus values, or components. These values may, for example, be dominant wavelength, purity, and luminance, or so-called primary colors: red, green, and blue.

The chromaticity diagram exhibits that the gamut of any three fixed colors cannot enclose all visible colors. For example, Figure CIE  $xyY$  Chromaticity Diagram and Color Gamut shows schematically the gamut of reproducible colors for the RGB primaries of a typical color CRT monitor, CMYK color printing, and for the NTSC television.

### CIE $xyY$ Chromaticity Diagram and Color Gamut



## Color Models

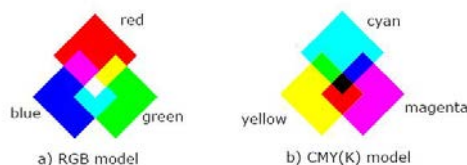
The purpose of a color model is to facilitate the specification of colors in some standard generally accepted way. In essence, a color model is a specification of a 3-D coordinate system and a subspace within that system where each color is represented by a single point.

Each industry that uses color employs the most suitable color model. For example, the RGB color model is used in computer graphics, YUV or YCbCr are used in video systems, PhotoYCC\* is used in PhotoCD\* production and so on. Transferring color information from one industry to another requires transformation from one set of values to another. Intel IPP provides a wide number of functions to convert different color spaces to RGB and vice versa.

## RGB Color Model

In the RGB model, each color appears as a combination of red, green, and blue. This model is called additive, and the colors are called primary colors. The primary colors can be added to produce the secondary colors of light (see Figure "Primary and Secondary Colors for RGB and CMYK Models") - magenta (red plus blue), cyan (green plus blue), and yellow (red plus green). The combination of red, green, and blue at full intensities makes white.

## Primary and Secondary Colors for RGB and CMYK Models



The color subspace of interest is a cube shown in Figure "RGB and CMY Color Models" (RGB values are normalized to 0..1), in which RGB values are at three corners; cyan, magenta, and yellow are the three other corners, black is at their origin; and white is at the corner farthest from the origin.

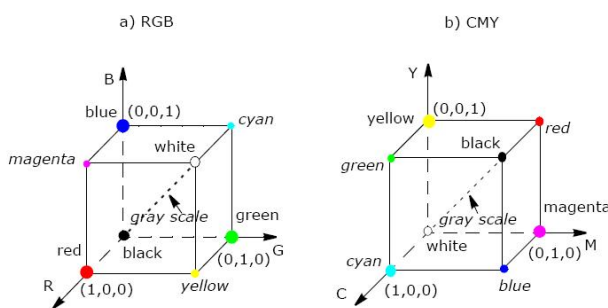
The gray scale extends from black to white along the diagonal joining these two points. The colors are the points on or inside the cube, defined by vectors extending from the origin.

Thus, images in the RGB color model consist of three independent image planes, one for each primary color.

As a rule, the Intel IPP color conversion functions operate with non-linear [gamma-corrected](#) images  $R'G'B'$ .

The importance of the RGB color model is that it relates very closely to the way that the human eye perceives color. RGB is a basic color model for computer graphics because color displays use red, green, and blue to create the desired color. Therefore, the choice of the RGB color space simplifies the architecture and design of the system. Besides, a system that is designed using the RGB color space can take advantage of a large number of existing software routines, because this color space has been around for a number of years.

## RGB and CMY Color Models



However, RGB is not very efficient when dealing with real-world images. To generate any color within the RGB color cube, all three RGB components need to be of equal pixel depth and display resolution. Also, any modification of the image requires modification of all three planes.



## CMYK Color Model

The CMYK color model is a subset of the RGB model and is primarily used in color print production. CMYK is an acronym for cyan, magenta, and yellow along with black (noted as K). The CMYK color space is subtractive, meaning that cyan, magenta yellow, and black pigments or inks are applied to a white surface to subtract some color from white surface to create the final color. For example (see [Figure "Primary and Secondary Colors for RGB and CMYK Models"](#)), cyan is white minus red, magenta is white minus green, and yellow is white minus blue. Subtracting all colors by combining the CMY at full saturation should, in theory, render black. However, impurities in the existing CMY inks make full and equal saturation impossible, and some RGB light does filter through, rendering a muddy brown color. Therefore, the black ink is added to CMY. The CMY cube is shown in [Figure "RGB and CMY Color Models"](#), in which CMY values are at three corners; red, green, and blue are the three other corners, white is at the origin; and black is at the corner farthest from the origin.

## YUV Color Model

The YUV color model is the basic color model used in analogue color TV broadcasting. Initially YUV is the re-coding of RGB for transmission efficiency (minimizing bandwidth) and for downward compatibility with black-and white television. The YUV color space is "derived" from the RGB space. It comprises the *luminance* (Y) and two color difference (U, V) components. The luminance can be computed as a weighted sum of red, green and blue components; the color difference, or *chrominance*, components are formed by subtracting luminance from blue and from red.

The principal advantage of the YUV model in image processing is decoupling of luminance and color information. The importance of this decoupling is that the luminance component of an image can be processed without affecting its color component. For example, the histogram equalization of the color image in the YUV format may be performed simply by applying histogram equalization to its Y component.

There are many combinations of YUV values from nominal ranges that result in invalid RGB values, because the possible RGB colors occupy only part of the YUV space limited by these ranges. [Figure "RGB Colors Cube in the YUV Color Space"](#) shows the valid color block in the YUV space that corresponds to the RGB color cube RGB values that are normalized to [0..1]).

The  $Y'U'V'$  notation means that the components are derived from gamma-corrected  $R'G'B'$ . Weighted sum of these non-linear components forms a signal representative of luminance that is called *luma*  $Y'$ . (*Luma* is often loosely referred to as *luminance*, so you need to be careful to determine whether a particular author assigns a linear or non-linear interpretation to the term *luminance*).

The Intel IPP functions use the following basic equation ([\[Jack01\]](#)) to convert between gamma-corrected  $R'G'B'$  and  $Y'U'V'$  models:

$$Y' = 0.299 * R' + 0.587 * G' + 0.114 * B'$$

$$U' = -0.147 * R' - 0.289 * G' + 0.436 * B' = 0.492 * (B' - Y')$$

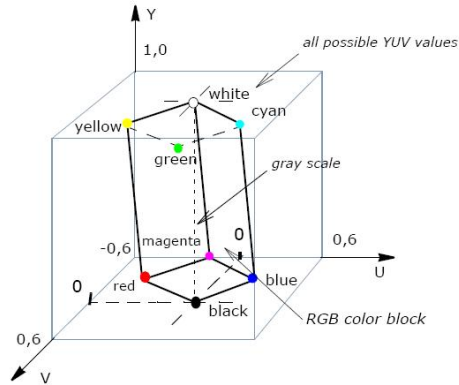
$$V' = 0.615 * R' - 0.515 * G' - 0.100 * B' = 0.877 * (R' - Y')$$

$$R' = Y' + 1.140 * V'$$

$$G' = Y' - 0.394 * U' - 0.581 * V'$$

$$B' = Y' + 2.032 * U'$$

### RGB Colors Cube in the YUV Color Space



There are several YUV sampling formats such as 4:4:4, 4:2:2, and 4:2:0 that are supported by the Intel IPP color conversion functions and are described later in this chapter in [Image Downsampling](#).

### YCbCr and YCCK Color Models

The YCbCr color space is used for component digital video and was developed as part of the ITU-R BT.601 Recommendation. YCbCr is a scaled and offset version of the YUV color space.

The Intel IPP functions use the following basic equations [Jack01] to convert between  $R'G'B'$  in the range 0-255 and  $Y'Cb'Cr'$  (this notation means that all components are derived from gamma-corrected  $R'G'B'$ ):

$$Y' = 0.257 * R' + 0.504 * G' + 0.098 * B' + 16$$

$$Cb' = -0.148 * R' - 0.291 * G' + 0.439 * B' + 128$$

$$Cr' = 0.439 * R' - 0.368 * G' - 0.071 * B' + 128$$

$$R' = 1.164 * (Y' - 16) + 1.596 * (Cr' - 128)$$

$$G' = 1.164 * (Y' - 16) - 0.813 * (Cr' - 128) - 0.392 * (Cb' - 128)$$

$$B' = 1.164 * (Y' - 16) + 2.017 * (Cb' - 128)$$

The Intel IPP color conversion functions specific for the JPEG codec use different equations:

$$Y = 0.299 * R + 0.587 * G + 0.114 * B$$

$$Cb = -0.16874 * R - 0.33126 * G + 0.5 * B + 128$$

$$Cr = 0.5 * R - 0.41869 * G - 0.08131 * B + 128$$

$$R = Y + 1.402 * Cr - 179.456$$

$$G = Y - 0.34414 * Cb - 0.71414 * Cr + 135.45984$$

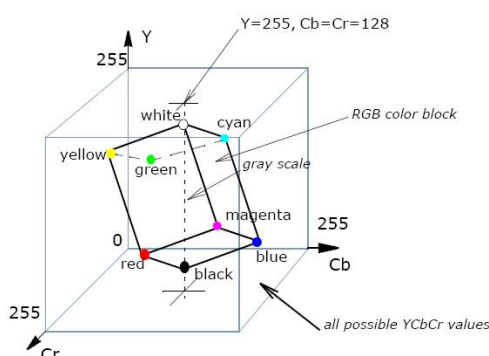
$$B = Y + 1.772 * Cb - 226.816$$

YCK model is specific for the JPEG image compression. It is a variant of the YCbCr model containing an additional K channel (black). The fact is that JPEG codec performs more effectively if the luminance and color information are decoupled. Therefore, a CMYK image must be converted to YCK before JPEG compression (see description of the function [ippiCMYKToYCK\\_JPEG](#) in Chapter 15 for more details).

Possible RGB colors occupy only part of the YCbCr color space (see [Figure "RGB Colors Cube in the YCbCr Space"](#)) limited by the nominal ranges, therefore there are many YCbCr combinations that result in invalid RGB values.

There are several YCbCr sampling formats such as 4:4:4, 4:2:2, 4:1:1, and 4:2:0, which are supported by the Intel IPP color conversion functions and are described later in this chapter in [Image Downsampling](#).

### RGB Colors Cube in the YCbCr Space



### PhotoYCC Color Model

The Kodak\* PhotoYCC\* was developed for encoding Photo CD\* image data. It is based on both the ITU Recommendations 601 and 709, using luminance-chrominance representation of color like in BT.601 YCbCr and BT.709 ([ITU709](#)). This model comprises luminance (Y) and two color difference, or chrominance (C1, C2) components. The PhotoYCC is optimized for the color photographic material, and provides a color gamut that is greater than the one that can currently be displayed.

The Intel IPP functions use the following basic equations [\[Jack01\]](#) to convert non-linear gamma-corrected  $R'G'B'$  to  $Y'C'C'$ :

$$Y' = 0.213 \cdot R' + 0.419 \cdot G' + 0.081 \cdot B'$$

$$C1' = -0.131 \cdot R' - 0.256 \cdot G' + 0.387 \cdot B' + 0.612$$

$$C2' = 0.373 \cdot R' - 0.312 \cdot G' - 0.061 \cdot B' + 0.537$$

The equations above are given on the assumption that  $R'$ ,  $G'$ , and  $B'$  values are normalized to the range [0..1].

Since the PhotoYCC model attempts to preserve the dynamic range of film, decoding PhotoYCC images requires selection of a color space and range appropriate for the output device. Thus, the decoding equations are not always the exact inverse of the encoding equations. The following equations [\[Jack01\]](#) are used in Intel IPP to generate  $R'G'B'$  values for driving a CRT display and require a unity relationship between the luma in the encoded image and the displayed image:

$$R' = 0.981 \cdot Y + 1.315 \cdot (C2' - 0.537)$$

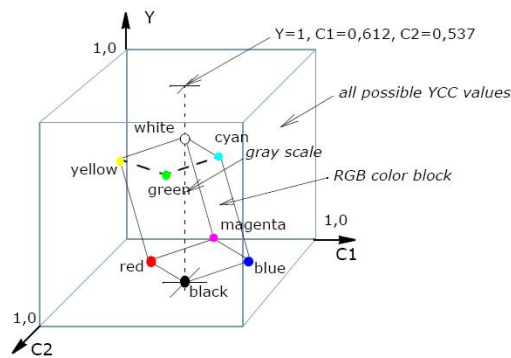
$$G' = 0.981 * Y - 0.311 * (C1 - 0.612) - 0.669 * (C2 - 0.537)$$

$$B' = 0.981 * Y + 1.601 * (C1 - 0.612)$$

The equations above are given on the assumption that source  $Y$ ,  $C1$  and  $C2$  values are normalized to the range  $[0..1]$ , and the display primaries have the chromaticity values in accordance with [ITU709] specifications.

The possible RGB colors occupy only part of the YCC color space (see Figure "RGB Colors in the YCC Color Space") limited by the nominal ranges, therefore there are many YCC combinations that result in invalid RGB values.

### RGB Colors in the YCC Color Space



### YCoCg Color Models

The YCoCg color model was developed to increase the effectiveness of the image compression [Malvar03]. This color model comprises the luminance ( $Y$ ) and two color difference components ( $C_o$  - offset orange,  $C_g$  - offset green).

The Intel IPP functions use the following simple basic equations [Malvar03] to convert between RGB and YCoCg:

$$Y = R/4 + G/2 + B/4$$

$$C_o = R/2 - B/2$$

$$C_g = -R/4 + G/2 - B/4$$

$$R = Y + C_o - C_g$$

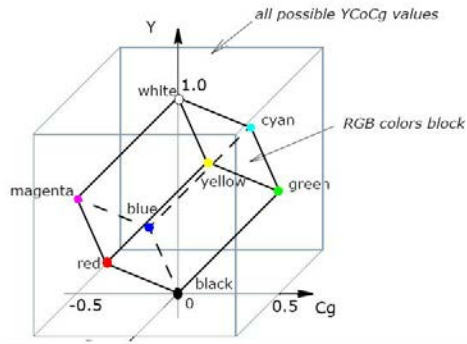
$$G = Y + C_g$$

$$B = Y - C_o - C_g$$

A variation of this color space which is called YCoCg-R, enables transformation reversibility with a smaller dynamic range requirements than does YCoCg [Malvar03-1].

The possible RGB colors occupy only part of the YCoCg color space (see Figure "RGB Color Cube in the YCoCg Color Space") limited by the nominal ranges, therefore there are many YCoCg combinations that result in invalid RGB values.

### RGB Color Cube in the YCoCg Color Space



### HSV, and HLS Color Models

The HLS (hue, lightness, saturation) and HSV (hue, saturation, value) color models were developed to be more "intuitive" in manipulating with color and were designed to approximate the way humans perceive and interpret color.

*Hue* defines the color itself. The values for the hue axis vary from 0 to 360 beginning and ending with red and running through green, blue and all intermediary colors.

*Saturation* indicates the degree to which the hue differs from a neutral gray. The values run from 0, which means no color saturation, to 1, which is the fullest saturation of a given hue at a given illumination.

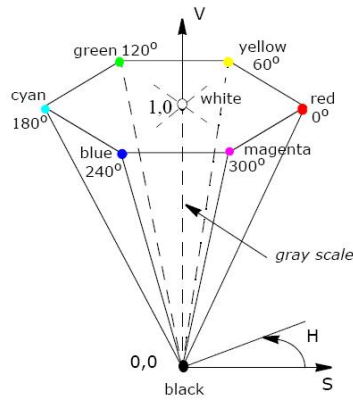
Intensity component - *lightness* (HLS) or *value* (HSV), indicates the illumination level. Both vary from 0 (black, no light) to 1 (white, full illumination). The difference between the two is that maximum saturation of hue ( $S=1$ ) is at *value*  $V=1$  (full illumination) in the HSV color model, and at *lightness*  $L=0.5$  in the HLS color model.

The HSV color space is essentially a cylinder, but usually it is represented as a cone or hexagonal cone (hexcone) as shown in the [Figure "HSV Solid"](#), because the hexcone defines the subset of the HSV space with valid RGB values. The *value*  $V$  is the vertical axis, and the vertex  $V=0$  corresponds to black color. Similarly, a color solid, or 3D-representation, of the HLS model is a double hexcone ([Figure "HSV Solid"](#)) with *lightness* as the axis, and the vertex of the second hexcone corresponding to white.

Both color models have intensity component decoupled from the color information. The HSV color space yields a greater dynamic range of saturation. Conversions from [RGBToHSV/RGBToHSV](#) and vice-versa in Intel IPP are performed in accordance with the respective pseudocode algorithms [\[Rogers85\]](#) given in the descriptions of corresponding conversion functions.

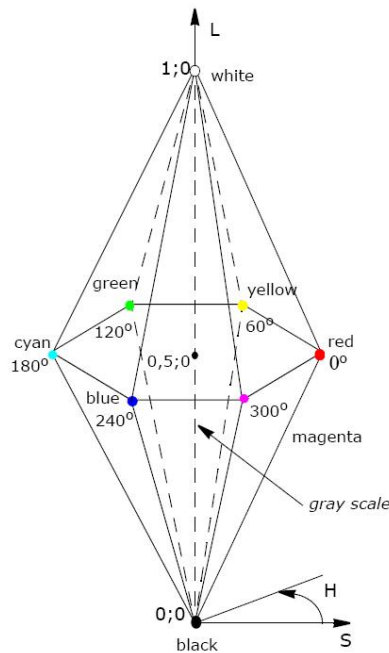
### HSV Solid

---



### HLS Solid

---

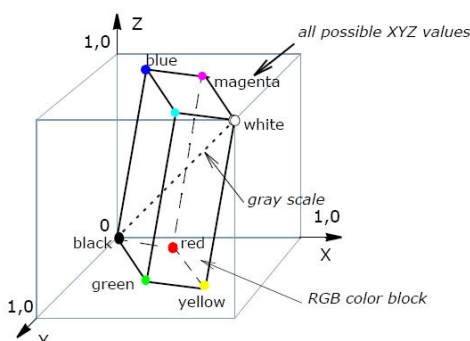


## CIE XYZ Color Model

The XYZ color space is an international standard developed by the CIE (Commission Internationale de l'Eclairage). This model is based on three hypothetical primaries, XYZ, and all visible colors can be represented by using only positive values of X, Y, and Z. The CIE XYZ primaries are hypothetical because they do not correspond to any real light wavelengths. The Y primary is intentionally defined to match closely to luminance, while X and Z primaries give color information. The main advantage of the CIE XYZ space (and any color space based on it) is that this space is completely device-independent. The chromaticity diagram in [Figure "CIE  \$x\$  \$y\$  \$z\$  Chromaticity Diagram and Color Gamut"](#) is in fact a two-dimensional projection of the CIE XYZ sub-space. Note that arbitrarily combining X, Y, and Z values within nominal ranges can easily lead to a "color" outside of the visible color spectrum.

The position of the block of RGB-representable colors in the XYZ space is shown in Figure "RGB Colors Cube in the XYZ Color Space".

## RGB Colors Cube in the XYZ Color Space



Intel IPP functions use the following basic equations [\[Rogers85\]](#), to convert between gamma-corrected  $R'G'B'$  and CIE XYZ models:

$$X = 0.412453 \cdot R' + 0.35758 \cdot G' + 0.180423 \cdot B'$$

$$Y = 0.212671 \cdot R' + 0.71516 \cdot G' + 0.072169 \cdot B'$$

$$Z = 0.019334 \cdot R' + 0.119193 \cdot G' + 0.950227 \cdot B'$$

The equations for  $X, Y, Z$  calculation are given on the assumption that  $R', G',$  and  $B'$  values are normalized to the range  $[0..1]$ .

$$R' = 3.240479 \cdot X - 1.53715 \cdot Y - 0.498535 \cdot Z$$

$$G' = -0.969256 \cdot X + 1.875991 \cdot Y + 0.041556 \cdot Z$$

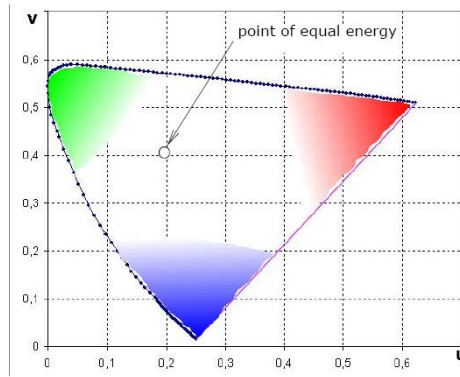
$$B' = 0.055648 \cdot X - 0.204043 \cdot Y + 1.057311 \cdot Z$$

The equations for  $R', G',$  and  $B'$  calculation are given on the assumption that  $x, y,$  and  $z$  values are in the range  $[0..1]$ .

## CIE LUV and CIE Lab Color Models

The CIE LUV and CIE Lab color models are considered to be perceptually uniform and are referred to as uniform color models. Both are uniform derivations from the standard CIE XYZ space. "Perceptually uniform" means that two colors that are equally distant in the color space are equally distant perceptually. To accomplish this approach, a uniform chromaticity scale (UCS) diagram was proposed by CIE (Figure "CIE  $u', v'$  Uniform Chromaticity Scale Diagram"). The UCS diagram uses a mathematical formula to transform the XYZ values or  $x, y$  coordinates (Figure "CIE  $xyY$  Chromaticity Diagram and Color Gamut"), to a new set of values that present a visually more accurate two-dimensional model. The  $Y$  lightness scale is replaced with a new scale called  $L$  that is approximately uniformly spaced but is more indicative of the actual visual differences. Chrominance components are  $U$  and  $V$  for CIE LUV, and  $a$  and  $b$  (referred to also respectively as red/blue and yellow/blue chrominances) in CIE Lab. Both color spaces are derived from the CIE XYZ color space.

### CIE $u', v'$ Uniform Chromaticity Scale Diagram



The CIE LUV color space is derived from CIE XYZ as follows ([Rogers85]),

$$L = 116. \cdot (Y/Y_n)^{1/3} - 16.$$

$$U = 13. \cdot L \cdot (u - u_n)$$

$$V = 13. \cdot L \cdot (v - v_n)$$

where

$$u = 4. \cdot X / (X + 15. \cdot Y + 3. \cdot Z)$$

$$v = 9. \cdot Y / (X + 15. \cdot Y + 3. \cdot Z)$$

$$u_n = 4. \cdot x_n / (-2. \cdot x_n + 12. \cdot y_n + 3. \cdot z_n)$$

$$v_n = 9. \cdot y_n / (-2. \cdot x_n + 12. \cdot y_n + 3. \cdot z_n)$$

Inverse conversion is performed in accordance with equations:

$$Y = Y_n \cdot ((L + 16.) / 116.)^3.$$

$$X = -9. \cdot Y \cdot u / ((u - 4.) \cdot v - u \cdot v_n)$$

$$Z = (9. \cdot Y - 15 \cdot v \cdot Y - v \cdot X) / 3. \cdot v$$

where



$$u = U / (13. * L) + u_n$$

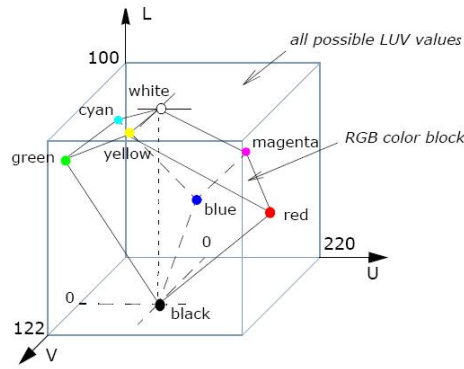
$$v = V / (13. * L) + v_n$$

and  $u_n, v_n$  are defined above.

Here  $x_n = 0.312713$ ,  $y_n = 0.329016$  are the CIE chromaticity coordinates of the D65 white point ([ITU709]), and  $y_n = 1.0$  is the luminance of the D65 white point. The values of the L component are in the range [0..100], U component in the range [-134..220], and V component in the range [-140..122].

The RGB-representable colors occupy only part of the LUV color space (see Figure 6-12) limited by the nominal ranges, therefore there are many LUV combinations that result in invalid RGB values.

### RGB Color Cube in the CIE LUV Color Space



The CIE Lab color space is derived from CIE XYZ as follows:

$$L = 116. * (Y/Y_n)^{1/3} - 16 \text{ for } Y/Y_n > 0.008856$$

$$L = 903.3 * (Y/Y_n)^{1/3} \text{ for } Y/Y_n \leq 0.008856$$

$$a = 500. * [f(X/X_n) - f(Y/Y_n)]$$

$$b = 200. * [f(Y/Y_n) - f(Z/Z_n)]$$

where

$$f(t) = t^{1/3} - 16 \text{ for } t > 0.008856$$

$$f(t) = 7.787*t + 16/116 \text{ for } t \leq 0.008856$$

Here  $y_n = 1.0$  is the luminance, and  $x_n = 0.950455$ ,  $z_n = 1.088753$  are the chrominances for the D65 white point.

The values of the L component are in the range [0..100],  $a$  and  $b$  component values are in the range [-128..127].

Inverse conversion is performed in accordance with equations:

$$Y = Y_n * P^3$$

$$X = X_n * (P + a/500.)^3$$

$$Z = Z_n * (P - b/200.)^3$$

where

$$P = (L + 16) / 116.$$

## Image Downsampling

Conventionally, digital color images are represented by setting specific values of the color space coordinates for each pixel. Color spaces with decoupled luminance and chrominance coordinates (YUV type) allow the number of bits required for acceptable color description of an image to be reduced. This reduction is based on greater sensitivity of the human eye to changes in luminance than to changes in chrominance. The idea behind this approach is to set individual value of luminance component to each pixel, while assigning the same color (chrominance components) to certain groups of pixels (sometimes called *macropixels*) in accordance with some specific rules. This process is called *downsampling* and there are different sampling formats depending on the underlying scheme.

The following sampling formats are supported by the Intel IPP image processing functions (excluding the JPEG functions):

**4:4:4 YUV (YCbCr)** - conventional format, no downsampling, Y, U(Cb), V(Cr) components are sampled at every pixel. If each component takes 8 bits, then every pixel requires 24 bits. This format is often denoted as YUV (YCbCr) with the 4:4:4 descriptor omitted.

**4:2:2 YUV (YCbCr)** - uses the 2:1 horizontal downsampling. It means that the Y component is sampled at each pixel, while U(Cb) and V(Cr) components are sampled every 2 pixels in horizontal direction. If each component takes 8 bits, the pair of pixels requires 32 bits.

**4:1:1 YCbCr** - uses the 4:1 horizontal downsampling. It means that the Y component is sampled at each pixel, while Cb and Cr components are sampled every 4 pixels horizontally. If each component takes 8 bits, each four horizontal pixels require 48 bits.

**4:2:0 YUV (YCbCr)** - uses the 2:1 horizontal downsampling and the 2:1 vertical downsampling. Y is sampled at each pixel, U(Cb) and V(Cr) are sampled at every block of 2x2 pixels. If each component takes 8 bits, each four-pixel block requires 48 bits.

In JPEG compression, downsampling has specific distinctive features and is denoted in a slightly different way. In JPEG domain, sampling formats determine the structure of minimal coded units, or MCUs. Therefore, the Intel IPP functions specific for a JPEG codec, support the following sampling formats:

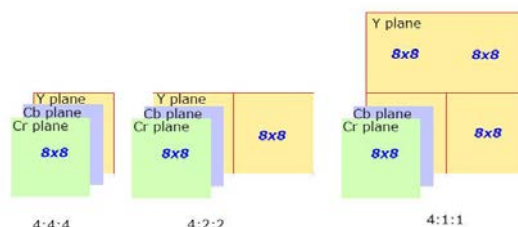
**4:4:4 YCbCr** - for every 8x8 block of Y samples, there is one 8x8 block of each Cb and Cr samples.

**4:2:2 YCbCr** - for every two horizontal 8x8 blocks of Y samples, there is one 8x8 block of each Cb and Cr samples.

**4:1:1 YCbCr** - for every four (two in horizontal and two in vertical direction) 8x8 blocks of Y samples, there is one 8x8 block of each Cb and Cr samples.

Structure of the corresponding MCU for each of these sampling formats is shown in Figure MCU Structure for Different JPEG Sampling Formats.

### MCU Structure for Different JPEG Sampling Formats



## RGB Image Formats

In addition to the 24-bit-per-pixel RGB/BGR image formats, the Intel IPP color conversion functions support 32-bit-per-pixel RGB/BGR formats, which include three RGB channels plus alpha channel. For 24-bit formats, each color is one byte, every pixel is three bytes. For 32-bit formats, each color is one byte and alpha component is one byte, which yields four bytes per pixel. Memory layout for these formats is given in [Table "Pixel-Order Image Formats"](#).

For 16-bit formats, every pixel is two bytes and each color occupies a specified number of bits. The figure below shows all the supported 16-bit-per-pixel formats and their memory layout (bit order):

### 16-bit pixel formats

16-bit pixel formats	
	High-order byte      Low-order byte
RGB565	B4 B3 B2 B1 B0 G5 G4 G3      G2 G1 G0 R4 R3 R2 R1 R0
RGB555	X B4 B3 B2 B1 B0 G4 G3      G2 G1 G0 R4 R3 R2 R1 R0
RGB444	X X X X B3 B2 B1 B0      G3 G2 G1 G0 R3 R2 R1 R0
BGR565	R4 R3 R2 R1 R0 G5 G4 G3      G2 G1 G0 B4 B3 B2 B1 B0
BGR555	X R4 R3 R2 R1 R0 G4 G3      G2 G1 G0 B4 B3 B2 B1 B0
BGR444	X X X X R3 R2 R1 R0      G3 G2 G1 G0 B3 B2 B1 B0

R - Red, G - Green, B - Blue, X - free bit

## Pixel and Planar Image Formats

Data storage for an image can be pixel-oriented or planar-oriented (planar). For images in pixel order, all channel values for each pixel are clustered and stored consecutively. Their layout depends on the color model and downsampling scheme.

Table “Pixel-Order Image Formats” lists all pixel-order image formats that are supported by the Intel IPP color conversion functions and shows the corresponding channel values order (here, group of underlined symbols represents one pixel and symbol A denotes alpha-channel value). The last column of this table gives an example of an Intel IPP color conversion function that uses the respective image format.

#### Pixel-Order Image Formats

Image Format	Number of Channels	Channel Values Order	Example
RGB	3	<u>R0 G0 B0</u> <u>R1 G1 B1</u> <u>R2 G2 B2</u>	ippiRGBToYUV_8u_C3R
RGB444			ippiYCbCrToRGB444_8u16u_C3R
RGB555			ippiYCbCrToRGB555_8u16u_C3R
RGB565			ippiYCbCrToRGB565_8u16u_C3R
RGB	4	<u>R0 G0 B0 A0</u> <u>R1 G1 B1 A1</u>	ippiRGBToYUV_8u_AC4R
BGR	3	<u>B0 G0 R0</u> <u>B1 G1 R1</u> <u>B2 G2 R2</u>	ippiYCbCrToBGR_8u_P3C3R
BGR444			ippiYCbCrToBGR444_8u16u_C3R
BGR555			ippiYCbCrToBGR555_8u16u_C3R
BGR565			ippiYCbCrToBGR565_8u16u_C3R
BGR	4	<u>B0 G0 R0 A0</u> <u>B1 G1 R1 A1</u>	ippiBGRToHLS_8u_A C4R
YUV	3	<u>Y0 U0 V0</u> <u>Y1 U1 V1</u> <u>Y2 U2 V2</u>	ippiYUVToRGB_8u_C3R
YUV	4	<u>Y0 U0 V0 A0</u> <u>Y1 U1 V1 A1</u>	ippiYUVToRGB_8u_A C4R
4:2:2 YUV	2	<u>Y0 U0</u> <u>Y1 V0</u> <u>Y2 U1</u> <u>Y3 V1</u>	ippiYUV422ToRGB_8u_C2C3R
YCbCr	3	<u>Y0 Cb0 Cr0</u> <u>Y1 Cb1 Cr1</u>	ippiYCbCrToRGB_8u_C3R
YCbCr	4	<u>Y0 Cb0 Cr0 A0</u> <u>Y1 Cb1 Cr1 A1</u>	ippiYCbCrToRGB_8u_A C4R
4:2:2 YCbCr	2	<u>Y0 Cb0</u> <u>Y1 Cr0</u> <u>Y2 Cb1</u> <u>Y3 Cr1</u>	ippiYCbCr422ToRGB_8u_C2C3R
4:2:2 YCrCb	2	<u>Y0 Cr0</u> <u>Y1 Cb0</u> <u>Y2 Cr1</u> <u>Y3 Cb1</u>	ippiYCrCb422ToYCbCr422_8u_C2P3R
4:2:2 CbYCr	2	<u>Cb0 Y0</u> <u>Cr0 Y1</u> <u>Cb1 Y2</u> <u>Cr1 Y3</u>	ippiCbYCr422ToRGB_8u_C2C3R
XYZ	3	<u>X0 Y0 Z0</u> <u>X1 Y1 Z1</u> <u>X2 Y2 Z2</u>	ippiXYZToRGB_8u_C3R
XYZ	4	<u>X0 Y0 Z0</u> <u>X1 Y1 Z1</u> <u>X2 Y2 Z2</u>	ippiXYZToRGB_16u_A C4R
LUV	3	<u>L0 U0 V0</u> <u>L1 U1 V1</u> <u>L2 U2 V2</u>	ippiLUVToRGB_16s_C3R
LUV	4	<u>L0 U0 V0 A0</u> <u>L1 U1 V1 A1</u>	ippiLUVToRGB_32f_A C4R
YCC	3	<u>Y0 C0 C0</u> <u>Y1 C1 C1</u> <u>Y2 C2 C2</u>	ippiYCCToRGB_8u_C3R
YCC	4	<u>Y0 C0 C0 A0</u> <u>Y1 C1 C1 A1</u>	ippiYCCToRGB_8u_A C4R
HLS	3	<u>H0 L0 S0</u> <u>H1 L1 S1</u> <u>H2 L2 S2</u>	ippiHLSToRGB_16u_C3R

Image Format	Number of Channels	Channel Values Order	Example
HLS	4	<u>H0 L0 S0 A0</u> <u>H1 L1 S1 A0</u>	ippi <b>HLS</b> ToRGB_16u_ <b>AC4R</b>
HSV	3	<u>H0 S0 V0</u> <u>H1 S1 V1</u> <u>H2 S2 V2</u>	ippi <b>HSV</b> ToRGB_16s_ <b>C3R</b>
HSV	4	<u>H0 S0 V0 A0</u> <u>H1 S1 C1 A1</u>	ippi <b>HSV</b> ToRGB_16s_ <b>AC4R</b>

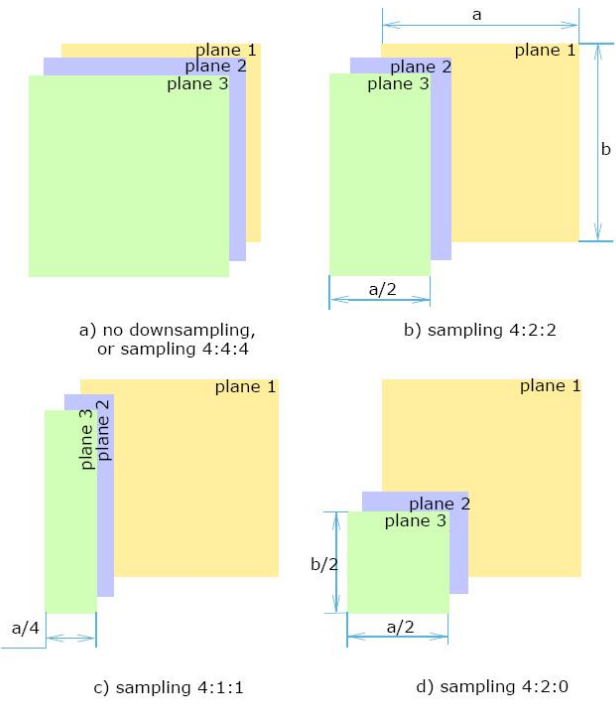
Planar image formats supported by the Intel IPP color conversion functions are listed in Table “Planar Image Formats” along with examples of the Intel IPP functions using that format. Planes layout and their relative sizes are shown in [Figure Plane Size and Layout: 3-planes Images](#) and [Figure Plane Size and Layout: 2-planes Images](#).

#### Planar Image Formats

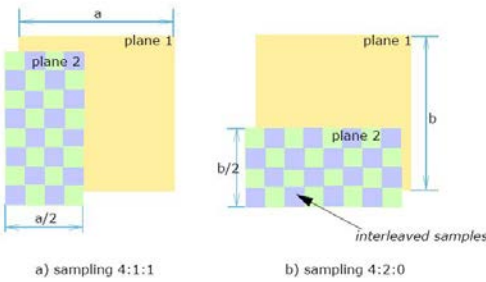
Image Format	Number of Planes	Planes Layout				Example
		pl.1	pl.2	pl.3		
RGB	3	R	G	B	see <a href="#">Figure below</a> , a	ippi <b>RGB</b> ToYUV_8u_ <b>P3R</b>
YUV	3	Y	U	V	see <a href="#">Figure below</a> , a	ippi <b>YUV</b> ToRGB_8u_ <b>P3R</b>
4:2:2 YUV	3	Y	U	V	see <a href="#">Figure below</a> , b	ippi <b>YUV422</b> ToRGB_8u_ <b>P3</b>
4:2:0 YUV	3	Y	U	V	see <a href="#">Figure below</a> , d	ippi <b>YUV420</b> ToRGB_8u_ <b>P3C3R</b>
YCbCr	3	Y	Cb	Cr	see <a href="#">Figure below</a> , a	ippi <b>YCbCr</b> ToRGB_8u_ <b>P3R</b>
4:2:2 YCbCr	3	Y	Cb	Cr	see <a href="#">Figure below</a> , b	ippi <b>YCbCr422</b> _8u_ <b>P3C2R</b>
4:1:1 YCbCr	3	Y	Cb	Cr	see <a href="#">Figure below</a> , c	ippi <b>YCbCr411</b> _8u_ <b>P3P2R</b>
4:1:1 YCbCr	2	Y	CbCr	-	see <a href="#">Figure below for 2-planes images</a> , a	ippi <b>YCbCr411</b> _8u_ <b>P2P3R</b>
4:2:0 YCbCr	3	Y	Cb	Cr	see <a href="#">Figure below</a> , d	ippi <b>RGBToYCbCr420</b> _8u_ <b>C3P3R</b>
4:2:0 YCbCr	2	Y	CbCr	-	see <a href="#">Figure below for 2-planes images</a> , b	ippi <b>YCbCr420</b> _8u_ <b>P2P3R</b>

Image Format	Number of Planes	Planes Layout			Example
		pl.1	pl.2	pl.3	
4:2:0 YCrCb	3	Y	Cr	Cb	see <a href="#">Figure below</a> , d
					ippiYCrCb420ToYCbCr422_8u_P3R

### Plane Size and Layout - 3-planes Images



### Plane Size and Layout - 2-planes Images



## Color Model Conversion

### RGBToYUV

*Converts an RGB image to the YUV color model.*

#### Syntax

##### Case 1: Operation on pixel-order data

```
IppStatus ippRGBToYUV_<mod>(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst, int dstStep,
IppiSize roiSize);
```

Supported values for mod:

8u\_C3R      8u\_AC4R

##### Case 2: Operation on planar data

```
IppStatus ippRGBToYUV_8u_P3R(const Ipp8u* const pSrc[3], int srcStep, Ipp8u* pDst[3],
int dstStep, IppiSize roiSize);
```

##### Case 3: Conversion from pixel-order to planar data

```
IppStatus ippRGBToYUV_8u_C3P3R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst[3],int dstStep,
IppiSize roiSize);
```

```
IppStatus ippRGBToYUV_8u_AC4P4R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst[4], int
dstStep, IppiSize roiSize);
```

#### Parameters

<i>pSrc</i>	Pointer to the source image ROI for pixel-order data. An array of pointers to the source image ROI in separate color planes in case of planar data.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination ROI for pixel-order data. An array of pointers to destination buffers in separate color planes in case of planar data.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>roiSize</i>	Size of the source and destination ROI in pixels.

#### Description

The function `ippRGBToYUV` is declared in the `ippcc.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts the gamma-corrected R'G'B' image *pSrc* to the Y'U'V' image *pDst* (see [Figure Converting an RGB image to YUV](#)) according to the following formulas:

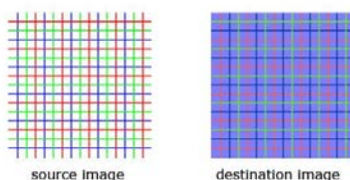
$$Y' = 0.299 * R' + 0.587 * G' + 0.114 * B'$$

$$U' = -0.147 * R' - 0.289 * G' + 0.436 * B' = 0.492 * (B' - Y')$$

$$V' = 0.615 * R' - 0.515 * G' - 0.100 * B' = 0.877 * (R' - Y')$$

For digital RGB values in the range [0..255],  $Y'$  has the range [0..255],  $U$  varies in the range [-112..+112], and  $V$  in the range [-157..+157]. To fit in the range of [0..255], a constant value 128 is added to computed  $U$  and  $V$  values, and  $V$  is then saturated.

## Converting an RGB image to YUV



## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <code>pSrc</code> or <code>pDst</code> pointer is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <code>roiSize</code> has a field with a zero or negative value.

## Example

The code example below shows how to use the function `ippiRGBToYUV_8u_C3R`.

```
# define nChannels 3

int main () {
    Ipp 8 u src [3*3* nChannels ] = {
        255, 0, 0, 255, 0, 0, 255, 0, 0,
        0, 255, 0, 0, 255, 0, 0, 255, 0,
        0, 0, 255, 0, 0, 255, 0, 0, 255};
    Ipp 8 u dst [3*3* nChannels ];
    IppiSize roiSize = { 3, 3 };
    IppStatus st = ippStsNoErr ;
    int srcStep = 3* nChannels ;
    int dstStep = 3* nChannels ;
    st = ippiRGBToYUV_8u_C3R ( src , srcStep , dst , dstStep , roiSize );
    if ( st == ippStsNoErr){
        printf("\n ***** passed *****\n");
    }else{
        printf("\n ***** failed *****\t");
    }
    return 0;
}
```

Result:

```
255  0  0 255  0  0 255  0  0
  0 255  0  0 255  0  0 255  0   src
  0  0 255  0  0 255  0  0 255
```



```

76 90 255 76 90 255 76 90 255
149 54 0 149 54 0 149 54 0 dst
29 239 102 29 239 102 29 239 102

```

## YUVToRGB

*Converts a YUV image to the RGB color model.*

### Syntax

#### Case 1: Operation on pixel-order data

```

IppStatus ippiYUVToRGB_<mod>(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst, int dstStep,
IppiSize roiSize);

```

Supported values for mod:

```

8u_C3R
8u_AC4R

```

```

IppStatus ippiYUVToRGB_8u_C3C4R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst, int dstStep,
IppiSize roiSize, Ippi8u aval);

```

#### Case 2: Operation on planar data

```

IppStatus ippiYUVToRGB_8u_P3R(const Ipp8u* const pSrc[3], int srcStep, Ipp8u* pDst[3],
int dstStep, IppiSize roiSize);

```

#### Case 3: Conversion from planar to pixel-order data

```

IppStatus ippiYUVToRGB_8u_P3C3R(const Ipp8u* const pSrc[3], int srcStep, Ipp8u* pDst, int
dstStep, IppiSize roiSize);

```

### Parameters

<i>pSrc</i>	Pointer to the source buffer for pixel-order data. An array of pointers to separate source color planes in case of planar data.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination buffer for pixel-order data. An array of pointers to separate destination color planes in case of planar data.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>aval</i>	Constant value to create the fourth channel.
<i>roiSize</i>	Size of the source and destination ROI in pixels.

### Description

The function `ippiYUVToRGB` is declared in the `ippcc.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts the **Y'U'V'** image *pSrc* to the gamma-corrected **R'G'B'** image *pDst* according to the following formulas:

$$R' = Y' + 1.140 * V'$$

$$G' = Y' - 0.394 * U' - 0.581 * V'$$

$$B' = Y' + 2.032 * U'$$

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pSrc</i> or <i>pDst</i> is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with a zero or negative value.

## RGBToYUV422

*Converts an RGB image to the YUV color model; uses 4:2:2 sampling.*

---

### Syntax

#### Case 1: Operation on pixel-order data

```
ippStatus ippRGBToYUV422_8u_C3C2R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst, int dstStep, IppiSize roiSize);
```

#### Case 2: Operation on planar data with ROI

```
ippStatus ippRGBToYUV422_8u_P3R(const Ipp8u* const pSrc[3], int srcStep, Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);
```

#### Case 3: Operation on planar data without ROI

```
ippStatus ippRGBToYUV422_8u_P3(const Ipp8u* const pSrc[3], Ipp8u* pDst[3], IppiSize imgSize);
```

#### Case 4: Conversion from pixel-order to planar data with ROI

```
ippStatus ippRGBToYUV422_8u_C3P3R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);
```

#### Case 5: Conversion from pixel-order to planar data without ROI

```
ippStatus ippRGBToYUV422_8u_C3P3(const Ipp8u* pSrc, Ipp8u* pDst[3], IppiSize imgSize);
```

## Parameters

<i>pSrc</i>	Pointer to the source image buffer for pixel-order image. An array of pointers to the source image buffer in each color plane for planar image.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image for operations with ROI.

<i>pDst</i>	Pointer to the destination image buffer for pixel-order image. An array of pointers to the destination image buffer in each color plane for planar image.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image for operations with ROI. An array of three values for planar image.
<i>roiSize</i>	Size of the source and destination ROI in pixels.
<i>imgSize</i>	Size of the source and destination images in pixels for operations without ROI.

## Description

The function `ippiRGBToYUV422` is declared in the `ippcc.h` file. This function converts the gamma-corrected R'G'B' image *pSrc* to the Y'U'V' image *pDst* with **4:2:2 sampling** format, according to the same formulas as the function `ippiRGBToYUV` does. For more details on this sampling format, see [Table "Pixel-Order Image Formats"](#) and [Table "Planar Image Formats"](#).

For digital RGB values in the range [0..255], Y' has the range [0..255], U varies in the range [-112..+112], and V in the range [-157..+157]. To fit in the range of [0..255], the constant value 128 is added to computed U and V values, and V is then saturated.

Some function flavors operates with ROI (see [Regions of Interest in Intel IPP](#)). The function flavors that does not use ROI operate on the assumption that both the source and destination images have the same size and occupy a contiguous memory area, which means that image rows are not padded with zeroes. In this case the step parameters are not needed.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pSrc</i> or <i>pDst</i> pointer is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> or <i>imgSize</i> has a field with a zero or negative value.

## YUV422ToRGB

*Converts a YUV image with the 4:2:2 sampling to the RGB color model.*

## Syntax

### Case 1: Operation on pixel-order data

```
ippStatus ippiYUV422ToRGB_8u_C2C3R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst, int dstStep, IppiSize roiSize);
```

### Case 2: Operation on planar data with ROI

```
ippStatus ippiYUV422ToRGB_8u_P3R(const Ipp8u* const pSrc[3], int [3], srcStep[3], Ipp8u* pDst[3], int dstStep, IppiSize roiSize);
```

## Case 3: Operation on planar data without ROI

```
IppStatus ippiYUV422ToRGB_8u_P3(const Ipp8u* const pSrc[3], Ipp8u* pDst[3], IppiSize
imgSize);
```

## Case 4: Conversion from planar to pixel-order data with ROI

```
IppStatus ippiYUV422ToRGB_<mod>(const Ipp8u* const pSrc[3], int srcStep[3], Ipp8u* pDst,
int dstStep, IppiSize roiSize);
```

Supported values for mod:

8u\_P3C3R    8u\_P3AC4R

## Case 5: Conversion from planar to pixel-order data without ROI

```
IppStatus ippiYUV422ToRGB_8u_P3C3(const Ipp8u* const pSrc[3], Ipp8u* pDst, IppiSize
imgSize);
```

## Parameters

<i>pSrc</i>	Pointer to the source image buffer for pixel-order image. An array of pointers to the source image buffer in each color plane for planar image.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image for operations with ROI. An array of three values in case of planar image.
<i>pDst</i>	Pointer to the destination image buffer for pixel-order image. An array of pointers to the destination image buffers in each color plane for planar image.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image for operations with ROI.
<i>roiSize</i>	Size of the source and destination ROI in pixels.
<i>imgSize</i>	Size of the source and destination images in pixels for operations without ROI.

## Description

The function `ippiYUV422ToRGB` is declared in the `ippcc.h` file. This function converts the `Y'U'V'` image *pSrc* to the gamma-corrected `R'G'B'` image *pDst* according to the same formulas as the function `ippiYUVToRGB` does. The difference is that `ippiYUV422ToRGB` [4:2:0 sampling](#) the input data to be in [4:2:2 sampling](#) format (see [Table "Pixel-Order Image Formats"](#) and [Table "Planar Image Formats"](#) for more details).

The function `ippiYUV422ToRGB_P3AC4R` additionally creates an alpha channel in the destination image with alpha values set to zero.

Some function flavors operates with ROI (see [Regions of Interest in Intel IPP](#)). The function flavors that do not use ROI operate on the assumption that both the source and destination images have the same size and occupy a contiguous memory area, which means that image rows are not padded with zeroes. In this case the step arguments are not needed.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <code>pSrc</code> or <code>pDst</code> is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <code>roiSize</code> or <code>imgSize</code> has a field with a zero or negative value.

## RGB565ToYUV422

Convert 16-bit RGB image to the 4:2:2 YUV image.

### Syntax

```
IppStatus ippRGB565ToYUV422_16u8u_C3P3R(const Ipp16u* pSrc, int srcStep, Ipp8u* pDst[3],
int dstStep[3], IppiSize roiSize);
```

### Parameters

<code>pSrc</code>	Pointer to the source image ROI.
<code>srcStep</code>	Distance in bytes between starts of consecutive lines in the source image.
<code>pDst</code>	Array of pointers to ROI in the separate color planes in the destination image.
<code>dstStep</code>	Array of distances in bytes between starts of consecutive lines in the destination image planes.
<code>roiSize</code>	Size of the source and destination ROI in pixels.

### Description

The function `ippRGB565ToYUV422` is declared in the `ippcc.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts the gamma-corrected R'G'B' image `pSrc` to the Y'U'V' image `pDst` according to the same formulas as the function `ippRGBToYUV` does.

The source image `pSrc` is a packed 16-bit RGB565 image with reduced bit depth; all 3 channel intensities for a pixel are packed into two consecutive bytes (16u data type) in the following order: 5 bits for red, 6 bits for green, 5 bits for blue (see [Figure 16-bit Pixel Formats](#)).

The destination image `pDst` has a 4:2:2 sampling format (see [Table "Planar Image Formats"](#) for more details).

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <code>pSrc</code> or <code>pDst</code> is NULL.
<code>ippStsDoubleSize</code>	Indicates a warning if <code>roiSize</code> has a field that is not a multiple of 2.

## RGBToYUV420

*Converts an RGB image to the 4:2:0 YUV image.*

---

### Syntax

#### Case 1: Operation on planar data with ROI

```
IppStatus ippiRGBToYUV420_8u_P3R(const Ipp8u* const pSrc[3], int srcStep, Ipp8u* pDst[3],
int dstStep[3], IppiSize roiSize);
```

#### Case 2: Operation on planar data without ROI

```
IppStatus ippiRGBToYUV420_8u_P3(const Ipp8u* const pSrc[3], Ipp8u* pDst[3], IppiSize
imgSize);
```

#### Case 3: Conversion from pixel-order to planar data with ROI

```
IppStatus ippiRGBToYUV420_8u_C3P3R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst[3], int
dstStep[3], IppiSize roiSize);
```

#### Case 4: Conversion from pixel-order to planar data without ROI

```
IppStatus ippiRGBToYUV420_8u_C3P3(const Ipp8u* pSrc, Ipp8u* pDst[3], IppiSize imgSize);
```

### Parameters

<i>pSrc</i>	Pointer to the source image buffer for pixel-order image. An array of pointers to the source image buffers in each color plane for planar image.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image for operations with ROI.
<i>pDst</i>	An array of pointers to the destination image buffers in each color plane.
<i>dstStep</i>	An array of distances in bytes between starts of consecutive lines in each plane of the destination image for operations with ROI.
<i>roiSize</i>	Size of the source and destination ROI in pixels.
<i>imgSize</i>	Size of the source and destination images in pixels for operations without ROI.

### Description

The function `ippiRGBToYUV420` is declared in the `ippcc.h` file. This function converts the gamma-corrected R'G'B' image *pSrc* to the Y'U'V' image *pDst* with the 4:2:0 sampling (see [Table "Planar Image Formats"](#) for more details). The conversion is performed in the accordance with the same formulas as the function `ippiRGBToYUV` does.

For digital RGB values in the range [0..255], Y' has the range [0..255], U varies in the range [-112..+112], and V in the range [-157..+157]. To fit in the range of [0..255], a constant value 128 is added to computed U and V values, and V is then saturated.

Some function flavors operates with ROI see [Regions of Interest in Intel IPP](#)).

The function flavors that does not use ROI operate on the assumption that both the source and destination images have the same size and occupy a contiguous memory area, which means that image rows are not padded with zeroes. In this case the step parameters are not needed.

`roiSize.width(imgSize.width)` and `roiSize.height(imgSize.height)` should be multiples of 2. Otherwise, the function reduces their original values to the nearest multiples of 2, performs operation, and returns warning message.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <code>pSrc</code> or <code>pDst</code> is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <code>roiSize</code> or <code>imgSize</code> has a field with a zero or negative value.
<code>ippStsDoubleSize</code>	Indicates a warning if <code>roiSize</code> or <code>imgSize</code> has a field that is not a multiple of 2.

## YUV420ToRGB

*Converts a YUV image that has 4:2:0 sampling format to the RGB image.*

### Syntax

#### Case 1: Operation on planar data with ROI

```
IppStatus ippYUV420ToRGB_8u_P3R(const Ipp8u* const pSrc[3], int srcStep[3], Ipp8u* pDst[3],
int dstStep, IppiSize roiSize);
```

#### Case 2: Operation on planar data without ROI

```
IppStatus ippYUV420ToRGB_8u_P3(const Ipp8u* const pSrc[3], Ipp8u* pDst[3], IppiSize
imgSize);
```

#### Case 3: Conversion from planar to pixel-order data with ROI

```
IppStatus ippYUV420ToRGB_<mod>(const Ipp8u* const pSrc[3], int srcStep[3], Ipp8u* pDst,
int dstStep, IppiSize roiSize);
```

Supported values for `mod`:

`8u_P3C3R`    `8u_P3AC4R`

#### Case 4: Conversion from planar to pixel-order data without ROI

```
IppStatus ippYUV420ToRGB_8u_P3C3(const Ipp8u* const pSrc[3], Ipp8u* pDst, IppiSize
imgSize);
```

## Parameters

<code>pSrc</code>	An array of pointers to the source image buffers in each color plane.
<code>srcStep</code>	An array of distances in bytes between starts of consecutive lines in each source image planes for operations with ROI.

<i>pDst</i>	Pointer to the destination image buffer for pixel-order images. An array of pointers to the destination image buffers in each color plane for planar images.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image for operations with ROI.
<i>roiSize</i>	Size of the source and destination ROI in pixels.
<i>imgSize</i>	Size of the source and destination images in pixels for operations without ROI.

## Description

The function `ippiYUV420ToRGB` is declared in the `ippcc.h` file. This function converts the `Y'U'V'` image *pSrc* to the gamma-corrected `R'G'B'` image *pDst* according to the same formulas as the function `ippiYUVToRGB` does. The difference is that `ippiYUV420ToRGB` [4:2:0 sampling](#) the input data to be in [4:2:2 sampling](#) format (see [Table "Planar Image Formats"](#) for more details).

The function `ippiYUV420ToRGB_P3AC4R` additionally creates an alpha channel in the destination image with alpha values set to zero.

Some function flavors operates with ROI see [Regions of Interest in Intel IPP](#) ).

The function flavors that does not use ROI operate on the assumption that both the source and destination images have the same size and occupy a contiguous memory area, which means that image rows are not padded with zeroes. In this case the step parameters are not needed.

*roiSize.width* (*imgSize.width*) and *roiSize.height* (*imgSize.height*) should be multiples of 2. Otherwise, the function reduces their original values to the nearest multiples of 2, performs operation, and returns warning message.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pSrc</i> or <i>pDst</i> pointer is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> or <i>imgSize</i> has a field with a zero or negative value.
<code>ippStsDoubleSize</code>	Indicates a warning if <i>roiSize</i> or <i>imgSize</i> has a field that is not a multiple of 2.

## BGRToYUV420

*Converts an BGR image to the YUV color model; uses 4:2:0 sampling*

---

### Syntax

```
ippStatus ippiBGRToYUV420_8u_AC4P3R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);
```

### Parameters

<i>pSrc</i>	Pointer to the source image ROI.
-------------	----------------------------------



<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	An array of pointers to the destination image buffers in each color plane.
<i>dstStep</i>	An array of distances in bytes between starts of consecutive lines in each plane of the destination image.
<i>roiSize</i>	Size of the source and destination ROI in pixels.

## Description

The function `ippiBGRToYUV420` is declared in the `ippcc.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts the gamma-corrected B'G'R' image *pSrc* to the Y'U'V' image *pDst* with the 4:2:0 sampling (see [Table "Planar Image Formats"](#) for more details). The function uses the same formulas as the function `ippiRGBToYUV` does.

For digital BGR values in the range [0..255], Y' varies in the range [0..255], U - in the range [-112..+112], and V - in the range [-157..+157]. To fit in the range of [0..255], a constant value 128 is added to the computed U and V values, and V is then saturated.

*roiSize.width* and *roiSize.height* should be multiples of 2. If not the function reduces their original values to the nearest multiples of 2, performs operation, and returns warning message.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pSrc</i> or <i>pDst</i> is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> or <i>imgSize</i> has a field with a zero or negative value.
<code>ippStsDoubleSize</code>	Indicates a warning if <i>roiSize</i> or <i>imgSize</i> has a field that is not a multiple of 2.

## YUV420ToBGR

*Converts a YUV image that has 4:2:0 sampling to the BGR image.*

## Syntax

```
ippStatus ippiYUV420ToBGR_8u_P3C3R(const Ipp8u* pSrc[3], int srcStep[3], Ipp8u* pDst, int dstStep, IppiSize roiSize);
```

## Parameters

<i>pSrc</i>	An array of pointers to ROI in each color plane in the source image.
<i>srcStep</i>	An array of distances in bytes between starts of consecutive lines in the source image planes.
<i>pDst</i>	Pointer to the destination image ROI.

<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>roiSize</i>	Size of the source and destination ROI in pixels.

### Description

The function `ippiYUV420ToBGR` is declared in the `ippcc.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts the `Y'U'V'` image *pSrc* to the gamma-corrected `B'G'R'` image *pDst* according to the same formulas as the function `ippiYUVToRGB` does. The input data must be presented in the `4:2:0` [sampling](#) format (see [Table "Planar Image Formats"](#) for more details).

*roiSize.width* and *roiSize.height* should be multiples of 2. Otherwise, the function reduces their original values to the nearest multiples of 2, performs operation, and returns warning message.

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pSrc</i> or <i>pDst</i> is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with a zero or negative value.
<code>ippStsDoubleSize</code>	Indicates a warning if <i>roiSize</i> has a field that is not a multiple of 2.

## RGB565ToYUV420

Converts 16-bit RGB image to the 4:2:0 YUV image.

### Syntax

```
IppStatus ippiRGB565ToYUV420_16u8u_C3P3R(const Ipp16u* pSrc, int srcStep, Ipp8u* pDst[3],
int dstStep[3], IppiSize roiSize);
```

### Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Array of pointers to ROI in the separate color planes in the destination image.
<i>dstStep</i>	Array of distances in bytes between starts of consecutive lines in the destination image planes.
<i>roiSize</i>	Size of the source and destination ROI in pixels.

### Description

The function `ippiRGB565ToYUV420` is declared in the `ippcc.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts the gamma-corrected R'G'B' image *pSrc* to the Y'U'V' image *pDst* according to the same formulas as the function `ippiRGBToYUV` does.

The source image *pSrc* is a packed 16-bit RGB565 image with reduced bit depth; all 3 channel intensities for a pixel are packed into two consecutive bytes (16u data type) in the following order: 5 bits for red, 6 bits for green, 5 bits for blue (see [Figure 16-bit Pixel Formats](#)).

The destination image *pDst* has a 4:2:0 [sampling](#) format (see [Table "Planar Image Formats"](#) for more details).

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pSrc</i> or <i>pDst</i> is NULL.
<code>ippStsDoubleSize</code>	Indicates a warning if <i>roiSize</i> has a field that is not a multiple of 2.

## YUV420ToRGB565, YUV420ToRGB555, YUV420ToRGB444

Convert a YUV image that has 4:2:0 sampling format to the 16-bit per pixel RGB image.

## Syntax

```
ippStatus ippiYUV420ToRGB565_8u16u_P3C3R(const Ipp8u* const pSrc[3], int srcStep[3],
Ipp16u* pDst, int dstStep, IppiSize roiSize);
```

```
ippStatus ippiYUV420ToRGB555_8u16u_P3C3R(const Ipp8u* const pSrc[3], int srcStep[3],
Ipp16u* pDst, int dstStep, IppiSize roiSize);
```

```
ippStatus ippiYUV420ToRGB444_8u16u_P3C3R(const Ipp8u* const pSrc[3], int srcStep[3],
Ipp16u* pDst, int dstStep, IppiSize roiSize);
```

## Parameters

<i>pSrc</i>	An array of pointers to ROI in each color plane in the source image.
<i>srcStep</i>	An array of distances in bytes between starts of consecutive lines in the source image planes.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>roiSize</i>	Size of the source and destination ROI in pixels.

## Description

The functions `ippiYUV420ToRGB444`, `ippiYUV420ToRGB555`, `ippiYUV420ToRGB565` are declared in the `ippcc.h` file. They operate with ROI (see [Regions of Interest in Intel IPP](#)).

These functions convert the Y'U'V' image *pSrc* to the gamma-corrected R'G'B' image *pDst* according to the same formulas as the function `ippiYUVToRGB` does. The difference is that the input data are in 4:2:0 [sampling](#) format (see [Table "Planar Image Formats"](#) for more details). The destination image *pDst* is a packed 16-bit RGB image with reduced bit depth; all 3 channel intensities for a pixel are packed into two consecutive bytes (16u

data type). It can be in one of three possible formats (see [Figure 16-bit Pixel Formats](#)): RGB565 (5 bits for red, 6 bits for green, 5 bits for blue), RGB555 (5 bits for red, green, blue), or RGB444 (4 bits for red, green, blue). Bit reduction is performed by discarding the least significant bits in the image after color conversion.

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <code>pSrc</code> or <code>pDst</code> is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <code>roiSize</code> has a field with a zero or negative value.

## YUV420ToRGB565Dither, YUV420ToRGB555Dither, YUV420ToRGB444Dither

*Convert a YUV image that has 4:2:0 sampling format to the 16-bit per pixel RGB image with dithering.*

---

### Syntax

```
ippStatus ippYUV420ToRGB565Dither_8u16u_P3C3R(const Ipp8u* pSrc[3], int srcStep[3],
Ipp16u* pDst, int dstStep, IppiSize roiSize);
```

```
ippStatus ippYUV420ToRGB555Dither_8u16u_P3C3R(const Ipp8u* pSrc[3], int srcStep[3],
Ipp16u* pDst, int dstStep, IppiSize roiSize);
```

```
ippStatus ippYUV420ToRGB444Dither_8u16u_P3C3R(const Ipp8u* pSrc[3], int srcStep[3],
Ipp16u* pDst, int dstStep, IppiSize roiSize);
```

### Parameters

<code>pSrc</code>	An array of pointers to ROI in each color plane in the source image.
<code>srcStep</code>	An array of distances in bytes between starts of consecutive lines in the source image planes.
<code>pDst</code>	Pointer to the destination image ROI.
<code>dstStep</code>	Distance in bytes between starts of consecutive lines in the destination image.
<code>roiSize</code>	Size of the source and destination ROI in pixels.

### Description

The functions `ippYUV420ToRGB565Dither`, `ippYUV420ToRGB555Dither`, `ippYUV420ToRGB444Dither` are declared in the `ippcc.h` file. They operate with ROI (see [Regions of Interest in Intel IPP](#)).

These functions convert the `Y'U'V'` image `pSrc` to the gamma-corrected `R'G'B'` image `pDst` according to the same formulas as the function `ippYUVToRGB` does. The difference is that the input data are in **4:2:0 sampling** format (see [Table "Planar Image Formats"](#) for more details). The destination image `pDst` is a packed 16-bit RGB image with reduced bit depth; all three channel intensities for a pixel are packed into two consecutive bytes (16u data type). It can be in one of three possible formats (see [Figure 16-bit Pixel Formats](#)): RGB565 (5 bits for red, 6 bits for green, 5 bits for blue), RGB555 (5 bits for red, green, blue), or RGB444 (4 bits for red, green, blue). Bit reduction is performed using the Bayer's threshold dithering algorithm with a 4x4 matrix [\[Tho91\]](#).

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <code>pSrc</code> or <code>pDst</code> is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <code>roiSize</code> has a field with a zero or negative value.

## BGR565ToYUV420, BGR555ToYUV420

Convert 16-bit BGR image to the 4:2:0 YUV image.

### Syntax

```
IppStatus ippibGR565ToYUV420_16u8u_C3P3R(const Ipp16u* pSrc, int srcStep, Ipp8u* pDst[3],
int dstStep[3], IppiSize roiSize);
```

```
IppStatus ippibGR555ToYUV420_16u8u_C3P3R(const Ipp16u* pSrc, int srcStep, Ipp8u* pDst[3],
int dstStep[3], IppiSize roiSize);
```

### Parameters

<code>pSrc</code>	Pointer to the source image ROI.
<code>srcStep</code>	Distance in bytes between starts of consecutive lines in the source image.
<code>pDst</code>	Array of pointers to ROI in the separate color planes in the destination image.
<code>dstStep</code>	Array of instances in bytes between starts of consecutive lines in the destination image planes.
<code>roiSize</code>	Size of the source and destination ROI in pixels.

### Description

The functions `ippibGR565ToYUV420`, `ippibGR555ToYUV420` are declared in the `ippcc.h` file. They operate with ROI (see [Regions of Interest in Intel IPP](#)).

These functions convert the gamma-corrected B'G'R' image `pSrc` to the Y'U'V' image `pDst` according to the same formulas as the function `ippiRGBToYUV` does. The source image `pSrc` is a packed 16-bit BGR image with reduced bit depth; all 3 channel intensities for a pixel are packed into two consecutive bytes (16u data type). It can be in one of two possible formats (see [Figure 16-bit Pixel Formats](#)): BGR565 (5 bits for blue, 6 bits for green, 5 bits for red), and BGR555 (5 bits for blue, green, red). The destination image `pDst` has a 4:2:0 sampling format (see [Table "Planar Image Formats"](#) for more details).

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <code>pSrc</code> or <code>pDst</code> is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <code>roiSize.width</code> is less than 2 or <code>roiSize.height</code> is less than 0.

## YUV420ToBGR565, YUV420ToBGR555, YUV420ToBGR444

*Convert a YUV image that has 4:2:0 sampling format to the 16-bit per pixel BGR image.*

---

### Syntax

```
IppStatus ippYUV420ToBGR565_8u16u_P3C3R(const Ipp8u* const pSrc[3], int srcStep[3],
Ipp16u* pDst, int dstStep, IppiSize roiSize);
```

```
IppStatus ippYUV420ToBGR555_8u16u_P3C3R(const Ipp8u* const pSrc[3], int srcStep[3],
Ipp16u* pDst, int dstStep, IppiSize roiSize);
```

```
IppStatus ippYUV420ToBGR444_8u16u_P3C3R(const Ipp8u* const pSrc[3], int srcStep[3],
Ipp16u* pDst, int dstStep, IppiSize roiSize);
```

### Parameters

<i>pSrc</i>	An array of pointers to ROI in each separate color plane in the source image.
<i>srcStep</i>	An array of distances in bytes between starts of consecutive lines in the source image planes.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>roiSize</i>	Size of the source and destination ROI in pixels.

### Description

The functions `ippYUV420ToBGR565`, `ippYUV420ToBGR555`, `ippYUV420ToBGR444` are declared in the `ippcc.h` file. They operate with ROI (see [Regions of Interest in Intel IPP](#)).

These functions convert the `Y'U'V'` image *pSrc* to the gamma-corrected `B'G'R'` image *pDst* according to the same formulas as the function `ippYUVToRGB` does. The difference is that the input data are in [4:2:0 sampling](#) format (see [Table "Planar Image Formats"](#) for more details). The destination image *pDst* is a packed 16-bit BGR image with reduced bit depth; all 3 channel intensities for a pixel are packed into two consecutive bytes (16u data type). It can be in one of three possible formats (see [Figure 16-bit Pixel Formats](#)): BGR565 (5 bits for blue, 6 bits for green, 5 bits for red), BGR555 (5 bits for blue, green, red), or BGR444 (4 bits for blue, green, red). Bit reduction is performed by discarding the least significant bits in the image after color conversion.

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pSrc</i> or <i>pDst</i> is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with a zero or negative value.

## YUV420ToBGR565Dither, YUV420ToBGR555Dither, YUV420ToBGR444Dither

*Convert a YUV image that has 4:2:0 sampling format to the 16-bit per pixel BGR image with dithering.*

### Syntax

```
IppStatus ippiYUV420ToBGR565Dither_8u16u_P3C3R(const Ipp8u* pSrc[3], int srcStep[3],
Ipp16u* pDst, int dstStep, IppiSize roiSize);
```

```
IppStatus ippiYUV420ToBGR555Dither_8u16u_P3C3R(const Ipp8u* pSrc[3], int srcStep[3],
Ipp16u* pDst, int dstStep, IppiSize roiSize);
```

```
IppStatus ippiYUV420ToBGR444Dither_8u16u_P3C3R(const Ipp8u* pSrc[3], int srcStep[3],
Ipp16u* pDst, int dstStep, IppiSize roiSize);
```

### Parameters

<i>pSrc</i>	An array of pointers to ROI in each separate color plane in the source image.
<i>srcStep</i>	An array of distances in bytes between starts of consecutive lines in the source image planes.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>roiSize</i>	Size of the source and destination ROI in pixels.

### Description

The functions `ippiYUV420ToBGR565Dither`, `ippiYUV420ToBGR555Dither`, `ippiYUV420ToBGR444Dither` are declared in the `ippcc.h` file. They operate with ROI (see [Regions of Interest in Intel IPP](#)).

These functions convert the `Y'U'V'` image *pSrc* to the gamma-corrected `B'G'R'` image *pDst* according to the same formulas as the function `ippiYUVToRGB` does. The difference is that the input data are in [4:2:0 sampling](#) format (see [Table "Planar Image Formats"](#) for more details). The destination image *pDst* is a packed 16-bit BGR image with reduced bit depth; all 3 channel intensities for a pixel are packed into two consecutive bytes (16u data type). It can be in one of three possible formats (see [Figure 16-bit Pixel Formats](#)): BGR565 (5 bits for blue, 6 bits for green, 5 bits for red), BGR555 (5 bits for blue, green, red), or BGR444 (4 bits for blue, green, red). Bit reduction is performed using the Bayer's threshold dithering algorithm with a 4x4 matrix [\[Tho91\]](#).

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pSrc</i> or <i>pDst</i> is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with a zero or negative value.

## RGBToYCbCr

*Converts an RGB image to the YCbCr color model.*

---

### Syntax

#### Case 1: Operation on pixel-order data

```
IppStatus ippiRGBToYCbCr_<mod>(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst, int dstStep, IppiSize roiSize);
```

Supported values for mod:

8u\_C3R      8u\_AC4R

#### Case 2: Operation on planar data

```
IppStatus ippiRGBToYCbCr_8u_P3R(const Ipp8u* const pSrc[3], int srcStep, Ipp8u* pDst[3], int dstStep, IppiSize roiSize);
```

#### Case 3: Conversion from pixel-order to planar data

```
IppStatus ippiRGBToYCbCr_<mod>(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst[3], int dstStep, IppiSize roiSize);
```

Supported values for mod:

8u\_C3P3R    8u\_AC4P3R

### Parameters

<i>pSrc</i>	Pointer to the source image ROI for a pixel-order image. An array of pointers to ROI in each separate source color planes for planar images.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI. Array of pointers to ROI in the separate destination color planes for planar images.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>roiSize</i>	Size of the source and destination ROI in pixels.

### Description

The function `ippiRGBToYCbCr` is declared in the `ippcc.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts the gamma-corrected R'G'B' image *pSrc* with values in the range [0..255] to the Y'Cb'Cr' image *pDst* according to the following formulas:

$$Y' = 0.257 \cdot R' + 0.504 \cdot G' + 0.098 \cdot B' + 16$$

$$Cb' = -0.148 \cdot R' - 0.291 \cdot G' + 0.439 \cdot B' + 128$$

$$Cr' = 0.439 \cdot R' - 0.368 \cdot G' - 0.071 \cdot B' + 128$$



In the YCbCr model, Y is defined to have a nominal range [16..235], while Cb and Cr are defined to have a range [16..240], with the value of 128 as corresponding to zero.

Both the source and destination images have the same bit depth.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <code>pSrc</code> or <code>pDst</code> is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <code>roiSize</code> has a field with a zero or negative value.

## YCbCrToRGB

*Converts a YCbCr image to the RGB color model.*

### Syntax

#### Case 1: Operation on pixel-order data

```
IppStatus ippYCbCrToRGB_<mod>(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst, int dstStep, IppiSize roiSize);
```

Supported values for mod:

`8u_C3R`                `8u_AC4R`

#### Case 2: Operation on planar data

```
IppStatus ippYCbCrToRGB_<mod>(const Ipp8u* const pSrc[3], int srcStep, Ipp8u* pDst[3], int dstStep, IppiSize roiSize);
```

Supported values for mod:

`8u_P3R`                `8u_P3C3R`

#### Case 3: Conversion from planar to pixel-order data

```
IppStatus ippYCbCrToRGB_8u_P3C4R(const Ipp8u* pSrc[3], int srcStep, Ipp8u* pDst, int dstStep, IppiSize roiSize, Ipp8u aval);
```

### Parameters

<code>pSrc</code>	Pointer to the source image ROI for pixel-order image. Array of pointers to the ROI in each separate source color planes for planar images.
<code>srcStep</code>	Distance in bytes between starts of consecutive lines in the source image.
<code>pDst</code>	Pointer to the destination image ROI. Array of pointers to the ROI in the separate destination color planes for planar images.
<code>dstStep</code>	Distance in bytes between starts of consecutive lines in the destination image.

<i>roiSize</i>	Size of the source and destination ROI in pixels.
<i>aval</i>	Constant value to create the fourth channel.

## Description

The function `ippiYCbCrToRGB` is declared in the `ippcc.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts the `Y'Cb'Cr'` image *pSrc* to the 24-bit gamma-corrected `R'G'B'` image *pDst*. The following formulas are used for conversion:

$$R' = 1.164 * (Y' - 16) + 1.596 * (Cr' - 128)$$

$$G' = 1.164 * (Y' - 16) - 0.813 * (Cr' - 128) - 0.392 * (Cb' - 128)$$

$$B' = 1.164 * (Y' - 16) + 2.017 * (Cb' - 128)$$

The output `R'G'B'` values are saturated to the range [0..255].

The fourth channel is created by setting channel values to the constant value *aval*.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pSrc</i> or <i>pDst</i> pointer is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with a zero or negative value.

## YCbCrToBGR

*Converts a YCbCr image to the BGR color model.*

---

## Syntax

```
IppStatus ippiYCbCrToBGR_8u_P3C3R(const Ipp8u* pSrc[3], int srcStep, Ipp8u* pDst, int dstStep, IppiSize roiSize);
```

```
IppStatus ippiYCbCrToBGR_8u_P3C4R(const Ipp8u* pSrc[3], int srcStep, Ipp8u* pDst, int dstStep, IppiSize roiSize, Ipp8u aval);
```

## Parameters

<i>pSrc</i>	An array of pointers to ROI in each separate source color planes.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>roiSize</i>	Size of the source and destination ROI in pixels.
<i>aval</i>	Constant value to create the fourth channel.

## Description

The function `ippiYCbCrToBGR` is declared in the `ippcc.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts the `Y'Cb'Cr'` image `pSrc` to the 24-bit gamma-corrected `B'G'R'` image `pDst` according to the same formulas as the function `ippiYCbCrToRGB` does. The output `B'G'R'` values are saturated to the range `[0..255]`.

The fourth channel is created by setting channel values to the constant value `aval`.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <code>pSrc</code> or <code>pDst</code> is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <code>roiSize.width</code> is less than 4 or <code>roiSize.height</code> is less than 1.

## YCbCrToBGR\_709CSC

*Converts a YCbCr image to the BGR image for ITU-R BT.709 CSC signal.*

## Syntax

```
ippStatus ippiYCbCrToBGR_709CSC_8u_P3C3R(const Ipp8u* pSrc[3], int srcStep, Ipp8u* pDst,
int dstStep, IppiSize roiSize);
```

```
ippStatus ippiYCbCrToBGR_709CSC_8u_P3C4R(const Ipp8u* pSrc[3], int srcStep, Ipp8u* pDst,
int dstStep, IppiSize roiSize, Ipp8u aval);
```

## Parameters

<code>pSrc</code>	An array of pointers to ROI in separate planes of the source image.
<code>srcStep</code>	An array of distances in bytes between starts of consecutive lines in the source image planes.
<code>pDst</code>	Pointer to the destination image ROI.
<code>dstStep</code>	Distance in bytes between starts of consecutive lines in the destination image.
<code>roiSize</code>	Size of the source and destination ROI in pixels.
<code>aval</code>	Constant value to create fourth channel.

## Description

The function `ippiYCbCrToBGR_709CSC` is declared in the `ippcc.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts a planar `Y'Cb'Cr'` image `pSrc` to the three- or four-channel gamma-corrected `B'G'R'` image `pDst` for digital component video signals complied with the ITU-R BT.709 Recommendation [\[ITU709\]](#) for computer systems consideration (CSC). The conversion is performed according to the following formulas [\[Jack01\]](#):

$$R' = 1.164 * (Y' - 16) + 1.793 * (Cr' - 128)$$

$$G' = 1.164 * (Y' - 16) - 0.534 * (Cr' - 128) - 0.213 * (Cb' - 128)$$

$$B' = 1.164 * (Y' - 16) + 2.115 * (Cb' - 128)$$

The output R'G'B' values are saturated to the range [0..255].

The fourth channel is created by setting channel values to the constant value *aval*.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pSrc</i> or <i>pDst</i> is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with a zero or negative value.

## YCbCrToRGB565, YCbCrToRGB555, YCbCrToRGB444

Convert a YCbCr image to the 16-bit per pixel RGB image.

### Syntax

#### Case 1: Operation on pixel-order data

```
IppStatus ippYCbCrToRGB565_8u16u_C3R(const Ipp8u* pSrc, int srcStep, Ipp16u* pDst, int dstStep, IppiSize roiSize);
```

```
IppStatus ippYCbCrToRGB555_8u16u_C3R(const Ipp8u* pSrc, int srcStep, Ipp16u* pDst, int dstStep, IppiSize roiSize);
```

```
IppStatus ippYCbCrToRGB444_8u16u_C3R(const Ipp8u* pSrc, int srcStep, Ipp16u* pDst, int dstStep, IppiSize roiSize);
```

#### Case 2: Conversion from planar to pixel-order data

```
IppStatus ippYCbCrToRGB565_8u16u_P3C3R(const Ipp8u* pSrc[3], int srcStep, Ipp16u* pDst, int dstStep, IppiSize roiSize);
```

```
IppStatus ippYCbCrToRGB555_8u16u_P3C3R(const Ipp8u* pSrc[3], int srcStep, Ipp16u* pDst, int dstStep, IppiSize roiSize);
```

```
IppStatus ippYCbCrToRGB444_8u16u_P3C3R(const Ipp8u* pSrc[3], int srcStep, Ipp16u* pDst, int dstStep, IppiSize roiSize);
```

### Parameters

<i>pSrc</i>	Pointer to the source image ROI for pixel-order image. An array of pointers to ROI in each separate source color planes for planar images.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.

*roiSize*                      Size of the source and destination ROI in pixels.

## Description

The functions `ippiYCbCrToRGB565`, `ippiYCbCrToRGB555`, `ippiYCbCrToRGB444` are declared in the `ippcc.h` file. They operate with ROI (see [Regions of Interest in Intel IPP](#)).

These functions convert the `Y'Cb'Cr'` image *pSrc* to the gamma-corrected `R'G'B'` image *pDst* according to the same formulas as the function `ippiYCbCrToRGB` does.

The destination image *pDst* is a packed 16-bit RGB image with reduced bit depth; all 3 channel intensities for a pixel are packed into two consecutive bytes (16u data type). It can be in one of three possible formats (see [Figure 16-bit Pixel Formats](#) for more details): `RGB565` (5 bits for red, 6 bits for green, 5 bits for blue), `RGB555` (5 bits for red, green, blue), or `RGB444` (4 bits for red, green, blue). Bit reduction is performed by discarding the least significant bits in the image after color conversion.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pSrc</i> or <i>pDst</i> is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with a zero or negative value.

## YCbCrToRGB565Dither, YCbCrToRGB555Dither, YCbCrToRGB444Dither

*Convert a YCbCr image to the 16-bit per pixel RGB image with dithering.*

## Syntax

### Case 1: Operation on pixel-order data

```
ippStatus ippiYCbCrToRGB565Dither_8u16u_C3R(const Ipp8u* pSrc, int srcStep, Ipp16u* pDst,
int dstStep, IppiSize roiSize);
```

```
ippStatus ippiYCbCrToRGB555Dither_8u16u_C3R(const Ipp8u* pSrc, int srcStep, Ipp16u* pDst,
int dstStep, IppiSize roiSize);
```

```
ippStatus ippiYCbCrToRGB444Dither_8u16u_C3R(const Ipp8u* pSrc, int srcStep, Ipp16u* pDst,
int dstStep, IppiSize roiSize);
```

### Case 2: Conversion from planar to pixel-order data

```
ippStatus ippiYCbCrToRGB565Dither_8u16u_P3C3R(const Ipp8u* pSrc[3], int srcStep, Ipp16u*
pDst, int dstStep, IppiSize roiSize);
```

```
ippStatus ippiYCbCrToRGB555Dither_8u16u_P3C3R(const Ipp8u* pSrc[3], int srcStep, Ipp16u*
pDst, int dstStep, IppiSize roiSize);
```

```
ippStatus ippiYCbCrToRGB444Dither_8u16u_P3C3R(const Ipp8u* pSrc[3], int srcStep, Ipp16u*
pDst, int dstStep, IppiSize roiSize);
```

## Parameters

<i>pSrc</i>	Pointer to the source image ROI for pixel-order image. An array of pointers to ROI in each separate source color planes for planar images.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>roiSize</i>	Size of the source and destination ROI in pixels.

## Description

The functions `ippiYCbCrToRGB565Dither`, `ippiYCbCrToRGB555Dither`, `ippiYCbCrToRGB444Dither` are declared in the `ippcc.h` file. They operate with ROI (see [Regions of Interest in Intel IPP](#)).

These functions convert the 'Y'Cb'Cr' image *pSrc* to the gamma-corrected R'G'B' image *pDst* according to the same formulas as the function `ippiYCbCrToRGB` does.

The destination image *pDst* is a packed 16-bit RGB image with reduced bit depth; all 3 channel intensities for a pixel are packed into two consecutive bytes (16u data type). It can be in one of three possible formats (see [Figure 16-bit Pixel Formats](#) for more details): RGB565 (5 bits for red, 6 bits for green, 5 bits for blue), RGB555 (5 bits for red, green, blue), or RGB444 (4 bits for red, green, blue). Bit reduction is performed using the Bayer's threshold dithering algorithm with a 4x4 matrix [\[Tho91\]](#).

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pSrc</i> or <i>pDst</i> is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with a zero or negative value.

## YCbCrToBGR565, YCbCrToBGR555, YCbCrToBGR444

Convert a YCbCr image to the 16-bit per pixel BGR image.

### Syntax

#### Case 1: Operation on pixel-order data

```
ippStatus ippiYCbCrToBGR565_8u16u_C3R(const Ipp8u* pSrc, int srcStep, Ipp16u* pDst, int dstStep, IppiSize roiSize);
```

```
ippStatus ippiYCbCrToBGR555_8u16u_C3R(const Ipp8u* pSrc, int srcStep, Ipp16u* pDst, int dstStep, IppiSize roiSize);
```

```
ippStatus ippiYCbCrToBGR444_8u16u_C3R(const Ipp8u* pSrc, int srcStep, Ipp16u* pDst, int dstStep, IppiSize roiSize);
```

**Case 2: Conversion from planar to pixel-order data**

```
IppStatus ippiYCbCrToBGR565_8u16u_P3C3R(const Ipp8u* pSrc[3], int srcStep, Ipp16u* pDst,
int dstStep, IppiSize roiSize);
```

```
IppStatus ippiYCbCrToBGR555_8u16u_P3C3R(const Ipp8u* pSrc[3], int srcStep, Ipp16u* pDst,
int dstStep, IppiSize roiSize);
```

```
IppStatus ippiYCbCrToBGR444_8u16u_P3C3R(const Ipp8u* pSrc[3], int srcStep, Ipp16u* pDst,
int dstStep, IppiSize roiSize);
```

**Parameters**

<i>pSrc</i>	Pointer to the source image ROI for pixel-order image. An array of pointers to ROI in each separate source color planes for planar images.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>roiSize</i>	Size of the source and destination ROI in pixels.

**Description**

The functions `ippiYCbCrToBGR565`, `ippiYCbCrToBGR555`, `ippiYCbCrToBGR444` are declared in the `ippcc.h` file. They operate with ROI (see [Regions of Interest in Intel IPP](#)).

These functions convert the `Y'Cb'Cr'` image *pSrc* to the gamma-corrected `B'G'R'` image *pDst* according to the same formulas as the function `ippiYCbCrToRGB` does.

The destination image *pDst* is a packed 16-bit BGR image with reduced bit depth; all 3 channel intensities for a pixel are packed into two consecutive bytes (16u data type). It can be in one of three possible formats (see [Figure 16-bit Pixel Formats](#)): BGR565 (5 bits for blue, 6 bits for green, 5 bits for red), BGR555 (5 bits for blue, green, red), or BGR444 (4 bits for blue, green, red). Bit reduction is performed by discarding the least significant bits in the image after color conversion.

**Return Values**

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pSrc</i> or <i>pDst</i> is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with a zero or negative value.

## YCbCrToBGR565Dither, YCbCrToBGR555Dither, YCbCrToBGR444Dither

*Convert a YCbCr image to the 16-bit per pixel BGR image with dithering.*

---

### Syntax

#### Case 1: Operation on pixel-order data

```
IppStatus ippiYCbCrToBGR565Dither_8u16u_C3R(const Ipp8u* pSrc, int srcStep, Ipp16u* pDst,
int dstStep, IppiSize roiSize);
```

```
IppStatus ippiYCbCrToBGR555Dither_8u16u_C3R(const Ipp8u* pSrc, int srcStep, Ipp16u* pDst,
int dstStep, IppiSize roiSize);
```

```
IppStatus ippiYCbCrToBGR444Dither_8u16u_C3R(const Ipp8u* pSrc, int srcStep, Ipp16u* pDst,
int dstStep, IppiSize roiSize);
```

#### Case 2: Conversion from planar to pixel-order data

```
IppStatus ippiYCbCrToBGR565Dither_8u16u_P3C3R(const Ipp8u* pSrc[3], int srcStep, Ipp16u*
pDst, int dstStep, IppiSize roiSize);
```

```
IppStatus ippiYCbCrToBGR555Dither_8u16u_P3C3R(const Ipp8u* pSrc[3], int srcStep, Ipp16u*
pDst, int dstStep, IppiSize roiSize);
```

```
IppStatus ippiYCbCrToBGR444Dither_8u16u_P3C3R(const Ipp8u* pSrc[3], int srcStep, Ipp16u*
pDst, int dstStep, IppiSize roiSize);
```

### Parameters

<i>pSrc</i>	Pointer to the source image ROI for pixel-order image. An array of pointers to ROI in each separate source color planes for planar images.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>roiSize</i>	Size of the source and destination ROI in pixels.

### Description

The functions `ippiYCbCrToBGR565Dither`, `ippiYCbCrToBGR555Dither`, `ippiYCbCrToBGR444Dither` are declared in the `ippcc.h` file. They operate with ROI (see [Regions of Interest in Intel IPP](#)).

These functions convert the Y'Cb'Cr' image *pSrc* to the gamma-corrected B'G'R' image *pDst* according to the same formulas as the function `ippiYCbCrToBGR` does.

The destination image *pDst* is a packed 16-bit BGR image with reduced bit depth; all 3 channel intensities for a pixel are packed into two consecutive bytes (16u data type). It can be in one of three possible formats (see [Figure 16-bit Pixel Formats](#) for more details): BGR565 (5 bits for blue, 6 bits for green, 5 bits for red), BGR555 (5 bits for blue, green, red), or BGR444 (4 bits for blue, green, red). Bit reduction is performed using the Bayer's threshold dithering algorithm with a 4x4 matrix [Tho91].



## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pSrc</i> or <i>pDst</i> is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with a zero or negative value.

## RGBToYCbCr422

Converts an RGB image to the YCbCr image with 4:2:2 sampling.

### Syntax

#### Case 1: Operation on pixel-order data

```
IppStatus ippRGBToYCbCr422_8u_C3C2R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst, int dstStep, IppiSize roiSize);
```

#### Case 2; Conversion from pixel-order to planar data

```
IppStatus, ippRGBToYCbCr422_8u_C3P3R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);
```

#### Case 2: Conversion from planar to pixel-order data

```
IppStatus ippRGBToYCbCr422_8u_P3C2R(const Ipp8u* const pSrc[3], int srcStep, Ipp8u* pDst, int dstStep, IppiSize roiSize);
```

### Parameters

<i>pSrc</i>	Pointer to the source image ROI for pixel-order image. An array of pointers to ROI in each separate source color planes for planar images.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI for pixel-order image. An array of pointer to ROI in each separate planes for the planar destination image.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>roiSize</i>	Size of the source and destination ROI in pixels.

### Description

The function `ippRGBToYCbCr422` is declared in the `ippcc.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)). This function converts the gamma-corrected R'G'B' image *pSrc* to the Y'Cb'Cr' image *pDst* with 4:2:2 sampling (see [Table "Pixel-Order Image Formats"](#) and [Table "Planar Image Formats"](#) for more details). The conversion is performed according to the same formulas as the function `ippRGBToYCbCr` does.

The converted buffer for pixel-order image has the reduced bit depth of a 16 bits per pixel, whereas the source buffer has 24 bit depth.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pSrc</i> or <i>pDst</i> is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with a zero or negative value.

## YCbCr422ToRGB

*Converts an YCbCr image with the 4:2:2 sampling to the RGB image.*

---

### Syntax

#### Case 1: Operation on pixel-order data

```
IppStatus ippYCbCr422ToRGB_8u_C2C3R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst, int dstStep, IppiSize roiSize);
```

#### Case 2: Conversion from pixel-order to planar data

```
IppStatus ippYCbCr422ToRGB_8u_C2P3R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst[3], int dstStep, IppiSize roiSize);
```

#### Case 3: Conversion from planar to pixel-order data

```
IppStatus ippYCbCr422ToRGB_8u_P3C3R(const Ipp8u* const pSrc[3], int srcStep[3], Ipp8u* pDst, int dstStep, IppiSize roiSize);
```

### Parameters

<i>pSrc</i>	Pointer to the source image ROI for pixel-order image. An array of pointers to ROI in each separate source planes for planar images.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the ROI in the destination pixel-order image. An array of pointers to ROI in each planes of the destination planar image.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>roiSize</i>	Size of the source and destination ROI in pixels.

### Description

The function `ippYCbCr422ToRGB` is declared in the `ippcc.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts the `Y'Cb'Cr'` image *pSrc* with the `4:2:2` sampling (see [Table "Pixel-Order Image Formats"](#) and [Table "Planar Image Formats"](#) for more details) to the gamma-corrected `R'G'B'` image *pDst* according to the same formulas as the function `ippYCbCrToRGB` does. The output `R'G'B'` values are saturated to the range [0..255].

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pSrc</i> or <i>pDst</i> is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with a zero or negative value.

## RGBToYCrCb422

Converts 24-bit per pixel RGB image to 16-bit per pixel YCrCb image

---

### Syntax

#### Case 1: Operation on pixel-order data

```
IppStatus ippRGBToYCrCb422_8u_C3C2R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst, int dstStep, IppiSize roiSize);
```

#### Case 2: Conversion from planar to pixel-order data

```
IppStatus ippRGBToYCrCb422_8u_P3C2R(const Ipp8u* const pSrc[3], int srcStep, Ipp8u* pDst, int dstStep, IppiSize roiSize);
```

### Parameters

<i>pSrc</i>	Pointer to the source image ROI for pixel-order image. An array of pointers to ROI in each separate source color planes for planar images.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>roiSize</i>	Size of the source and destination ROI in pixels.

### Description

The function `ippRGBToYCrCb422` is declared in the `ippcc.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)). This function converts the gamma-corrected R'G'B' image *pSrc* to the Y'Cb'Cr' image *pDst* according to the same formulas as the function `ippRGBToYCbCr` does. The difference is that `ippRGBToYCrCb422` uses 4:2:2 sampling format for the converted image (see [Table "Pixel-Order Image Formats"](#) and [Table "Planar Image Formats"](#) for more details).

The converted buffer has the reduced bit depth of 16 bits per pixel, whereas the source buffer has 24 bit depth.

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pSrc</i> or <i>pDst</i> is NULL.

`ippStsSizeErr` Indicates an error condition if `roiSize` has a field with a zero or negative value.

## YCrCb422ToRGB

*Converts 16-bit per pixel YCrCb image to 24-bit per pixel RGB image.*

---

### Syntax

#### Case 1: Operation on pixel-order data

```
IppStatus ippYCrCb422ToRGB_8u_C2C3R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst, int dstStep, IppiSize roiSize);
```

#### Case 2: Conversion from pixel-order to planar data

```
IppStatus ippYCrCb422ToRGB_8u_C2P3R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst[3], int dstStep, IppiSize roiSize);
```

### Parameters

<code>pSrc</code>	Pointer to the source image ROI for pixel-order image. An array of pointers to ROI in each separate source planes for planar images.
<code>srcStep</code>	Distance in bytes between starts of consecutive lines in the source image.
<code>pDst</code>	Pointer to the ROI in the destination pixel-order image. An array of pointers to ROI in each planes of the destination planar image.
<code>dstStep</code>	Distance in bytes between starts of consecutive lines in the destination image.
<code>roiSize</code>	Size of the source and destination ROI in pixels.

### Description

The function `ippYCrCb422ToRGB` is declared in the `ippcc.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts the `Y'Cr'Cb'` image `pSrc`, packed in `4:2:2` [sampling](#) format (see [Table "Pixel-Order Image Formats"](#) and [Table "Planar Image Formats"](#) for more details) to the 24-bit gamma-corrected `R'G'B'` image `pDst` according to the same formulas as the function `ippYCbCrToRGB` does. The output `R'G'B'` values are saturated to the range `[0..255]`.

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <code>pSrc</code> or <code>pDst</code> is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <code>roiSize</code> has a field with a zero or negative value.

## BGRToYCbCr422

Converts 24-bit per pixel BGR image to 16-bit per pixel YCbCr image.

### Syntax

#### Case 1: Operation on pixel-order data

```
IppStatus ippiBGRToYCbCr422_8u_C3C2R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst, int dstStep, IppiSize roiSize);
```

```
IppStatus ippiBGRToYCbCr422_8u_AC4C2R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst, int dstStep, IppiSize roiSize);
```

#### Case 2: Conversion from pixel-order to planar data

```
IppStatus ippiBGRToYCbCr422_8u_C3P3R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);
```

```
IppStatus ippiBGRToYCbCr422_8u_AC4P3R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);
```

### Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the ROI in the destination pixel-order image. An array of pointers to ROI in each planes of the destination planar image.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination pixel-order image. An array of distances in bytes for each plane of the destination planar image.
<i>roiSize</i>	Size of the source and destination ROI in pixels.

### Description

The function `ippiBGRToYCbCr422` is declared in the `ippcc.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts a three- or four-channel gamma-corrected B'G'R' image *pSrc* to the two-channel or three-planes Y'Cb'Cr' image *pDst* according to the same formulas as the function `ippiRGBToYCbCr` does. The difference is that `ippiBGRToYCbCr422` uses the 4:2:2 sampling format (see [Table "Pixel-Order Image Formats"](#) and [Table "Planar Image Formats"](#) for more details).

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pSrc</i> or <i>pDst</i> is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize.width</i> is less than 2 or <i>roiSize.height</i> is less than 1.

## YCbCr422ToBGR

*Converts a YCbCr image with 4:2:2 sampling to the BGR image.*

---

### Syntax

#### Case 1: Operation on pixel-order data

```
IppStatus ippiYCbCr422ToBGR_8u_C2C3R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst, int dstStep, IppiSize roiSize);
```

```
IppStatus ippiYCbCr422ToBGR_8u_C2C4R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst, int dstStep, IppiSize roiSize, Ipp8u aval);
```

#### Case 2: Conversion from planar to pixel-order data

```
IppStatus ippiYCbCr422ToBGR_8u_P3C3R(const Ipp8u* pSrc[3], int srcStep[3], Ipp8u* pDst, int dstStep, IppiSize roiSize);
```

### Parameters

<i>pSrc</i>	Pointer to the source image ROI. An array of pointers to the ROI in each separate plane of the source planar image.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image. An array of such distances in bytes for each plane of the source planar image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>roiSize</i>	Size of the source and destination ROI in pixels.
<i>aval</i>	Constant value to create the fourth channel.

### Description

The function `ippiYCbCr422ToBGR` is declared in the `ippcc.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts the Y'Cb'Cr' image *pSrc* with 4:2:2 sampling (see [Table "Pixel-Order Image Formats"](#) and [Table "Planar Image Formats"](#) for more details) to the gamma-corrected B'G'R' image *pDst* according to the same formulas as the function `ippiYCbCrToRGB` does.

The output B'G'R' values are saturated to the range [0..255].

The fourth channel is created by setting channel values to the constant value *aval*.

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pSrc</i> or <i>pDst</i> is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize.width</i> is less than 2 or <i>roiSize.height</i> is less than 1.

## BGR565ToYCbCr422, BGR555ToYCbCr422

Converts 16-bit per pixel BGR image to 16-bit per pixel YCbCr image.

### Syntax

#### Case 1: Operation on pixel-order data

```
IppStatus ippiBGR565ToYCbCr422_16u8u_C3C2R(const Ipp16u* pSrc, int srcStep, Ipp8u* pDst,
int dstStep, IppiSize roiSize);
```

```
IppStatus ippiBGR555ToYCbCr422_16u8u_C3C2R(const Ipp16u* pSrc, int srcStep, Ipp8u* pDst,
int dstStep, IppiSize roiSize);
```

#### Case 2: Conversion from pixel-order to planar data

```
IppStatus ippiBGR565ToYCbCr422_16u8u_C3P3R(const Ipp16u* pSrc, int srcStep, Ipp8u* pDst[3],
int dstStep[3], IppiSize roiSize);
```

```
IppStatus ippiBGR555ToYCbCr422_16u8u_C3P3R(const Ipp16u* pSrc, int srcStep, Ipp8u* pDst[3],
int dstStep[3], IppiSize roiSize);
```

### Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the ROI in the destination pixel-order image. An array of pointers to ROI in each planes of the destination planar image.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination pixel-order image. An array of distances in bytes for each plane of the destination planar image.
<i>roiSize</i>	Size of the source and destination ROI in pixels.

### Description

The functions `ippiBGR565ToYCbCr422` and `ippiBGR555ToYCbCr422` are declared in the `ippcc.h` file. They operate with ROI (see [Regions of Interest in Intel IPP](#)).

These functions convert a three- or four-channel gamma-corrected B'G'R' image *pSrc* to the two-channel or three-planes Y'Cb'Cr' image *pDst* according to the same formulas as the function `ippiRGBToYCbCr` does. The difference is that these functions use 4:2:2 sampling format (see [Table "Pixel-Order Image Formats"](#) and [Table "Planar Image Formats"](#) for more details).

The source image *pSrc* is a packed 16-bit BGR image with reduced bit depth; all 3 channel intensities for a pixel are packed into two consecutive bytes (16u data type). It can be in one of two possible formats (see [Figure 16-bit Pixel Formats](#) for more details): BGR565 (5 bits for blue, 6 bits for green, 5 bits for red), and BGR555 (5 bits for blue, green, red).

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
--------------------------	--

<code>ippStsNullPtrErr</code>	Indicates an error condition if <code>pSrc</code> or <code>pDst</code> is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <code>roiSize.width</code> is less than 2 or <code>roiSize.height</code> is less than 1.

## RGBToCbYCr422, RGBToCbYCr422Gamma

Convert 24-bit per pixel RGB image to 16-bit per pixel CbYCr image.

---

### Syntax

```
IppStatus ippRGBToCbYCr422_8u_C3C2R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst, int dstStep, IppiSize roiSize);
```

```
IppStatus ippRGBToCbYCr422Gamma_8u_C3C2R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst, int dstStep, IppiSize roiSize);
```

### Parameters

<code>pSrc</code>	Pointer to the source image ROI.
<code>srcStep</code>	Distance in bytes between starts of consecutive lines in the source image.
<code>pDst</code>	Pointer to the destination image ROI.
<code>dstStep</code>	Distance in bytes between starts of consecutive lines in the destination image.
<code>roiSize</code>	Size of the source and destination ROI in pixels.

### Description

The functions `ippRGBToCbYCr422` and `ippRGBToCbYCr422Gamma` are declared in the `ippcc.h` file. They operate with ROI (see [Regions of Interest in Intel IPP](#)).

The function `ippRGBToCbYCr422` converts the gamma-corrected R'G'B' image `pSrc` to the Cb'Y'Cr' image `pDst` according to the same formulas as the function `ippRGBToYCbCr` does.

The function `ippRGBToCbYCr422Gamma` performs gamma-correction of the source RGB image `pSrc` according to the same formula as the function `ippGammaFwd` does, and then converts it to the Cb'Y'Cr' image `pDst` according to the same formulas as the function `ippRGBToYCbCr` does.

The functions `ippRGBToCbYCr422` and `ippRGBToCbYCr422Gamma` use 4:2:2 sampling format for the converted image.

A CbYCr image has the following sequence of bytes: Cb0Y0Cr0Y1, Cb1Y2Cr1Y3, ...

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <code>pSrc</code> or <code>pDst</code> is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <code>roiSize</code> has a field with a zero or negative value.



## CbYCr422ToRGB

*Converts 16-bit per pixel CbYCr image to 24-bit per pixel RGB image.*

---

### Syntax

```
IppStatus ippCbYCr422ToRGB_8u_C2C3R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst, int dstStep, IppiSize roiSize);
```

### Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>roiSize</i>	Size of the source and destination ROI in pixels

### Description

The function `ippCbYCr422ToRGB` is declared in the `ippcc.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts the `Cb'Y'Cr'` image *pSrc*, packed in the [4:2:2 sampling](#) format, to the 24-bit gamma-corrected `R'G'B'` image *pDst* according to the same formulas as the function `ippiYCbCrToRGB` does.

A CbYCr image has the following sequence of bytes: `Cb0Y0Cr0Y1, Cb1Y2Cr1Y3, ...`.

The output `R'G'B'` values are saturated to the range `[0..255]`.

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pSrc</i> or <i>pDst</i> is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with a zero or negative value.

## BGRToCbYCr422

*Converts 32-bit per pixel BGR image to 16-bit per pixel CbYCr image.*

---

### Syntax

```
IppStatus ippBGRToCbYCr422_8u_AC4C2R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst, int dstStep, IppiSize roiSize);
```

### Parameters

<i>pSrc</i>	Pointer to the source image ROI.
-------------	----------------------------------

<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>roiSize</i>	Size of the source and destination ROI in pixels.

## Description

The function `ippiBGRTToCbYCr422` is declared in the `ippcc.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts the four-channel gamma-corrected B'G'R' image *pSrc* to the two-channel Cb'Y'Cr' image *pDst* according to the same formulas as the function `ippiRGBToYCbCr` does. The function `ippiBGRTToCbYCr422` uses 4:2:2 sampling format for the converted image. An alpha-channel information is lost.

An CbYCr image has the following sequence of bytes: Cb0Y0Cr0Y1, Cb1Y2Cr1Y3, ... .

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pSrc</i> or <i>pDst</i> is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with a zero or negative value.

## BGRTToCbYCr422\_709HDTV

*Converts BGR image to 16-bit per pixel CbYCr image for ITU-R BT.709 HDTV signal.*

---

## Syntax

```
IppStatus ippiBGRTToCbYCr422_709HDTV_8u_C3C2R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst,
int dstStep, IppiSize roiSize);
```

```
IppStatus ippiBGRTToCbYCr422_709HDTV_8u_AC4C2R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst,
int dstStep, IppiSize roiSize);
```

## Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>roiSize</i>	Size of the source and destination ROI in pixels.

## Description

The function `ippiBGRTToCbYCr422_709HDTV` is declared in the `ippcc.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts the three- or four-channel gamma-corrected B'G'R' image *pSrc* to the two-channel Cb'Y'Cr' image *pDst* for digital component video signals complied with the ITU-R BT.709 Recommendation [ITU709] for high-definition TV (HDTV). The source image pixel values are in the range [16..235]. The conversion is performed according to the following formulas [Jack01]:

$$Y' = 0.213 * R' + 0.715 * G' + 0.072 * B'$$

$$Cb' = -0.117 * R' - 0.394 * G' + 0.511 * B' + 128$$

$$Cr' = 0.511 * R' - 0.464 * G' - 0.047 * B' + 128$$

The values of *Y'* of the destination image are in the range [16..235], the values of *Cb'*, *Cr'* are in the range [16..240]. They should be saturated at the 1 and 254 levels.

The function `ippiBGRTToCbYCr422_709HDTV` uses the 4:2:2 sampling format for the converted image. The alpha-channel information is lost.

A CbYCr image has the following sequence of bytes: Cb0Y0Cr0Y1, Cb1Y2Cr1Y3, ... .

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pSrc</i> or <i>pDst</i> pointer is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with a zero or negative value.

## CbYCr422ToBGR

*Converts 16-bit per pixel CbYCr image to four channel BGR image.*

---

## Syntax

```
IppStatus ippiCbYCr422ToBGR_8u_C2C4R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst, int dstStep, IppiSize roiSize, Ipp8u aval);
```

## Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>roiSize</i>	Size of the source and destination ROI in pixels.
<i>aval</i>	Constant value to create the fourth channel.

## Description

The function `ippiCbYCr422ToBGR` is declared in the `ippcc.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts the `Cb'Y'Cr'` image `pSrc`, packed in `4:2:2` [sampling](#) format, to the four channel gamma-corrected `B'G'R'` image `pDst` according to the same formulas as the function `ippiYCbCrToRGB` does.

A `CbYCr` image has the following sequence of bytes: `Cb0Y0Cr0Y1, Cb1Y2Cr1Y3, ...`.

The output `B'G'R'` values are saturated to the range `[0..255]`.

The fourth channel is created by setting channel values to the constant value `aval`.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <code>pSrc</code> or <code>pDst</code> is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <code>roiSize</code> has a field with a zero or negative value.

## CbYCr422ToBGR\_709HDTV

*Converts 16-bit per pixel CbYCr image to the BGR image for ITU-R BT.709 HDTV signal.*

---

## Syntax

```
ippStatus ippiCbYCr422ToBGR_709HDTV_8u_C2C3R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst,
int dstStep, IppiSize roiSize);
```

```
ippStatus ippiCbYCr422ToBGR_709HDTV_8u_C2C4R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst,
int dstStep, IppiSize roiSize, Ipp8u aval);
```

## Parameters

<code>pSrc</code>	Pointer to the source image ROI.
<code>srcStep</code>	Distance in bytes between starts of consecutive lines in the source image.
<code>pDst</code>	Pointer to the destination image ROI.
<code>dstStep</code>	Distance in bytes between starts of consecutive lines in the destination image.
<code>roiSize</code>	Size of the source and destination ROI in pixels.
<code>aval</code>	Constant value to create the fourth channel.

## Description

The function `ippiCbYCr422ToBGR_709HDTV` is declared in the `ippcc.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts the `Cb'Y'Cr'` image `pSrc`, packed in `4:2:2` [sampling](#) format, to the three- or four-channel gamma-corrected `B'G'R'` image `pDst` for digital component video signals complied with the ITU-R BT.709 Recommendation [\[ITU709\]](#) for high-definition TV (HDTV). A source `CbYCr` image has the following sequence of bytes: `Cb0Y0Cr0Y1, Cb1Y2Cr1Y3, ...`. The values of `Y'` are in the range `[16..235]`, the values of `Cb'`, `Cr'` are in the range `[16..240]`. The conversion is performed according to the following formulas [\[Jack01\]](#):

$$R' = Y' + 1.540 * (Cr' - 128)$$

$$G' = Y' - 0.459 * (Cr' - 128) - 0.183 * (Cb' - 128)$$

$$B' = Y' + 1.816 * (Cb' - 128)$$

The destination image pixel values have a nominal range `[16..235]`. The resulting `R'G'B'` values should be saturated at the 0 and 255 levels.

The output `B'G'R'` values are saturated to the range `[0..255]`.

The fourth channel is created by setting channel values to the constant value `aval`.

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <code>pSrc</code> or <code>pDst</code> is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <code>roiSize</code> has a field with a zero or negative value.

## YCbCr422ToRGB565, YCbCr422ToRGB555, YCbCr422ToRGB444

*Convert 16-bit per pixel YCbCr image to 16-bit per pixel RGB image.*

### Syntax

#### Case 1: Operation on pixel-order data

```
IppStatus ippYCbCr422ToRGB565_8u16u_C2C3R(const Ipp8u* pSrc, int srcStep, Ipp16u* pDst,
int dstStep, IppiSize roiSize);
```

```
IppStatus ippYCbCr422ToRGB555_8u16u_C2C3R(const Ipp8u* pSrc, int srcStep, Ipp16u* pDst,
int dstStep, IppiSize roiSize);
```

```
IppStatus ippYCbCr422ToRGB444_8u16u_C2C3R(const Ipp8u* pSrc, int srcStep, Ipp16u* pDst,
int dstStep, IppiSize roiSize);
```

#### Case 2: Conversion from planar to pixel-order data

```
IppStatus ippYCbCr422ToRGB565_8u16u_P3C3R(const Ipp8u* pSrc[3], int srcStep[3], Ipp16u*
pDst, int dstStep, IppiSize roiSize);
```

```
IppStatus ippYCbCr422ToRGB555_8u16u_P3C3R(const Ipp8u* pSrc[3], int srcStep[3], Ipp16u*
pDst, int dstStep, IppiSize roiSize);
```

```
IppStatus ippYCbCr422ToRGB444_8u16u_P3C3R(const Ipp8u* pSrc[3], int srcStep[3], Ipp16u*
pDst, int dstStep, IppiSize roiSize);
```

## Parameters

<i>pSrc</i>	Pointer to the source image ROI for pixel-order images. An array of pointers to ROI in each separate source planes for planar images.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image. An array of three values in case of planar data.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>roiSize</i>	Size of the source and destination ROI in pixels.

## Description

The functions `ippiYCbCr422ToRGB565`, `ippiYCbCr422ToRGB565`, `ippiYCbCr422ToRGB565` are declared in the `ippcc.h` file. They operate with ROI (see [Regions of Interest in Intel IPP](#)).

These functions convert the `Y'Cb'Cr'` image *pSrc*, packed in `4:2:2` [sampling](#) format (see [Table "Pixel-Order Image Formats"](#) for more details), to the gamma-corrected `R'G'B'` image *pDst* according to the same formulas as the function `ippiYCbCrToRGB` does.

The destination image *pDst* is a packed 16-bit RGB image with reduced bit depth; all 3 channel intensities for a pixel are packed into two consecutive bytes (`16u` data type). It can be in one of three possible formats (see [Figure 16-bit Pixel Formats](#)): `RGB565` (5 bits for red, 6 bits for green, 5 bits for blue), `RGB555` (5 bits for red, green, blue), `RGB444` (4 bits for red, green, blue).

Bit reduction is performed by discarding the least significant bits in the image after color conversion.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pSrc</i> or <i>pDst</i> is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with a zero or negative value.

## YCbCr422ToRGB565Dither, YCbCr422ToRGB555Dither, YCbCr422ToRGB444Dither

*Convert 16-bit per pixel YCbCr image to 16-bit per pixel RGB image with dithering.*

---

## Syntax

### Case 1: Operation on pixel-order data

```
ippStatus ippiYCbCr422ToRGB565Dither_8u16u_C2C3R(const Ipp8u* pSrc, int srcStep, Ipp16u* pDst, int dstStep, IppiSize roiSize);
```

```
ippStatus ippiYCbCr422ToRGB555Dither_8u16u_C2C3R(const Ipp8u* pSrc, int srcStep, Ipp16u* pDst, int dstStep, IppiSize roiSize);
```

```
ippStatus ippiYCbCr422ToRGB444Dither_8u16u_C2C3R(const Ipp8u* pSrc, int srcStep, Ipp16u* pDst, int dstStep, IppiSize roiSize);
```

**Case 2: Conversion from planar to pixel-order data**

```

IppStatus ippiYCbCr422ToRGB565Dither_8u16u_P3C3R(const Ipp8u* pSrc[3], int srcStep[3],
Ipp16u* pDst, int dstStep, IppiSize roiSize);

IppStatus ippiYCbCr422ToRGB555Dither_8u16u_P3C3R(const Ipp8u* pSrc[3], int srcStep[3],
Ipp16u* pDst, int dstStep, IppiSize roiSize);

IppStatus ippiYCbCr422ToRGB444Dither_8u16u_P3C3R(const Ipp8u* pSrc[3], int srcStep[3],
Ipp16u* pDst, int dstStep, IppiSize roiSize);

```

**Parameters**

<i>pSrc</i>	Pointer to the source image ROI for pixel-order images. An array of pointers to ROI in each separate source planes for planar images.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image. An array of three values in case of planar data.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>roiSize</i>	Size of the source and destination ROI in pixels.

**Description**

The functions `ippiYCbCr422ToRGB565Dither`, `ippiYCbCr422ToRGB565Dither`, `ippiYCbCr422ToRGB565Dither` are declared in the `ippcc.h` file. They operate with ROI (see [Regions of Interest in Intel IPP](#)).

These functions convert the `Y'Cb'Cr'` image *pSrc*, packed in `4:2:2` [sampling](#) format (see [Table "Pixel-Order Image Formats"](#) for more details), to the gamma-corrected `R'G'B'` image *pDst* according to the same formulas as the function `ippiYCbCrToRGB` does.

The destination image *pDst* is a packed 16-bit RGB image with reduced bit depth; all 3 channel intensities for a pixel are packed into two consecutive bytes (`16u` data type). It can be in one of three possible formats (see [Figure 16-bit Pixel Formats](#)): RGB565 (5 bits for red, 6 bits for green, 5 bits for blue), RGB555 (5 bits for red, green, blue), RGB444 (4 bits for red, green, blue).

Bit reduction is performed using the Bayer's threshold dithering algorithm with a 4x4 matrix [\[Tho91\]](#).

**Return Values**

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pSrc</i> or <i>pDst</i> is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with a zero or negative value.

## YCbCr422ToBGR565, YCbCr422ToBGR555, YCbCr422ToBGR444

Convert 16-bit per pixel YCbCr image to 16-bit per pixel BGR image.

---

### Syntax

#### Case 1: Operation on pixel-order data

```
IppStatus ippiYCbCr422ToBGR565_8u16u_C2C3R(const Ipp8u* pSrc, int srcStep, Ipp16u* pDst,
int dstStep, IppiSize roiSize);
```

```
IppStatus ippiYCbCr422ToBGR555_8u16u_C2C3R(const Ipp8u* pSrc, int srcStep, Ipp16u* pDst,
int dstStep, IppiSize roiSize);
```

```
IppStatus ippiYCbCr422ToBGR444_8u16u_C2C3R(const Ipp8u* pSrc, int srcStep, Ipp16u* pDst,
int dstStep, IppiSize roiSize);
```

#### Case 2: Conversion from planar to pixel-order data

```
IppStatus ippiYCbCr422ToBGR565_8u16u_P3C3R(const Ipp8u* pSrc[3], int srcStep[3], Ipp16u*
pDst, int dstStep, IppiSize roiSize);
```

```
IppStatus ippiYCbCr422ToBGR555_8u16u_P3C3R(const Ipp8u* pSrc[3], int srcStep[3], Ipp16u*
pDst, int dstStep, IppiSize roiSize);
```

```
IppStatus ippiYCbCr422ToBGR444_8u16u_P3C3R(const Ipp8u* pSrc[3], int srcStep[3], Ipp16u*
pDst, int dstStep, IppiSize roiSize);
```

### Parameters

<i>pSrc</i>	Pointer to the source image ROI for pixel-order images. An array of pointers to ROI in each separate source planes for planar images.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image. An array of three values in case of planar data.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>roiSize</i>	Size of the source and destination ROI in pixels.

### Description

The functions `ippiYCbCr422ToBGR565`, `ippiYCbCr422ToBGR555`, `ippiYCbCr422ToBGR565` are declared in the `ippcc.h` file. They operate with ROI (see [Regions of Interest in Intel IPP](#)).

These functions convert the 'YCbCr' image *pSrc*, packed in 4:2:2 sampling format (see [Table "Pixel-Order Image Formats"](#) for more details), to the gamma-corrected 'BGR' image *pDst* according to the same formulas as the function `ippiYCbCrToRGB` does.

The destination image *pDst* is a packed 16-bit BGR image with reduced bit depth; all 3 channel intensities for a pixel are packed into two consecutive bytes (16u data type). It can be in one of three possible formats (see [Figure 16-bit Pixel Formats](#)): BGR565 (5 bits for blue, 6 bits for green, 5 bits for red), BGR555 (5 bits for blue, green, red), BGR444 (4 bits for blue, green, red).



Bit reduction is performed by discarding the least significant bits in the image after color conversion.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <code>pSrc</code> or <code>pDst</code> is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <code>roiSize</code> has a field with a zero or negative value.

## YCbCr422ToBGR565Dither, YCbCr422ToBGR555Dither, YCbCr422ToBGR444Dither

*Convert 16-bit per pixel YCbCr image to 16-bit per pixel BGR image with dithering.*

## Syntax

### Case 1: Operation on pixel-order data

```
IppStatus ippYCbCr422ToBGR565Dither_8u16u_C2C3R(const Ipp8u* pSrc, int srcStep, Ipp16u* pDst, int dstStep, IppiSize roiSize);
```

```
IppStatus ippYCbCr422ToBGR555Dither_8u16u_C2C3R(const Ipp8u* pSrc, int srcStep, Ipp16u* pDst, int dstStep, IppiSize roiSize);
```

```
IppStatus ippYCbCr422ToBGR444Dither_8u16u_C2C3R(const Ipp8u* pSrc, int srcStep, Ipp16u* pDst, int dstStep, IppiSize roiSize);
```

### Case 2: Conversion from planar to pixel-order data

```
IppStatus ippYCbCr422ToBGR565Dither_8u16u_P3C3R(const Ipp8u* pSrc[3], int srcStep[3], Ipp16u* pDst, int dstStep, IppiSize roiSize);
```

```
IppStatus ippYCbCr422ToBGR555Dither_8u16u_P3C3R(const Ipp8u* pSrc[3], int srcStep[3], Ipp16u* pDst, int dstStep, IppiSize roiSize);
```

```
IppStatus ippYCbCr422ToBGR444Dither_8u16u_P3C3R(const Ipp8u* pSrc[3], int srcStep[3], Ipp16u* pDst, int dstStep, IppiSize roiSize);
```

## Parameters

<code>pSrc</code>	Pointer to the source image ROI for pixel-order images. An array of pointers to ROI in each separate source planes for planar images.
<code>srcStep</code>	Distance in bytes between starts of consecutive lines in the source image. An array of three values in case of planar data.
<code>pDst</code>	Pointer to the destination image ROI.
<code>dstStep</code>	Distance in bytes between starts of consecutive lines in the destination image.
<code>roiSize</code>	Size of the source and destination ROI in pixels.

## Description

The functions `ippiYCbCr422ToBGR565Dither`, `ippiYCbCr422ToBGR565Dither`, `ippiYCbCr422ToBGR565Dither` are declared in the `ippcc.h` file. They operate with ROI (see [Regions of Interest in Intel IPP](#)).

These functions convert the `Y'Cb'Cr'` image `pSrc`, packed in the [4:2:2 sampling](#) format (see [Table "Pixel-Order Image Formats"](#) for more details), to the gamma-corrected `B'G'R'` image `pDst` according to the same formulas as the function `ippiYCbCrToRGB` does.

The destination image `pDst` is a packed 16-bit BGR image with reduced bit depth; all 3 channel intensities for a pixel are packed into two consecutive bytes (`16u` data type). It can be in one of three possible formats (see [Figure 16-bit Pixel Formats](#)): `BGR565` (5 bits for blue, 6 bits for green, 5 bits for red), `BGR555` (5 bits for blue, green, red), `BGR444` (4 bits for blue, green, red).

Bit reduction is performed using the Bayer's threshold dithering algorithm with a 4x4 matrix [\[Tho91\]](#).

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <code>pSrc</code> or <code>pDst</code> is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <code>roiSize</code> has a field with a zero or negative value.

## RGBToYCbCr420

*Converts an RGB image to the YCbCr color model; uses 4:2:0 sampling.*

---

## Syntax

```

IppStatus ippiRGBToYCbCr420_8u_C3P3R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst[3], int
dstStep[3], IppiSize roiSize);

IppStatus ippiRGBToYCbCr420_8u_C3P2R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDstY, int
dstYStep, Ipp8u* pDstCbCr, int dstCbCrStep, IppiSize roiSize);

IppStatus ippiRGBToYCbCr420_8u_C4P2R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDstY, int
dstYStep, Ipp8u* pDstCbCr, int dstCbCrStep, IppiSize roiSize);

```

## Parameters

<code>pSrc</code>	Pointer to the source image ROI.
<code>srcStep</code>	Distance in bytes between starts of consecutive lines in the source image.
<code>pDst</code>	An array of pointers to ROI in the separate planes of the destination image for three-plane image.
<code>dstStep</code>	Array of distances in bytes between starts of consecutive lines in each plane of the destination image for three-plane image.
<code>pDstY</code>	Pointer to ROI in the luminance plane of the destination image for two-plane image.

<code>dstStep</code>	Distance in bytes between starts of consecutive lines in the luminance plane for two-plane image.
<code>pDstCbCr</code>	Pointer to ROI in the interleaved chrominance plane of the destination image for two-plane image.
<code>dstCbCrStep</code>	Distance in bytes between starts of consecutive lines in the interleaved chrominance plane for two-plane image.
<code>roiSize</code>	Size of the source and destination ROI in pixels.

## Description

The function `ippiRGBToYCbCr420` is declared in the `ippcc.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts the gamma-corrected R'G'B' image `pSrc` to the Y'Cb'Cr' image according to the same formulas as the function `ippiRGBToYCbCr` does. The difference is that `ippiRGBToYCbCr420` uses **4:2:0 sampling** format for the converted image (see [Table "Planar Image Formats"](#) for more details).

`roiSize.width` should be multiple of 2, and `roiSize.height` should be multiple of 2 (for three-plane image) or 4 (for two-plane image). If not the function reduces their original values to the nearest multiples of 2 or 4 correspondingly, performs operation, and returns warning message.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <code>pSrc</code> or <code>pDst</code> is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <code>roiSize</code> has a field with value less than 2.
<code>ippStsDoubleSize</code>	Indicates a warning if <code>roiSize.width</code> is not a multiple of 2, or if <code>roiSize.height</code> is not a multiple of 2 (for three-plane image) or 4 (for two-plane image).

## YCbCr420ToRGB

*Converts a YCbCr image that has 4:2:0 sampling format to the RGB color model.*

## Syntax

```
IppStatus ippiYCbCr420ToRGB_8u_P3C3R(const Ipp8u* const pSrc[3], int srcStep[3], Ipp8u* pDst, int dstStep, IppiSize roiSize);
```

```
IppStatus ippiYCbCr420ToRGB_8u_P2C3R(const Ipp8u* pSrcY, int srcYStep, const Ipp8u* pSrcCbCr, int srcCbCrStep, Ipp8u* pDst, int dstStep, IppiSize roiSize);
```

```
IppStatus ippiYCbCr420ToRGB_8u_P2C4R(const Ipp8u* pSrcY, int srcYStep, const Ipp8u* pSrcCbCr, int srcCbCrStep, Ipp8u* pDst, int dstStep, IppiSize roiSize);
```

## Parameters

<code>pSrc</code>	An array of pointers to ROI in separate planes of the three-plane source image.
-------------------	---

<i>srcStep</i>	An array of distances in bytes between starts of consecutive lines in each plane of the three-plane source image.
<i>pSrcY</i>	Pointer to ROI in the luminance plane of the two-plane source image.
<i>srcYStep</i>	Distance in bytes between starts of consecutive lines in the luminance plane of the two-plane source image.
<i>pSrcCbCr</i>	Pointer to ROI in the interleaved chrominance plane of the two-plane source image.
<i>srcCbCrStep</i>	Distance in bytes between starts of consecutive lines in the interleaved chrominance plane of the two-plane source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>roiSize</i>	Size of the source and destination ROI in pixels.

## Description

The function `ippiYCbCr420ToRGB` is declared in the `ippcc.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts the `Y'Cb'Cr'` image *pSrc* to the gamma-corrected `R'G'B'` image *pDst* according to the same formulas as the function `ippiYCbCrToRGB` does. The difference is that `ippiYCbCr420ToRGB` uses the input data in the `4:2:0` [sampling](#) format, in which the number of `Cb` and `Cr` samples is reduced by half in both vertical and horizontal directions (see [Table "Planar Image Formats"](#) for more details).

*roiSize.width* should be multiple of 2, and *roiSize.height* should be multiple of 2 (for three-plane image) or 4 (for two-plane image). If not the function reduces their original values to the nearest multiples of 2 or 4 correspondingly, performs operation, and returns warning message.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pSrc</i> or <i>pDst</i> is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with value less than 2.
<code>ippStsDoubleSize</code>	Indicates a warning if <i>roiSize.width</i> is not a multiple of 2, or if <i>roiSize.height</i> is not a multiple of 2 (for three-plane image) or 4 (for two-plane image).

## YCbCr420ToRGB565, YCbCr420ToRGB555, YCbCr420ToRGB444

*Convert a YCbCr image that has 4:2:0 sampling format to the 16-bit per pixel RGB image.*

---

## Syntax

```
ippStatus ippiYCbCr420ToRGB565_8u16u_P3C3R(const Ipp8u* pSrc[3], int srcStep[3], Ipp16u* pDst, int dstStep, IppiSize roiSize);
```

```
IppStatus ippiYCbCr420ToRGB555_8u16u_P3C3R(const Ipp8u* pSrc[3], int srcStep[3], Ipp16u* pDst, int dstStep, IppiSize roiSize);
```

```
IppStatus ippiYCbCr420ToRGB444_8u16u_P3C3R(const Ipp8u* pSrc[3], int srcStep[3], Ipp16u* pDst, int dstStep, IppiSize roiSize);
```

## Parameters

<i>pSrc</i>	An array of pointers to ROI in separate planes of the source image.
<i>srcStep</i>	An array of distances in bytes between starts of consecutive lines in the source image planes.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>roiSize</i>	Size of the source and destination ROI in pixels.

## Description

The functions `ippiYCbCr420ToRGB565`, `ippiYCbCr420ToRGB565`, `ippiYCbCr420ToRGB565` are declared in the `ippcc.h` file. They operate with ROI (see [Regions of Interest in Intel IPP](#)).

These functions convert the `Y'Cb'Cr'` image *pSrc*, packed in `4:2:0` [sampling](#) format (see [Table "Pixel-Order Image Formats"](#) for more details), to the gamma-corrected `R'G'B'` image *pDst* according to the same formulas as the function `ippiYCbCrToRGB` does.

The destination image *pDst* is a packed 16-bit RGB image with reduced bit depth; all 3 channel intensities for a pixel are packed into two consecutive bytes (16u data type). It can be in one of three possible formats (see [Figure 16-bit Pixel Formats](#)): RGB565 (5 bits for red, 6 bits for green, 5 bits for blue), RGB555 (5 bits for red, green, blue), RGB444 (4 bits for red, green, blue).

Bit reduction is performed by discarding the least significant bits in the image after color conversion.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pSrc</i> or <i>pDst</i> is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with a zero or negative value.

## YCbCr420ToRGB565Dither, YCbCr420ToRGB555Dither, YCbCr420ToRGB444Dither

Convert a YCbCr image that has 4:2:0 sampling format to the 16-bit per pixel RGB image with dithering.

## Syntax

```
IppStatus ippiYCbCr420ToRGB565Dither_8u16u_P3C3R(const Ipp8u* pSrc[3], int srcStep[3], Ipp16u* pDst, int dstStep, IppiSize roiSize);
```

```
IppStatus ippiYCbCr420ToRGB555Dither_8u16u_P3C3R(const Ipp8u* pSrc[3], int srcStep[3], Ipp16u* pDst, int dstStep, IppiSize roiSize);
```

```
ippStatus ippiYCbCr420ToRGB444Dither_8u16u_P3C3R(const Ipp8u* pSrc[3], int srcStep[3],
Ipp16u* pDst, int dstStep, IppiSize roiSize);
```

### Parameters

<i>pSrc</i>	An array of pointers to ROI in separate planes of the source image.
<i>srcStep</i>	An array of distances in bytes between starts of consecutive lines in the source image planes.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>roiSize</i>	Size of the source and destination ROI in pixels.

### Description

The functions `ippiYCbCr420ToRGB565Dither`, `ippiYCbCr420ToRGB555Dither`, `ippiYCbCr420ToRGB444Dither` are declared in the `ippcc.h` file. They operate with ROI (see [Regions of Interest in Intel IPP](#)).

These functions convert the `Y'Cb'Cr'` image *pSrc*, packed in the `4:2:0 sampling` format (see [Table "Pixel-Order Image Formats"](#) for more details), to the gamma-corrected `R'G'B'` image *pDst* according to the same formulas as the function `ippiYCbCrToRGB` does.

The destination image *pDst* is a packed 16-bit RGB image with reduced bit depth; all 3 channel intensities for a pixel are packed into two consecutive bytes (`16u` data type). It can be in one of three possible formats (see [Figure 16-bit Pixel Formats](#) for more details): `RGB565` (5 bits for red, 6 bits for green, 5 bits for blue), `RGB555` (5 bits for red, green, blue), or `RGB444` (4 bits for red, green, blue).

Bit reduction is performed using the Bayer's threshold dithering algorithm with a 4x4 matrix [\[Tho91\]](#).

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pSrc</i> or <i>pDst</i> is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with a zero or negative value.

## RGBToYCrCb420

*Converts an RGB image to the YCrCb image with 4:2:0 sampling format.*

---

### Syntax

```
ippStatus ippiRGBToYCrCb420_8u_AC4P3R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst[3], int
dstStep[3], IppiSize roiSize);
```

### Parameters

<i>pSrc</i>	Pointer to the source image ROI.
-------------	----------------------------------

<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Array of pointers to ROI in separate planes of the destination image.
<i>dstStep</i>	Array of distances in bytes between starts of consecutive lines in the destination image planes.
<i>roiSize</i>	Size of the source and destination ROI in pixels.

## Description

The function `ippiRGBToYCrCb420` is declared in the `ippcc.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts a four-channel gamma-corrected R'G'B' image *pSrc* to the planar Y'Cr'Cb' image *pDst* with the 4:2:0 sampling (see [Table "Planar Image Formats"](#) for more details). The conversion is performed according to the same formulas as the function `ippiRGBToYCbCr` does.

*roiSize.width* and *roiSize.height* should be multiples of 2. If not the function reduces their original values to the nearest multiples of 2, performs the operation, and returns a warning message.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pSrc</i> or <i>pDst</i> is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with a zero or negative value.
<code>ippStsDoubleSize</code>	Indicates a warning if <i>roiSize</i> has a field that is not a multiple of 2.

## YCrCb420ToRGB

*Converts a YCrCb image with the 4:2:0 sampling to the RGB image.*

## Syntax

```
IppStatus, ippiYCrCb420ToRGB_8u_P3C4R, (const Ipp8u* pSrc[3], int srcStep[3], Ipp8u* pDst,
int dstStep, IppiSize roiSize, Ipp8u aval);
```

## Parameters

<i>pSrc</i>	An array of pointers to ROI in separate planes of the source image.
<i>srcStep</i>	An array of distances in bytes between starts of consecutive lines in the source image planes.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>roiSize</i>	Size of the source and destination ROI in pixels.

*aval* Constant value to create fourth channel.

## Description

The function `ippiYCrCb420ToRGB` is declared in the `ippcc.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts the `Y'Cr'Cb'` image *pSrc* with the `4:2:0` sampling (see [Table "Planar Image Formats"](#) for more details) to the four-channel gamma-corrected `R'G'B'` image *pDst* according to the same formulas as the function `ippiYCbCrToRGB` does.

Fourth channel is created by setting channel values to the constant value *aval*.

*roiSize.width* and *roiSize.height* should be multiples of 2. If not the function reduces their original values to the nearest multiples of 2, performs the operation, and returns a warning message.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pSrc</i> or <i>pDst</i> is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with a zero or negative value.
<code>ippStsDoubleSize</code>	Indicates a warning if <i>roiSize</i> has a field that is not a multiple of 2.

## BGRToYCbCr420

*Converts a BGR image to the YCbCr image with 4:2:0 sampling format.*

---

## Syntax

```
ippStatus ippiBGRToYCbCr420_8u_C3P3R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);
```

```
ippStatus ippiBGRToYCbCr420_8u_AC4P3R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);
```

## Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	An array of pointers to ROI in separate planes of the destination image.
<i>dstStep</i>	An array of distances in bytes between starts of consecutive lines in the destination image planes.
<i>roiSize</i>	Size of the source and destination ROI in pixels.



## Description

The function `ippiBGRToYCbCr420` is declared in the `ippcc.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts a three- or four-channel gamma-corrected B'G'R' image `pSrc` to the planar Y'Cb'Cr' image `pDst` according to the same formulas as the function `ippiRGBToYCbCr` does. The difference is that `ippiBGRToYCbCr420` uses 4:2:0 sampling format (see [Table "Planar Image Formats"](#) for more details).

`roiSize.width` and `roiSize.height` should be multiples of 2. If not the function reduces their original values to the nearest multiples of 2, performs operation, and returns warning message.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <code>pSrc</code> or <code>pDst</code> is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <code>roiSize.width</code> is less than 2 or <code>roiSize.height</code> is less than 2.
<code>ippStsDoubleSize</code>	Indicates a warning if <code>roiSize</code> has a field that is not a multiple of 2.

## BGRToYCbCr420\_709CSC

*Converts a BGR image to the YCbCr image with 4:2:0 sampling for ITU-R BT.709 CSC signal.*

## Syntax

```
ippStatus ippiBGRToYCbCr420_709CSC_8u_C3P3R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst[3],
int dstStep[3], IppiSize roiSize);
```

```
ippStatus ippiBGRToYCbCr420_709CSC_8u_AC4P3R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst[3],
int dstStep[3], IppiSize roiSize);
```

## Parameters

<code>pSrc</code>	Pointer to the source image ROI.
<code>srcStep</code>	Distance in bytes between starts of consecutive lines in the source image.
<code>pDst</code>	An array of pointers to ROI in separate planes of the destination image.
<code>dstStep</code>	An array of distances in bytes between starts of consecutive lines in the destination image planes.
<code>roiSize</code>	Size of the source and destination ROI in pixels.

## Description

The function `ippiBGRToYCrCb420_709CSC` is declared in the `ippcc.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts a three- or four-channel gamma-corrected B'G'R' image *pSrc* to the planar Y'Cb'Cr' image *pDst* for digital component video signals complied with the ITU-R BT.709 Recommendation [ITU709] for computer systems consideration (CSC). The source image pixel values are in the range [0..255]. The conversion is performed according to the following formulas [Jack01]:

$$Y' = 0.183 * R' + 0.614 * G' + 0.062 * B' + 16$$

$$Cb' = -0.101 * R' - 0.338 * G' + 0.439 * B' + 128$$

$$Cr' = 0.439 * R' - 0.399 * G' - 0.040 * B' + 128$$

The destination image *pDst* has 4:2:0 sampling format (see Table “Planar Image Formats” for more details).

The values of *roiSize.width* and *roiSize.height* should be multiples of 2. Otherwise, the function reduces their original values to the nearest multiples of 2, performs operation, and returns a warning message.

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pSrc</i> or <i>pDst</i> is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize.width</i> is less than 2 or <i>roiSize.height</i> is less than 2.
<code>ippStsDoubleSize</code>	Indicates a warning if <i>roiSize</i> has a field that is not a multiple of 2.

## BGRToYCbCr420\_709HDTV

*Converts a BGR image to the YCbCr image with 4:2:0 sampling for ITU-R BT.709 HDTV signal.*

---

### Syntax

```
ippStatus ipp_iBGRToYCbCr420_709HDTV_8u_AC4P3R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);
```

### Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	An array of pointers to ROI in separate planes of the destination image.
<i>dstStep</i>	An array of distances in bytes between starts of consecutive lines in the destination image planes.
<i>roiSize</i>	Size of the source and destination ROI in pixels.

### Description

The function `ipp_iBGRToYCbCr420_709HDTV` is declared in the `ippcc.h` file. It operates with ROI (see Regions of Interest in Intel IPP).

This function converts a four-channel gamma-corrected B'G'R' image *pSrc* to the planar Y'Cb'Cr' image *pDst* for digital component video signals complied with the ITU-R BT.709 Recommendation [ITU709] for high-definition TV (HDTV). The source image pixel values are in the range [16..235]. The conversion is performed according to the following formulas [Jack01]:

$$Y' = 0.213 * R' + 0.715 * G' + 0.072 * B'$$

$$Cb' = -0.117 * R' - 0.394 * G' + 0.511 * B' + 128$$

$$Cr' = 0.511 * R' - 0.464 * G' - 0.047 * B' + 128$$

The values of Y' of the destination image are in the range [16..235], the values of Cb', Cr' are in the range [16..240]. They should be saturated at the 1 and 254 levels.

The destination image *pDst* has the 4:2:0 sampling format (see Table "Planar Image Formats" for more details).

The values of *roiSize.width* and *roiSize.height* should be multiples of 2. Otherwise, the function reduces their original values to the nearest multiples of 2, performs the operation, and returns a warning message.

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pSrc</i> or <i>pDst</i> is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize.width</i> is less than 2 or <i>roiSize.height</i> is less than 2.
<code>ippStsDoubleSize</code>	Indicates a warning if <i>roiSize</i> has a field that is not a multiple of 2.

## BGRToYCrCb420\_709CSC

Converts a BGR image to the YCrCb image with 4:2:0 sampling for ITU-R BT.709 CSC signal.

### Syntax

```
IppStatus ippBGRToYCrCb420_709CSC_8u_AC4P3R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst[3],
int dstStep[3], IppiSize roiSize);
```

```
IppStatus ippBGRToYCrCb420_709CSC_8u_C3P3R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst[3],
int dstStep[3], IppiSize roiSize);
```

```
IppStatus ippBGRToYCrCb420_709CSC_8u_C3P2R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDstY,
int dstYStep, Ipp8u* pDstCbCr, int dstCbCrStep, IppiSize roiSize);
```

### Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	An array of pointers to ROI in separate planes of the destination image.

<i>dstStep</i>	An array of distances in bytes between starts of consecutive lines in the destination image planes.
<i>pDstY</i>	Pointer to the ROI in the luminance plane for a two-plane destination image.
<i>dstYStep</i>	Distance in bytes between starts of consecutive lines in the luminance plane of a destination image.
<i>pDstCbCr</i>	Pointer to the ROI in the interleaved chrominance plane for a two-plane destination image.
<i>dstCbCrStep</i>	Distance in bytes between starts of consecutive lines in the chrominance plane of a destination image.
<i>roiSize</i>	Size of the source and destination ROI in pixels.

## Description

The function `ippiBGRToYCrCb420_709CSC` is declared in the `ippcc.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts a three- or four-channel gamma-corrected B'G'R' image *pSrc* to the planar Y'Cr'Cb' image *pDst* for digital component video signals complied with the ITU-R BT.709 Recommendation [\[ITU709\]](#) for computer systems consideration (CSC). The source image pixel values are in the range [0..255]. The conversion is performed according to the following formulas [\[Jack01\]](#):

$$Y' = 0.183 * R' + 0.614 * G' + 0.062 * B' + 16$$

$$Cb' = -0.101 * R' - 0.338 * G' + 0.439 * B' + 128$$

$$Cr' = 0.439 * R' - 0.399 * G' - 0.040 * B' + 128$$

The destination image *pDst* has 4:2:0 sampling format (see [Table "Planar Image Formats"](#) for more details).

The values of *roiSize.width* and *roiSize.height* should be multiples of 2. Otherwise, the function reduces their original values to the nearest multiples of 2, performs the operation, and returns a warning message.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pSrc</i> or <i>pDst</i> is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize.width</i> is less than 2 or <i>roiSize.height</i> is less than 2.
<code>ippStsDoubleSize</code>	Indicates a warning if <i>roiSize</i> has a field that is not a multiple of 2.

## YCbCr420ToBGR

*Converts a YCbCr image with the 4:2:0 sampling to the BGR image.*

---

## Syntax

```
ippStatus ippiYCbCr420ToBGR_8u_P3C3R(const Ipp8u* const pSrc[3], int srcStep[3], Ipp8u* pDst, int dstStep, IppiSize roiSize);
```

```
ippStatus ippiYCbCr420ToBGR_8u_P3C4R(const Ipp8u* pSrc[3], int srcStep[3], Ipp8u* pDst,
int dstStep, IppiSize roiSize, Ipp8u aval);
```

## Parameters

<i>pSrc</i>	An array of pointers to ROI in separate planes of the source image.
<i>srcStep</i>	An array of distances in bytes between starts of consecutive lines in the source image planes.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>roiSize</i>	Size of the source and destination ROI in pixels.
<i>aval</i>	Constant value to create fourth channel.

## Description

The function `ippiYCbCr420ToBGR` is declared in the `ippcc.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts the `Y'Cb'Cr'` image *pSrc* with the `4:2:0` sampling (see [Table “Planar Image Formats”](#) for more details) to the gamma-corrected three- or four-channel `B'G'R'` image *pDst*. The conversion is performed according to the same formulas as the function `ippiYCbCrToRGB` does.

Fourth channel is created by setting channel values to the constant value *aval*.

The values of *roiSize.width* and *roiSize.height* should be multiples of 2. If not the function reduces their original values to the nearest multiples of 2, performs operation, and returns warning message.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pSrc</i> or <i>pDst</i> is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with a zero or negative value.
<code>ippStsDoubleSize</code>	Indicates a warning if <i>roiSize</i> has a field that is not a multiple of 2.

## YCbCr420ToBGR\_709CSC

*Converts a YCbCr image with 4:2:0 sampling to the BGR image for ITU-R BT.709 CSC signal.*

---

## Syntax

```
ippStatus ippiYCbCr420ToBGR_709CSC_8u_P3C3R(const Ipp8u* pSrc[3], int srcStep[3], Ipp8u*
pDst, int dstStep, IppiSize roiSize);
```

## Parameters

<i>pSrc</i>	An array of pointers to ROI in separate planes of the source image.
<i>srcStep</i>	An array of distances in bytes between starts of consecutive lines in the source image planes.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>roiSize</i>	Size of the source and destination ROI in pixels.

## Description

The function `ippiYCbCr420ToBGR_709CSC` is declared in the `ippcc.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts a planar `Y'Cb'Cr'` image *pSrc* to the three-channel gamma-corrected `B'G'R'` image *pDst* for digital component video signals complied with the ITU-R BT.709 Recommendation [\[ITU709\]](#) for computer systems consideration (CSC). The conversion is performed according to the following formulas [\[Jack01\]](#):

$$R' = 1.164 \cdot (Y' - 16) + 1.793 \cdot (Cr' - 128)$$

$$G' = 1.164 \cdot (Y' - 16) - 0.534 \cdot (Cr' - 128) - 0.213 \cdot (Cb' - 128)$$

$$B' = 1.164 \cdot (Y' - 16) + 2.115 \cdot (Cb' - 128)$$

The output `R'G'B'` values are saturated to the range `[0..255]`. The source image *pDst* has the `4:2:0 sampling` format (see [Table "Planar Image Formats"](#) for more details).

The values of *roiSize.width* and *roiSize.height* should be multiples of 2. Otherwise, the function reduces their original values to the nearest multiples of 2, performs the operation, and returns a warning message.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pSrc</i> or <i>pDst</i> is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with a zero or negative value.
<code>ippStsDoubleSize</code>	Indicates a warning if <i>roiSize</i> has a field that is not a multiple of 2.

## YCbCr420ToBGR\_709HDTV

*Converts a YCbCr image with 4:2:0 sampling to the BGR image for ITU-R BT.709 HDTV signal.*

---

## Syntax

```
ippStatus ippiYCbCr420ToBGR_709HDTV_8u_P3C4R(const Ipp8u* pSrc[3], int srcStep[3], Ipp8u* pDst, int dstStep, IppiSize roiSize, Ipp8u aval);
```

## Parameters

<i>pSrc</i>	An array of pointers to ROI in separate planes of the source image.
<i>srcStep</i>	An array of distances in bytes between starts of consecutive lines in the source image planes.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>roiSize</i>	Size of the source and destination ROI in pixels.
<i>aval</i>	Constant value to create fourth channel.

## Description

The function `ippiYCbCr420ToBGR_709HDTV` is declared in the `ippcc.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts a planar  $Y'Cb'Cr'$  image *pSrc* to the four-channel gamma-corrected  $B'G'R'$  image *pDst* for digital component video signals complied with the ITU-R BT.709 Recommendation [\[ITU709\]](#) for high-definition TV (HDTV). The values of  $Y'$  are in the range [16..235], the values of  $Cb'$ ,  $Cr'$  are in the range [16..240]. The conversion is performed according to the following formulas [\[Jack01\]](#):

$$R' = Y' + 1.540 * (Cr' - 128)$$

$$G' = Y' - 0.459 * (Cr' - 128) - 0.183 * (Cb' - 128)$$

$$B' = Y' + 1.816 * (Cb' - 128)$$

The destination image pixel values have a nominal range [16..235]. The resulting  $R'G'B'$  values should be saturated at the 0 and 255 levels.

The source image *pSrc* has the `4:2:0 sampling` format (see [Table "Planar Image Formats"](#) for more details).

The values of *roiSize.width* and *roiSize.height* should be multiples of 2. Otherwise the function reduces their original values to the nearest multiples of 2, performs operation, and returns a warning message.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pSrc</i> or <i>pDst</i> is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with a zero or negative value.
<code>ippStsDoubleSize</code>	Indicates a warning if <i>roiSize</i> has a field that is not a multiple of 2.

## BGR565ToYCbCr420, BGR555ToYCbCr420

*Converts a 16-bit per pixel BGR image to the YCbCr image that has 4:2:0 sampling format.*

---

### Syntax

```
IppStatus ippiBGR565ToYCbCr420_16u8u_C3P3R(const Ipp16u* pSrc, int srcStep, Ipp8u* pDst[3],
int dstStep[3], IppiSize roiSize);
```

```
IppStatus ippiBGR555ToYCbCr420_16u8u_C3P3R(const Ipp16u* pSrc, int srcStep, Ipp8u* pDst[3],
int dstStep[3], IppiSize roiSize);
```

### Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	An array of pointers to ROI in separate planes of the destination image.
<i>dstStep</i>	An array of distances in bytes between starts of consecutive lines in the destination image planes.
<i>roiSize</i>	Size of the source and destination ROI in pixels.

### Description

The functions `ippiBGR565ToYCbCr420` and `ippiBGR555ToYCbCr420` are declared in the `ippcc.h` file. They operate with ROI (see [Regions of Interest in Intel IPP](#)).

These functions convert a three-channel gamma-corrected B'G'R' image *pSrc* to the planar Y'Cb'Cr' image *pDst* according to the same formulas as the function `ippiRGBToYCbCr` does. The destination image has the 4:2:0 sampling format (see [Table "Planar Image Formats"](#) for more details).

The source image *pSrc* is a packed 16-bit BGR image with reduced bit depth; all 3 channel intensities for a pixel are packed into two consecutive bytes (16u data type). It can be in one of two possible formats (see [Figure 16-bit Pixel Formats](#) for more details): BGR565 (5 bits for blue, 6 bits for green, 5 bits for red), and BGR555 (5 bits for blue, green, red).

*roiSize.width* and *roiSize.height* should be multiples of 2. If not the function reduces their original values to the nearest multiples of 2, performs the operation, and returns a warning message.

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pSrc</i> or <i>pDst</i> is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize.width</i> is less than 2 or <i>roiSize.height</i> is less than 2.
<code>ippStsDoubleSize</code>	Indicates a warning if <i>roiSize</i> has a field that is not a multiple of 2.



## YCbCr420ToBGR565, YCbCr420ToBGR555, YCbCr420ToBGR444

Convert a YCbCr image that has 4:2:0 sampling format to the 16-bit per pixel BGR image.

### Syntax

```
IppStatus ippiYCbCr420ToBGR565_8u16u_P3C3R(const Ipp8u* pSrc[3], int srcStep[3], Ipp16u* pDst, int dstStep, IppiSize roiSize);
```

```
IppStatus ippiYCbCr420ToBGR555_8u16u_P3C3R(const Ipp8u* pSrc[3], int srcStep[3], Ipp16u* pDst, int dstStep, IppiSize roiSize);
```

```
IppStatus ippiYCbCr420ToBGR444_8u16u_P3C3R(const Ipp8u* pSrc[3], int srcStep[3], Ipp16u* pDst, int dstStep, IppiSize roiSize);
```

### Parameters

<i>pSrc</i>	An array of pointers to ROI in separate planes of the source image.
<i>srcStep</i>	An array of distances in bytes between starts of consecutive lines in the source image planes.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>roiSize</i>	Size of the source and destination ROI in pixels.

### Description

The functions `ippiYCbCr420ToBGR565`, `ippiYCbCr420ToBGR555`, `ippiYCbCr420ToBGR565` are declared in the `ippcc.h` file. They operate with ROI (see [Regions of Interest in Intel IPP](#)).

These functions convert the Y'Cb'Cr' image *pSrc*, packed in the 4:2:0 sampling format (see [Table "Pixel-Order Image Formats"](#) for more details), to the gamma-corrected B'G'R' image *pDst* according to the same formulas as the function `ippiYCbCrToRGB` does.

The destination image *pDst* is a packed 16-bit BGR image with reduced bit depth; all 3 channel intensities for a pixel are packed into two consecutive bytes (16u data type). It can be in one of three possible formats (see [Figure 16-bit Pixel Formats](#)): BGR565 (5 bits for blue, 6 bits for green, 5 bits for red), BGR555 (5 bits for blue, green, red), BGR444 (4 bits for blue, green, red).

Bit reduction is performed by discarding the least significant bits in the image after color conversion.

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pSrc</i> or <i>pDst</i> is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with a zero or negative value.

## YCbCr420ToBGR565Dither, YCbCr420ToBGR555Dither, YCbCr420ToBGR444Dither

*Convert a YCbCr image that has 4:2:0 sampling format to the 16-bit per pixel BGR image with dithering.*

---

### Syntax

```
IppStatus ippiYCbCr420ToBGR565Dither_8u16u_P3C3R(const Ipp8u* pSrc[3], int srcStep[3], Ipp16u* pDst, int dstStep, IppiSize roiSize);
```

```
IppStatus ippiYCbCr420ToBGR555Dither_8u16u_P3C3R(const Ipp8u* pSrc[3], int srcStep[3], Ipp16u* pDst, int dstStep, IppiSize roiSize);
```

```
IppStatus ippiYCbCr420ToBGR444Dither_8u16u_P3C3R(const Ipp8u* pSrc[3], int srcStep[3], Ipp16u* pDst, int dstStep, IppiSize roiSize);
```

### Parameters

<i>pSrc</i>	An array of pointers to ROI in separate planes of the source image.
<i>srcStep</i>	An array of distances in bytes between starts of consecutive lines in the source image planes.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>roiSize</i>	Size of the source and destination ROI in pixels.

### Description

The functions `ippiYCbCr420ToBGR565Dither`, `ippiYCbCr420ToBGR555Dither`, `ippiYCbCr420ToBGR444Dither` are declared in the `ippcc.h` file. They operate with ROI (see [Regions of Interest in Intel IPP](#)).

These functions convert the `Y'Cb'Cr'` image *pSrc*, packed in 4:2:0 sampling format (see [Table "Pixel-Order Image Formats"](#) for more details), to the gamma-corrected `B'G'R'` image *pDst* according to the same formulas as the function `ippiYCbCrToRGB` does.

The destination image *pDst* is a packed 16-bit BGR image with reduced bit depth; all 3 channel intensities for a pixel are packed into two consecutive bytes (16u data type). It can be in one of three possible formats (see [Figure 16-bit Pixel Formats](#) for more details): BGR565 (5 bits for blue, 6 bits for green, 5 bits for red), BGR555 (5 bits for blue, green, red), BGR444 (4 bits for blue, green, red). Bit reduction is performed using the Bayer's threshold dithering algorithm with a 4x4 matrix [\[Tho91\]](#).

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pSrc</i> or <i>pDst</i> is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with a zero or negative value.

## BGRToYCrCb420

*Converts a BGR image to the YCrCb image with 4:2:0 sampling format.*

---

### Syntax

```
IppStatus ippkBGRToYCrCb420_8u_C3P3R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);

IppStatus ippkBGRToYCrCb420_8u_AC4P3R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);
```

### Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	An array of pointers to ROI in separate planes of the destination image.
<i>dstStep</i>	An array of distances in bytes between starts of consecutive lines in the destination image planes.
<i>roiSize</i>	Size of the source and destination ROI in pixels.

### Description

The function `ippkBGRToYCrCb420` is declared in the `ippcc.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts a three- or four-channel gamma-corrected B'G'R' image *pSrc* to the planar Y'Cr'Cb' image *pDst* according to the same formulas as the function `ippkRGBToYCbCr` does. The destination image *pDst* has the 4:2:0 sampling format and the following order of pointers: Y-plane, Cr-plane, Cb-plane (see [Table "Planar Image Formats"](#) for more details).

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pSrc</i> or <i>pDst</i> is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize.width</i> is less than 2 or <i>roiSize.height</i> is less than 2.

## BGR565ToYCrCb420, BGR555ToYCrCb420

*Converts a 16-bit per pixel BGR image to the YCrCb image with 4:2:0 sampling format.*

---

### Syntax

```
IppStatus ippkBGR565ToYCrCb420_16u8u_C3P3R(const Ipp16u* pSrc, int srcStep, Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);
```

```
IppStatus ippBGR555ToYCrCb420_16u8u_C3P3R(const Ipp16u* pSrc, int srcStep, Ipp8u* pDst[3],
int dstStep[3], IppiSize roiSize);
```

## Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	An array of pointers to ROI in separate planes of the destination image.
<i>dstStep</i>	An array of distances in bytes between starts of consecutive lines in the destination image planes.
<i>roiSize</i>	Size of the source and destination ROI in pixels.

## Description

The functions `ippBGR565ToYCrCb420` and `ippBGR555ToYCrCb420` are declared in the `ippcc.h` file. They operate with ROI (see [Regions of Interest in Intel IPP](#)).

These functions convert a three-channel gamma-corrected B'G'R' image *pSrc* to the planar Y'Cr'Cb' image *pDst* according to the same formulas as the function `ippiRGBToYCbCr` does. The destination image *pDst* has the 4:2:0 sampling format and the following order of pointers: Y-plane, Cr-plane, Cb-plane (see [Table "Planar Image Formats"](#) for more details).

The source image *pSrc* is a packed 16-bit BGR image with reduced bit depth; all 3 channel intensities for a pixel are packed into two consecutive bytes (16u data type). It can be in one of two possible formats (see [Figure 16-bit Pixel Formats](#) for more details): BGR565 (5 bits for blue, 6 bits for green, 5 bits for red), and BGR555 (5 bits for blue, green, red).

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pSrc</i> or <i>pDst</i> is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize.width</i> is less than 2 or <i>roiSize.height</i> is less than 2.

## BGRToYCbCr411

*Converts a BGR image to the YCbCr planar image that has a 4:1:1 sampling format.*

---

## Syntax

```
IppStatus ippBGRToYCbCr411_8u_C3P3R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst[3], int
dstStep[3], IppiSize roiSize);
```

```
IppStatus ippBGRToYCbCr411_8u_AC4P3R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst[3], int
dstStep[3], IppiSize roiSize);
```

## Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	An array of pointers to ROI in separate planes of the destination image.
<i>dstStep</i>	An array of distances in bytes between starts of consecutive lines in the destination image planes.
<i>roiSize</i>	Size of the source and destination ROI in pixels.

## Description

The function `ippiBGRToYCbCr411` is declared in the `ippcc.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts a three- or four-channel gamma-corrected B'G'R' image *pSrc* to the planar Y'Cb'Cr' image *pDst* according to the same formulas as the function `ippiRGBToYCbCr` does. The difference is that `ippiBGRToYCbCr411` uses the 4:1:1 sampling format (see [Table "Planar Image Formats"](#) for more details).

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pSrc</i> or <i>pDst</i> is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize.width</i> is less than 4 or <i>roiSize.height</i> is less than 1.

## YCbCr411ToBGR

*Converts a YCbCr image that has 4:1:1 sampling format to the RGB color model.*

## Syntax

```
ippStatus ippiYCbCr411ToBGR_8u_P3C3R(const Ipp8u* pSrc[3], int srcStep[3], Ipp8u* pDst, int dstStep, IppiSize roiSize);
```

```
ippStatus ippiYCbCr411ToBGR_8u_P3C4R(const Ipp8u* pSrc[3], int srcStep[3], Ipp8u* pDst, int dstStep, IppiSize roiSize, Ipp8u aval);
```

## Parameters

<i>pSrc</i>	An array of pointers to ROI in separate planes of the source image.
<i>srcStep</i>	An array of distances in bytes between starts of consecutive lines in the source image planes.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.

<i>roiSize</i>	Size of the source and destination ROI in pixels.
<i>aval</i>	Constant value to create fourth channel.

### Description

The function `ippiYCbCr411ToBGR` is declared in the `ippcc.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts the planar `Y'Cb'Cr'` image *pSrc* to the three- or four-channel image *pDst*. To compute gamma-corrected `R'G'B'` (`B'G'R'`) channel values the above formulas are used. The difference is that `ippiYCbCr411ToBGR` uses the input data in the `4:1:1 sampling` format (see [Table "Planar Image Formats"](#) for more details). Fourth channel is created by setting channel values to the constant value *aval*.

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pSrc</i> or <i>pDst</i> is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with a zero or negative value.

## BGR565ToYCbCr411, BGR555ToYCbCr411

*Converts a 16-bit per pixel BGR image to the YCbCr image that has 4:1:1 sampling format.*

---

### Syntax

```
ippStatus ippiBGR565ToYCbCr411_16u8u_C3P3R(const Ipp16u* pSrc, int srcStep, Ipp8u* pDst[3],
int dstStep[3], IppiSize roiSize);
```

```
ippStatus ippiBGR555ToYCbCr411_16u8u_C3P3R(const Ipp16u* pSrc, int srcStep, Ipp8u* pDst[3],
int dstStep[3], IppiSize roiSize);
```

### Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	An array of pointers to ROI in separate planes of the destination image.
<i>dstStep</i>	An array of distances in bytes between starts of consecutive lines in the destination image planes.
<i>roiSize</i>	Size of the source and destination ROI in pixels.

### Description

The functions `ippiBGR565ToYCbCr411` and `ippiBGR555ToYCbCr411` are declared in the `ippcc.h` file. They operate with ROI (see [Regions of Interest in Intel IPP](#)).

These functions convert a three-channel gamma-corrected B'G'R' image *pSrc* to the planar Y'Cb'Cr' image *pDst* according to the same formulas as the function `ippiRGBToYCbCr` does. The difference is that these functions use 4:1:1 sampling (see Table “Planar Image Formats” for more details).

The source image *pSrc* is a packed 16-bit BGR image with reduced bit depth; all 3 channel intensities for a pixel are packed into two consecutive bytes (16u data type). It can be in one of two possible formats (see Figure 16-bit Pixel Formats for more details): BGR565 (5 bits for blue, 6 bits for green, 5 bits for red), and BGR555 (5 bits for blue, green, red).

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pSrc</i> or <i>pDst</i> is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize.width</i> is less than 4 or <i>roiSize.height</i> is less than 1.

## YCbCr411ToBGR565, YCbCr411ToBGR555

*Convert a YCbCr image that has 4:1:1 sampling format to the 16-bit per pixel BGR image.*

### Syntax

```
ippStatus ippiYCbCr411ToBGR565_8u16u_P3C3R(const Ipp8u* pSrc[3], int srcStep[3], Ipp16u* pDst, int dstStep, IppiSize roiSize);
```

```
ippStatus ippiYCbCr411ToBGR555_8u16u_P3C3R(const Ipp8u* pSrc[3], int srcStep[3], Ipp16u* pDst, int dstStep, IppiSize roiSize);
```

### Parameters

<i>pSrc</i>	An array of pointers to ROI in separate planes of the source image.
<i>srcStep</i>	An array of distances in bytes between starts of consecutive lines in the source image planes.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>roiSize</i>	Size of the source and destination ROI in pixels.

### Description

The functions `ippiYCbCr411ToBGR565` and `ippiYCbCr411ToBGR555` are declared in the `ippcc.h` file. They operate with ROI (see Regions of Interest in Intel IPP).

These functions convert the planar Y'Cb'Cr' image *pSrc*, packed in 4:1:1 sampling (see Table “Planar Image Formats” for more details), to the gamma-corrected B'G'R' image *pDst* according to the same formulas as the function `ippiYCbCrToRGB` does.

The destination image *pDst* is a packed 16-bit BGR image with reduced bit depth; all 3 channel intensities for a pixel are packed into two consecutive bytes (16u data type). It can be in one of two possible formats (see [Figure 16-bit Pixel Formats](#) for more details): BGR565 (5 bits for blue, 6 bits for green, 5 bits for red) and BGR555 (5 bits for blue, green, red).

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pSrc</i> or <i>pDst</i> is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with a zero or negative value.

## RGBToXYZ

*Converts an RGB image to the XYZ color model.*

---

### Syntax

```
ippStatus ippiRGBToXYZ_mod(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pDst,
int dstStep, IppiSize roiSize);
```

Supported values for mod:

8u_C3R	16u_C3R	16s_C3R	32f_C3R
8u_AC4R	16u_AC4R	16s_AC4R	32f_AC4R

### Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>roiSize</i>	Size of the source and destination ROI in pixels.

### Description

The function `ippiRGBToXYZ` is declared in the `ippcc.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts the RGB image *pSrc* to the CIE XYZ image *pDst* according to the following basic equations:

$$X = 0.412453 * R + 0.35758 * G + 0.180423 * B$$

$$Y = 0.212671 * R + 0.71516 * G + 0.072169 * B$$

$$Z = 0.019334 * R + 0.119193 * G + 0.950227 * B$$

The equations above are given on the assumption that R, G, and B values are normalized to the range [0..1]. In case of the floating-point data type, the input RGB values must already be in the range [0..1]. For integer data types, normalization is done by the conversion function internally.



The computed XYZ values are saturated if they fall out of range [0..1].

In case of integer function flavors, these values are then scaled to the full range of the destination data type (see [Table “Image Data Types and Ranges”](#) in Chapter 2).

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <code>pSrc</code> or <code>pDst</code> is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <code>roiSize</code> has a field with a zero or negative value.

## XYZToRGB

*Converts an XYZ image to the RGB color model.*

### Syntax

```
ippStatus ippXYZToRGB_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pDst,
int dstStep, IppiSize roiSize);
```

Supported values for mod:

<code>8u_C3R</code>	<code>16u_C3R</code>	<code>16s_C3R</code>	<code>32f_C3R</code>
<code>8u_AC4R</code>	<code>16u_AC4R</code>	<code>16s_AC4R</code>	<code>32f_AC4R</code>

### Parameters

<code>pSrc</code>	Pointer to the source image ROI.
<code>srcStep</code>	Distance in bytes between starts of consecutive lines in the source image.
<code>pDst</code>	Pointer to the destination image ROI.
<code>dstStep</code>	Distance in bytes between starts of consecutive lines in the destination image.
<code>roiSize</code>	Size of the source and destination ROI in pixels.

### Description

The function `ippXYZToRGB` is declared in the `ippcc.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts the CIE XYZ image `pSrc` to the RGB image `pDst` according to the following basic equations:

$$R = 3.240479 * X - 1.53715 * Y - 0.498535 * Z$$

$$G = -0.969256 * X + 1.875991 * Y + 0.041556 * Z$$

$$B = 0.055648 * X - 0.204043 * Y + 1.057311 * Z$$

The equations above are given on the assumption that *X*, *Y*, and *Z* values are in the range [0..1]. In case of the floating-point data type, the input XYZ values must already be in the range [0..1]. For integer data types, normalization is done by the conversion function internally.

The computed RGB values are saturated if they fall out of range [0..1].

In case of integer function flavors, these values are then scaled to the full range of the destination data type (see [Table “Image Data Types and Ranges”](#) in Chapter 2).

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <code>pSrc</code> or <code>pDst</code> is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <code>roiSize</code> has a field with a zero or negative value.

## RGBToLUV

*Converts an RGB image to the LUV color model.*

---

### Syntax

```
ippStatus ippRGBToLUV_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pDst,
int dstStep, IppiSize roiSize);
```

Supported values for mod:

<code>8u_C3R</code>	<code>16u_C3R</code>	<code>16s_C3R</code>	<code>32f_C3R</code>
<code>8u_AC4R</code>	<code>16u_AC4R</code>	<code>16s_AC4R</code>	<code>32f_AC4R</code>

### Parameters

<code>pSrc</code>	Pointer to the source image ROI.
<code>srcStep</code>	Distance in bytes between starts of consecutive lines in the source image.
<code>pDst</code>	Pointer to the destination image ROI.
<code>dstStep</code>	Distance in bytes between starts of consecutive lines in the destination image.
<code>roiSize</code>	Size of the source and destination ROI in pixels.

### Description

The function `ippRGBToLUV` is declared in the `ippcc.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts the RGB image `pSrc` to the CIE LUV [CIE LUV](#) image `pDst` in two steps. First, the conversion is done into [CIE XYZ](#) format, using equations defined for the `ippRGBToXYZ` function. After that, conversion to LUV image is performed in accordance with the following equations:

$$L = 116. \cdot (Y/Y_n)^{1/3} - 16.$$

$$U = 13. \cdot L \cdot (u - u_n)$$

$$V = 13. \cdot L \cdot (v - v_n)$$

where

$$u = 4.*X / (X + 15.*Y + 3.*Z)$$

$$v = 9.*Y / (X + 15.*Y + 3.*Z)$$

$$u_n = 4.*x_n / ( -2.*x_n + 12.*y_n + 3. )$$

$$v_n = 9.*y_n / ( -2.*x_n + 12.*y_n + 3. )$$

Here  $x_n = 0.312713$ ,  $y_n = 0.329016$  are the CIE chromaticity coordinates of the D65 white point, and  $y_n = 1.0$  is the luminance of the D65 white point.

The computed values of the  $L$  component are in the range  $[0..100]$ ,  $U$  component in the range  $[-134..220]$ , and  $V$  component in the range  $[-140..122]$ .

The equations above are given on the assumption that  $R$ ,  $G$ , and  $B$  values are normalized to the range  $[0..1]$ . In case of the floating-point data type, the input  $RGB$  values must already be in the range  $[0..1]$ . For integer data types, normalization is done by the conversion function internally.

In case of  $8u$  data type, the computed  $L$ ,  $U$ , and  $V$  values are quantized and converted to fit in the range  $[0..IPP\_MAX\_8U]$  as follows:

$$L = L * IPP\_MAX\_8U / 100.$$

$$U = (U + 134.) * IPP\_MAX\_8U / 354.$$

$$V = (V + 140.) * IPP\_MAX\_8U / 262.$$

In case of  $16u$  data type, the computed  $L$ ,  $U$ , and  $V$  values are quantized and converted to fit in the range  $[0..IPP\_MAX\_16U]$  as follows:

$$L = L * IPP\_MAX\_16U / 100.$$

$$U = (U + 134.) * IPP\_MAX\_16U / 354.$$

$$V = (V + 140.) * IPP\_MAX\_16U / 262.$$

In case of  $16s$  data type, the computed  $L$ ,  $U$ , and  $V$  values are quantized and converted to fit in the range  $[IPP\_MIN\_16S..IPP\_MAX\_16S]$  as follows:

$$L = L * IPP\_MAX\_16U / 100. + IPP\_MIN\_16S$$

$$U = (U + 134.) * IPP\_MAX\_16U / 354. + IPP\_MIN\_16S$$

$$V = (V + 140.) * IPP\_MAX\_16U / 262. + IPP\_MIN\_16S$$

For  $32f$  data type, no further conversion is done and  $L$ ,  $U$ , and  $V$  components remain in the ranges  $[0..100]$ ,  $[-134..220]$ , and  $[-140..122]$ , respectively.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pSrc</i> or <i>pDst</i> is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with a zero or negative value.

## LUVToRGB

*Converts a LUV image to the RGB color model.*

---

### Syntax

```
IppStatus ippiLUVToRGB_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pDst,
int dstStep, IppiSize roiSize);
```

Supported values for mod:

8u_C3R	16u_C3R	16s_C3R	32f_C3R
8u_AC4R	16u_AC4R	16s_AC4R	32f_AC4R

### Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>roiSize</i>	Size of the source and destination ROI in pixels.

### Description

The function `ippiLUVToRGB` is declared in the `ippcc.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts the [CIE LUV](#) image *pSrc* to the RGB image *pDst* in two steps. First, the conversion is carried out into [CIE XYZ](#) format.

To accomplish it, LUV components are transformed back into their original range. This is done for different data types in the following way.

For 8u data type:

$$L = L * 100. / IPP\_MAX\_8U$$

$$U = (U * 354. / IPP\_MAX\_8U) - 134.$$

$$V = (V * 262. / IPP\_MAX\_8U) - 140.$$

For 16u data type:

$$L = L * 100. / IPP\_MAX\_16U$$

$$U = (U * 354. / IPP\_MAX\_16U) - 134.$$

$$V = (V * 262. / IPP\_MAX\_16U) - 140.$$

For 16s data type:

$$L = (L - IPP\_MIN\_16S) * 100. / IPP\_MAX\_16U$$

$$U = ((U - IPP\_MIN\_16S) * 354. / IPP\_MAX\_16U) - 134.$$

$V = ((V - \text{IPP\_MIN\_16S}) * 262. / \text{IPP\_MAX\_16U}) - 140.$

After that, conversion to XYZ format takes place as follows:

$Y = Y_n * ((L + 16.) / 116.)^{**3}.$

$X = -9.* Y * u / ((u - 4.)* v - u* v )$

$Z = (9.* Y - 15*v*Y - v*X) / 3. * v$

where

$u = U / (13.* L) + u_n$

$v = V / (13.* L) + v_n$

and

$u_n = 4.*x_n / (-2.*x_n + 12.*y_n + 3.)$

$v_n = 9.*y_n / (-2.*x_n + 12.*y_n + 3.)$

Here  $x_n = 0.312713$ ,  $y_n = 0.329016$  are the CIE chromaticity coordinates of the D65 white point, and  $y_n = 1.0$  is the luminance of the D65 white point.

After this intermediate conversion is done, the obtained XYZ image is then converted to the destination RGB format using equations defined for the [ippiXYZToRGB](#) function.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <code>pSrc</code> or <code>pDst</code> is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <code>roiSize</code> has a field with a zero or negative value.

## BGRToLab

*Converts a BGR image to the Lab color model.*

### Syntax

```
IppStatus ippiBGRToLab_8u_C3R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst, int dstStep, IppiSize roiSize);
```

```
IppStatus ippiBGRToLab_8u16u_C3R(const Ipp8u* pSrc, int srcStep, Ipp16u* pDst, int dstStep, IppiSize roiSize);
```

### Parameters

<code>pSrc</code>	Pointer to the source image ROI.
<code>srcStep</code>	Distance in bytes between starts of consecutive lines in the source image.
<code>pDst</code>	Pointer to the destination image ROI.
<code>dstStep</code>	Distance in bytes between starts of consecutive lines in the destination image.

*roiSize*

Size of the source and destination ROI in pixels.

## Description

The function `ippiBGRTToLab` is declared in the `ippcc.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts the BGR image *pSrc* to the CIE Lab image *pDst* in two steps. First, the conversion is done into CIE XYZ format, using equations defined for the function `ippiRGBToXYZ`. After that, conversion to the Lab image is performed in accordance with the following equations:

$$L = 116. \cdot (Y/Y_n)^{1/3} - 16 \text{ for } Y/Y_n > 0.008856$$

$$L = 903.3 \cdot (Y/Y_n)^{1/3} \text{ for } Y/Y_n \leq 0.008856$$

$$a = 500. \cdot [f(X/X_n) - f(Y/Y_n)]$$

$$b = 200. \cdot [f(Y/Y_n) - f(Z/Z_n)]$$

where

$$f(t) = t^{1/3} - 16 \text{ for } t > 0.008856$$

$$f(t) = 7.787 \cdot t + 16/116 \text{ for } t \leq 0.008856$$

Here  $Y_n = 1.0$ ,  $X_n = 0.950455$ ,  $Z_n = 1.088753$  for the D65 white point with the CIE chromaticity coordinates  $x_n = 0.312713$ ,  $y_n = 0.329016$ .

The equations above are given on the assumption that initial B, G, R values are normalized to the range [0..1]. The computed values of the L component are in the range [0..100], a and b component values are in the range [-128..127].

These values are quantized and scaled to the 8-bit range of 0 to 255 for `ippiBGRTToLab_8u_C3`:

$$L = L \cdot 255./100.$$

$$a = (a + 128.)$$

$$b = (a + 128.)$$

or to the 16-bit range of 0 to 65535 for `ippiBGRTToLab_8u16u_C3R`:

$$L = L \cdot 65535./100.$$

$$a = (a + 128.) \cdot 255$$

$$b = (a + 128.) \cdot 255$$

## Return Values

`ippStsNoErr`

Indicates no error. Any other value indicates an error.

`ippStsNullPtrErr`Indicates an error condition if *pSrc* or *pDst* is NULL.`ippStsSizeErr`Indicates an error condition if *roiSize* has a field with a zero or negative value.

## LabToBGR

*Converts a Lab image to the BGR color model.*

---

### Syntax

```
IppStatus ippILabToBGR_8u_C3R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst, int dstStep,
IppSize roiSize);

IppStatus ippILabToBGR_16u8u_C3R(const Ipp16u* pSrc, int srcStep, Ipp8u* pDst, int dstStep,
IppSize roiSize);
```

### Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>roiSize</i>	Size of the source and destination ROI in pixels.

### Description

The function `ippILabToBGR` is declared in the `ippcc.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts the [CIE Lab](#) image *pSrc* to the BGR image *pDst* in two steps. First, the conversion is carried out into [CIE XYZ](#) format.

To accomplish it, Lab components are transformed back into their original range. This is done for different data types in the following way.

For 8u data type:

$$L = L * 100. / 255.$$

$$a = a - 128.$$

$$b = b - 128.$$

For 16u data type:

$$L = L * 100. / 65535.$$

$$a = (a / 255. - 128.)$$

$$b = (b / 255.) - 128.)$$

After that, conversion to XYZ format takes place as follows:

$$Y = Y_n * P^3.$$

$$X = X_n * (P + a / 500.)^3.$$

$$Z = Z_n * (P - b / 200.)^3.$$

where

$P = (L + 16) / 116.$

After this intermediate conversion is done, the obtained XYZ image is then converted to the destination BGR format using equations defined for the [ippiXYZToRGB](#) function.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <code>pSrc</code> or <code>pDst</code> is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <code>roiSize</code> has a field with a zero or negative value.

## RGBToYCC

*Converts an RGB image to the YCC color model.*

---

### Syntax

```
ippiStatus ippiRGBToYCC_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pDst,
int dstStep, IppiSize roiSize);
```

Supported values for mod:

<code>8u_C3R</code>	<code>16u_C3R</code>	<code>16s_C3R</code>	<code>32f_C3R</code>
<code>8u_AC4R</code>	<code>16u_AC4R</code>	<code>16s_AC4R</code>	<code>32f_AC4R</code>

### Parameters

<code>pSrc</code>	Pointer to the source image ROI.
<code>srcStep</code>	Distance in bytes between starts of consecutive lines in the source image.
<code>pDst</code>	Pointer to the destination image ROI.
<code>dstStep</code>	Distance in bytes between starts of consecutive lines in the destination image.
<code>roiSize</code>	Size of the source and destination ROI in pixels.

### Description

The function `ippiRGBToYCC` is declared in the `ippcc.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts the gamma-corrected R'B'G' image `pSrc` to the `PhotoY'C'C'` image `pDst` according to the following basic equations:

$$Y' = 0.299 * R' + 0.587 * G' + 0.114 * B'$$

$$C1' = -0.299 * R' - 0.587 * G' + 0.886 * B' = B' - Y'$$

$$C2' = 0.701 * R' - 0.587 * G' - 0.114 * B' = R' - Y'$$

The equations above are given on the assumption that  $R'$ ,  $G'$ , and  $B'$  values are normalized to the range [0..1]. In case of the floating-point data type, the input R'G'B' values must already be in the range [0..1]. For integer data types, normalization is done by the conversion function internally.



The computed  $Y'$ ,  $C1'$ ,  $C2'$  values are then quantized and converted to fit in the range  $[0..1]$  as follows:

$$Y' = 1. / 1.402 * Y'$$

$$C1' = 111.4 / 255. * C1' + 156. / 255.$$

$$C2' = 135.64 / 255. * C2' + 137. / 255.$$

In case of integer function flavors, these values are then scaled to the full range of the destination data type (see [Table “Image Data Types and Ranges”](#) in Chapter 2).

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <code>pSrc</code> or <code>pDst</code> is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <code>roiSize</code> has a field with a zero or negative value.

## YCCToRGB

*Converts a YCC image to the RGB color model.*

### Syntax

```
IppStatus ippICYCCToRGB_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pDst,
int dstStep, IppiSize roiSize);
```

Supported values for mod:

8u_C3R	16u_C3R	16s_C3R	32f_C3R
8u_AC4R	16u_AC4R	16s_AC4R	32f_AC4R

### Parameters

<code>pSrc</code>	Pointer to the source image ROI.
<code>srcStep</code>	Distance in bytes between starts of consecutive lines in the source image.
<code>pDst</code>	Pointer to the destination image ROI.
<code>dstStep</code>	Distance in bytes between starts of consecutive lines in the destination image.
<code>roiSize</code>	Size of the source and destination ROI in pixels.

### Description

The function `ippICYCCToRGB` is declared in the `ippcc.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts the [PhotoY'C'C'](#) image `pSrc` to the [R'B'G'](#) image `pDst`. The function `ippICYCCToRGB` first restores normal luminance and chrominance data as:

$$Y' = 1.3584 * Y'$$

$$C1' = 2.2179 * (C1' - 156./255.)$$

$$C2' = 1.8215 * (C2' - 137./255.)$$

The equations above are given on the assumption that source  $Y$ ,  $C1$ , and  $C2$  values are normalized to the range  $[0..1]$ . In case of the floating-point data type, the input YCC values must already be in the range  $[0..1]$ . For integer data types, normalization is done by the conversion function internally.

After that, YCC data are transformed into RGB format according to the following basic equations:

$$R' = Y' + C2'$$

$$G' = Y' - 0.194 * C1' - 0.509 * C2'$$

$$B' = Y' + C1'$$

In case of integer function flavors, the computed  $R'B'G'$  values are then scaled to the full range of the destination data type (see [Table "Image Data Types and Ranges"](#) in Chapter 2).

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <code>pSrc</code> or <code>pDst</code> is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <code>roiSize</code> has a field with a zero or negative value.

## RGBToHLS

*Converts an RGB image to the HLS color model.*

---

### Syntax

```
ippStatus ippiRGBToHLS_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pDst,
int dstStep, IppiSize roiSize);
```

Supported values for `mod`:

8u_C3R	16u_C3R	16s_C3R	32f_C3R
8u_AC4R	16u_AC4R	16s_AC4R	32f_AC4R

### Parameters

<code>pSrc</code>	Pointer to the source image ROI.
<code>srcStep</code>	Distance in bytes between starts of consecutive lines in the source image.
<code>pDst</code>	Pointer to the destination image ROI.
<code>dstStep</code>	Distance in bytes between starts of consecutive lines in the destination image.
<code>roiSize</code>	Size of the source and destination ROI in pixels.

### Description

The function `ippiRGBToHLS` is declared in the `ippcc.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts the R'B'G' image *pSrc* to the HLS image *pDst*. For function flavors operating on the floating point data, source RGB values must be in the range [0..1].

The conversion algorithm from RGB to HLS can be represented in pseudocode as follows:

```
// Lightness:
M1 = max(R,G,B); M2 = min(R,G,B); L = (M1+M2)/2
// Saturation:
if M1 = M2 then // achromatics case
    S = 0
    H = 0
else // chromatics case
    if L <= 0.5 then
        S = (M1-M2) / (M1+M2)
    else
        S = (M1-M2) / (2-M1-M2)
// Hue:
Cr = (M1-R) / (M1-M2)
Cg = (M1-G) / (M1-M2)
Cb = (M1-B) / (M1-M2)
if R = M2 then H = Cb - Cg
if G = M2 then H = 2 + Cr - Cb
if B = M2 then H = 4 + Cg - Cr
H = 60*H
if H < 0 then H = H + 360
```

For floating point function flavors, the computed H, L, S values are scaled to the range [0..1]. In case of integer function flavors, these values are scaled to the full range of the destination data type ([Table "Image Data Types and Ranges"](#) in Chapter 2).

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pSrc</i> or <i>pDst</i> is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with a zero or negative value.

## HLSToRGB

*Converts an HLS image to the RGB color model.*

### Syntax

```
IppStatus ippHLSToRGB_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pDst,
int dstStep, IppiSize roiSize);
```

Supported values for mod:

8u_C3R	16u_C3R	16s_C3R	32f_C3R
8u_AC4R	16u_AC4R	16s_AC4R	32f_AC4R

### Parameters

*pSrc*                                      Pointer to the source image ROI.

<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>roiSize</i>	Size of the source and destination ROI in pixels.

## Description

The function `ippiHLSToRGB` is declared in the `ippcc.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts the [HLS](#) image *pSrc* to the R'B'G' image *pDst*. For function flavors operating on the floating point data, source HLS values must be in the range [0..1]. The conversion algorithm from HLS to RGB can be represented in pseudocode as follows:

```

if L <= 0.5 then
    M2 = L * (1 + S)
else
    M2 = L + S - L * S
M1 = 2 * L - M2
if S = 0 then
    R = G = B = L
else
    h = H + 120
    if h > 360 then
        h = h - 360
    if h < 60 then
        R = ( M1 + ( M2 - M1 ) * h / 60 )
    else if h < 180 then
        R = M2
    else if h < 240 then
        R = M1 + ( M2 - M1 ) * ( 240 - h ) / 60
    else
        R = M1
    h = H
    if h < 60 then
        G = ( M1 + ( M2 - M1 ) * h / 60 )
    else if h < 180 then
        G = M2
    else if h < 240 then
        G = M1 + ( M2 - M1 ) * ( 240 - h ) / 60
    else
        G = M1
    h = H - 120
    if h < 0 then
        h += 360
    if h < 60 then
        B = ( M1 + ( M2 - M1 ) * h / 60 )
    else if h < 180 then
        B = M2
    else if h < 240 then
        B = M1 + ( M2 - M1 ) * ( 240 - h ) / 60
    else
        B = M1

```

For floating point function flavors, the computed  $R'$ ,  $G'$ ,  $B'$  values are scaled to the range  $[0..1]$ . In case of integer function flavors, these values are scaled to the full range of the destination data type (see [Table “Image Data Types and Ranges”](#) in Chapter 2).

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <code>pSrc</code> or <code>pDst</code> is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <code>roiSize</code> has a field with a zero or negative value.

## BGRToHLS

*Converts a BGR image to the HLS color model.*

### Syntax

```

IppStatus ippIBGRToHLS_8u_AC4R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst, int dstStep,
IppiSize roiSize);

IppStatus ippIBGRToHLS_8u_C3P3R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst[3], int dstStep,
IppiSize roiSize);

IppStatus ippIBGRToHLS_8u_AC4P4R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst[4], int
dstStep, IppiSize roiSize);

IppStatus ippIBGRToHLS_8u_P3C3R(const Ipp8u* const pSrc[3], int srcStep, Ipp8u* pDst, int
dstStep, IppiSize roiSize);

IppStatus ippIBGRToHLS_8u_AP4C4R(const Ipp8u* const pSrc[4], int srcStep, Ipp8u* pDst,
int dstStep, IppiSize roiSize);

IppStatus ippIBGRToHLS_8u_P3R(const Ipp8u* const pSrc[3], int srcStep, Ipp8u* pDst[3],
int dstStep, IppiSize roiSize);

IppStatus ippIBGRToHLS_8u_AP4R(const Ipp8u* const pSrc[4], int srcStep, Ipp8u* pDst[4],
int dstStep, IppiSize roiSize);

```

### Parameters

<code>pSrc</code>	Pointer to the ROI in the pixel-order source image. An array of pointers to ROI in each plane in the planar source image.
<code>srcStep</code>	Distance in bytes between starts of consecutive lines in the source image.
<code>pDst</code>	Pointer to the ROI in the pixel-order destination image. An array of pointers to ROI in each plane in the planar destination image.
<code>dstStep</code>	Distance in bytes between starts of consecutive lines in the destination image.
<code>roiSize</code>	Size of the source and destination ROI in pixels.

## Description

The function `ippiBGRTToHLS` is declared in the `ippcc.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts the B'G'R' image `pSrc` to the [HLS](#) image `pDst` according to the same formula as the function `ippiRGBToHLS` does.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <code>pSrc</code> or <code>pDst</code> is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <code>roiSize</code> has a field with a zero or negative value.

## HLSToBGR

*Converts an HLS image to the RGB color model.*

---

## Syntax

```
ippStatus ippiHLSToBGR_8u_C3P3R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst[3], int dstStep,
IppiSize roiSize);
```

```
ippStatus ippiHLSToBGR_8u_AC4P4R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst[4], int
dstStep, IppiSize roiSize);
```

```
ippStatus ippiHLSToBGR_8u_AP4R(const Ipp8u* const pSrc[4], int srcStep, Ipp8u* pDst[4],
int dstStep, IppiSize roiSize);
```

```
ippStatus ippiHLSToBGR_8u_P3R(const Ipp8u* const pSrc[3], int srcStep, Ipp8u* pDst[3],
int dstStep, IppiSize roiSize);
```

```
ippStatus ippiHLSToBGR_8u_AP4C4R(const Ipp8u* const pSrc[4], int srcStep, Ipp8u* pDst,
int dstStep, IppiSize roiSize);
```

```
ippStatus ippiHLSToBGR_8u_P3C3R(const Ipp8u* const pSrc[3], int srcStep, Ipp8u* pDst, int
dstStep, IppiSize roiSize);
```

## Parameters

<code>pSrc</code>	Pointer to the ROI in the pixel-order source image. An array of pointers to ROI in each plane in the planar source image.
<code>srcStep</code>	Distance in bytes between starts of consecutive lines in the source image.
<code>pDst</code>	Pointer to the ROI in the pixel-order destination image. An array of pointers to ROI in each plane in the planar destination image.
<code>dstStep</code>	Distance in bytes between starts of consecutive lines in the destination image.
<code>roiSize</code>	Size of the source and destination ROI in pixels.

## Description

The function `ippiHLSToBGR` is declared in the `ippcc.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts the [HLS](#) image `pSrc` to the B'G'R' image `pDst` according to the same formula as the function `ippiHLSToRGB` does.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <code>pSrc</code> or <code>pDst</code> is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <code>roiSize</code> has a field with a zero or negative value.

## RGBToHSV

*Converts an RGB image to the HSV color model.*

### Syntax

```
ippStatus ippiRGBToHSV_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pDst,
int dstStep, IppiSize roiSize);
```

Supported values for `mod`:

<code>8u_C3R</code>	<code>16u_C3R</code>
<code>8u_AC4R</code>	<code>16u_AC4R</code>

## Parameters

<code>pSrc</code>	Pointer to the source image ROI.
<code>srcStep</code>	Distance in bytes between starts of consecutive lines in the source image.
<code>pDst</code>	Pointer to the destination image ROI.
<code>dstStep</code>	Distance in bytes between starts of consecutive lines in the destination image.
<code>roiSize</code>	Size of the source and destination ROI in pixels.

## Description

The function `ippiRGBToHSV` is declared in the `ippcc.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts the R'G'B' image `pSrc` to the [HSV](#) image `pDst`.

The conversion algorithm from RGB to HSV can be represented in pseudocode as follows:

```
// Value:
V = max(R,G,B);
// Saturation:
temp = min(R,G,B);
if V = 0 then // achromatics case
```

```

    S = 0//          H = 0
else // chromatics case
    S = (V - temp)/V
// Hue:
Cr = (V - R) / (V - temp)
Cg = (V - G) / (V - temp)
Cb = (V - B) / (V - temp)
if R = V then H = Cb - Cg
if G = V then H = 2 + Cr - Cb
if B = V then H = 4 + Cg - Cr
H = 60*H
if H < 0 then H = H + 360

```

The computed  $H, S, V$  values are scaled to the full range of the destination data type (see [Table “Image Data Types and Ranges”](#) in Chapter 2).

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <code>pSrc</code> or <code>pDst</code> is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <code>roiSize</code> has a field with a zero or negative value.

## HSVToRGB

*Converts an HSV image to the RGB color model.*

---

### Syntax

```

IppStatus ippHSVToRGB_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pDst,
int dstStep, IppiSize roiSize);

```

Supported values for `mod`:

```

8u_C3R      16u_C3R
8u_AC4R      16u_AC4R

```

### Parameters

<code>pSrc</code>	Pointer to the source image ROI.
<code>srcStep</code>	Distance in bytes between starts of consecutive lines in the source image.
<code>pDst</code>	Pointer to the destination image ROI.
<code>dstStep</code>	Distance in bytes between starts of consecutive lines in the destination image.
<code>roiSize</code>	Size of the source and destination ROI in pixels.

### Description

The function `ippiHSVToRGB` is declared in the `ippcc.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts a [HSV](#) image `pSrc` to the image  $R'G'B'$  `pDst`.



The conversion algorithm from HSV to RGB can be represented in pseudocode as follows:

```

if S = 0 then
    R = G = B = V
else
    if H = 360 then
        H = 0
    else
        H = H/60
        I = floor(H)
        F = H - I;
        M = V * ( 1 - S );
        N = V * ( 1 - S * F );
        K = V * ( 1 - S * (1 - F) );
        if(I == 0)then{ R = V;G = K;B = M;}
        if(I == 1)then{ R = N;G = V;B = M;}
        if(I == 2)then{ R = M;G = V;B = K;}
        if(I == 3)then{ R = M;G = N;B = V;}
        if(I == 4)then{ R = K;G = M;B = V;}
        if(I == 5)then{ R = V;G = M;B = N;}

```

The computed  $R'$ ,  $G'$ ,  $B'$  values are scaled to the full range of the destination data type (see [Table "Image Data Types and Ranges"](#) in Chapter 2).

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pSrc</i> or <i>pDst</i> is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with a zero or negative value.

## RGBToYCoCg

*Converts a RGB image to the YCoCg color model.*

### Syntax

```

IppStatus ippRGBToYCoCg_8u_C3P3R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst[3], int
dstStep, IppiSize roiSize);

```

### Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Array of pointers to the destination image ROI in each plane.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>roiSize</i>	Size of the source and destination ROI in pixels.

### Description

The function `ippRGBToYCoCg` is declared in the `ippcc.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts a RGB image *pSrc* to the **YCoCg** image *pDst* according to the following formulas:

$$Y = ((R + 2 * G + B) + 2) / 4$$

$$Co = ((R - B) + 1) / 2$$

$$Cg = ((-R + 2 * G - B) + 2) / 4$$

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointers is <code>NULL</code> .

## YCoCgToRGB

*Converts a YCoCg image to the RGB image.*

---

### Syntax

```
ippStatus ippiYCoCgToRGB_8u_P3C3R(const Ipp8u* pSrc[3], int srcStep, Ipp8u* pDst, int dstStep, IppiSize roiSize);
```

### Parameters

<i>pSrc</i>	Array of pointers to the source image ROI in each plane.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>roiSize</i>	Size of the source and destination ROI in pixels.

### Description

The function `ippiYCoCrToRGB` is declared in the `ippcc.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts the **YCoCg** image *pSrc* to the RGB image *pDst* according to the following formulas:

$$R = Y + Co - Cg$$

$$G = Y + Cg$$

$$B = Y - Co - Cg$$

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointers is <code>NULL</code> .

## BGRToYCoCg

Converts a 24-bit BGR image to the YCoCg color model.

### Syntax

```
IppStatus ippiBGRToYCoCg_8u16s_C3P3R(const Ipp8u* pBGR, int bgrStep, Ipp16s* pYCC[3], int yccStep, IppiSize roiSize);
```

```
IppStatus ippiBGRToYCoCg_8u16s_C4P3R(const Ipp8u* pBGR, int bgrStep, Ipp16s* pYCC[3], int yccStep, IppiSize roiSize);
```

### Parameters

<i>pBGR</i>	Pointer to the source image ROI.
<i>bgrStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pYCC</i>	Array of pointers to the destination image ROI in each plane.
<i>yccStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>roiSize</i>	Size of the source and destination ROI in pixels.

### Description

The function `ippiBGRToYCoCg` is declared in the `ippcc.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts the 24-bit BGR image *pBGR* to the YCoCg image *pYCC* according to the following formulas:

$$Y = ((R + 2 \cdot G + B) + 2) / 4$$

$$Co = ((R - B) + 1) / 2$$

$$Cg = ((-R + 2 \cdot G - B) + 2) / 4$$

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointers is <code>NULL</code> .

## SBGRToYCoCg

Converts a 48-bit BGR image to the YCoCg color model.

### Syntax

```
IppStatus ippiSBGRToYCoCg_<mod>(const Ipp16s* pBGR, int bgrStep, Ipp<dstDatatype>* pYCC[3], int yccStep, IppiSize roiSize);
```

Supported values for `mod`:

<code>16s_C3P3R</code>	<code>16s32s_C3P3R</code>
<code>16s_C4P3R</code>	<code>16s32s_C4P3R</code>

## Parameters

<i>pBGR</i>	Pointer to the source image ROI.
<i>bgrStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pYCC</i>	Array of pointers to the destination image ROI in each plane.
<i>yccStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>roiSize</i>	Size of the source and destination ROI in pixels.

## Description

The function `ippiBGRToYCoCg` is declared in the `ippcc.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts the 48-bit BGR image *pBGR* to the YCoCg image *pYCC* according to the following formulas:

$$Y = ((R + 2 \cdot G + B) + 2) / 4$$

$$Co = ((R - B) + 1) / 2$$

$$Cg = ((-R + 2 \cdot G - B) + 2) / 4$$

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointers is <code>NULL</code> .

## YCoCgToBGR

Converts a YCoCg image to the 24-bit BGR image.

### Syntax

```
ippStatus ippiYCoCgToBGR_16s8u_P3C3R(const Ipp16s* pYCC[3], int yccStep, Ipp8u* pBGR, int bgrStep, IppiSize roiSize);
```

```
ippStatus ippiYCoCgToBGR_16s8u_P3C4R(const Ipp16s* pYCC[3], int yccStep, Ipp8u* pBGR, int bgrStep, IppiSize roiSize, Ipp8u aval);
```

## Parameters

<i>pYCC</i>	Array of pointers to the source image ROI in each plane.
<i>yccStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pBGR</i>	Pointer to the destination image ROI.
<i>bgrStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>roiSize</i>	Size of the source and destination ROI in pixels.
<i>aval</i>	Constant value to create the fourth channel.

## Description

The function `ippiYCoCrToBGR` is declared in the `ippcc.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts the **YCoCg** image `pYCC` to the 24-bit BGR image `pBGR` according to the following formulas:

$$R = Y + Co - Cg$$

$$G = Y + Cg$$

$$B = Y - Co - Cg$$

The fourth channel is created by setting channel values to the constant value `aval`.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointers is <code>NULL</code> .

## YCoCgToSBGR

Converts a YCoCg image to the 48-bit BGR image.

## Syntax

### Case 1: Conversion to 3-channel image

```
ippiYCoCgToSBGR_16s_P3C3R(const Ipp16s* pYCC[3], int yccStep, Ipp16s* pBGR, int bgrStep, IppiSize roiSize);
```

```
IppStatus ippiYCoCgToSBGR_32s16s_P3C3R(const Ipp32s* pYCC[3], int yccStep, Ipp16s* pBGR, int bgrStep, IppiSize roiSize);
```

### Case 2: Conversion to 4-channel image

```
IppStatus ippiYCoCgToSBGR_16s_P3C4R(const Ipp16s* pYCC[3], int yccStep, Ipp16s* pBGR, int bgrStep, IppiSize roiSize, Ipp16s aval);
```

```
IppStatus ippiYCoCgToSBGR_32s16s_P3C4R(const Ipp32s* pYCC[3], int yccStep, Ipp16s* pBGR, int bgrStep, IppiSize roiSize, Ipp16s aval);
```

## Parameters

<code>pYCC</code>	Array of pointers to the source image ROI in each plane.
<code>yccStep</code>	Distance in bytes between starts of consecutive lines in the source image.
<code>pBGR</code>	Pointer to the destination image ROI.
<code>bgrStep</code>	Distance in bytes between starts of consecutive lines in the destination image.
<code>roiSize</code>	Size of the source and destination ROI in pixels.
<code>aval</code>	Constant value to create the fourth channel.

## Description

The function `ippiYCoCrToBGR` is declared in the `ippcc.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts the **YCoCg** image `pYCC` to the 48-bit BGR image `pBGR` according to the following formulas:

$$R = Y + Co - Cg$$

$$G = Y + Cg$$

$$B = Y - Co - Cg$$

The fourth channel is created by setting channel values to the constant value `aval`.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointers is <code>NULL</code> .

## BGRToYCoCg\_Rev

Converts a 24-bit BGR image to the YCoCg-R color model.

### Syntax

```
ippStatus ippiBGRToYCoCg_Rev_8u16s_C3P3R(const Ipp8u* pBGR, int bgrStep, Ipp16s* pYCC[3],
int yccStep, IppiSize roiSize);
```

```
ippStatus ippiBGRToYCoCg_Rev_8u16s_C4P3R(const Ipp8u* pBGR, int bgrStep, Ipp16s* pYCC[3],
int yccStep, IppiSize roiSize);
```

### Parameters

<code>pBGR</code>	Pointer to the source image ROI.
<code>bgrStep</code>	Distance in bytes between starts of consecutive lines in the source image.
<code>pYCC</code>	Array of pointers to the destination image ROI in each plane.
<code>yccStep</code>	Distance in bytes between starts of consecutive lines in the destination image.
<code>roiSize</code>	Size of the source and destination ROI in pixels.

## Description

The function `ippiBGRToYCoCg_rev` is declared in the `ippcc.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts the 24-bit BGR image `pSrc` to the **YCoCg-R** image `pDst` according to the following formulas:

$$Co = R - B$$

$$t = B + (Co >> 1)$$

$$Cg = G - t$$

$$Y = t + (Cg \gg 1)$$

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointers is <code>NULL</code> .

## SBGRToYCoCg\_Rev

Converts a 48-bit BGR image to the YCoCg-R color model.

### Syntax

```
IppStatus ippISBGRToYCoCg_Rev_<mod>(const Ipp16s* pBGR, int bgrStep, Ipp<dstDatatype>*
pYCC[3], int yccStep, IppiSize roiSize);
```

Supported values for `mod`:

<code>16s_C3P3R</code>	<code>16s32s_C3P3R</code>
<code>16s_C4P3R</code>	<code>16s32s_C4P3R</code>

### Parameters

<code>pBGR</code>	Pointer to the source image ROI.
<code>bgrStep</code>	Distance in bytes between starts of consecutive lines in the source image.
<code>pYCC</code>	Array of pointers to the destination image ROI in each plane.
<code>yccStep</code>	Distance in bytes between starts of consecutive lines in the destination image.
<code>roiSize</code>	Size of the source and destination ROI in pixels.

### Description

The function `ippISBGRToYCoCg_Rev` is declared in the `ippcc.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts the 48-bit BGR image `pBGR` to the [YCoCg-R](#) image `pYCC` according to the following formulas:

$$\begin{aligned} Co &= R - B \\ t &= B + (Co \gg 1) \\ Cg &= G - t \\ Y &= t + (Cg \gg 1) \end{aligned}$$

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointers is <code>NULL</code> .

## YCoCgToBGR\_Rev

Converts a YCoCg-R image to the 24-bit BGR image.

### Syntax

```
IppStatus ippiYCoCgToBGR_Rev_16s8u_P3C3R(const Ipp16s* pYCC[3], int yccStep, Ipp8u* pBGR,
int bgrStep, IppiSize roiSize);
```

```
IppStatus ippiYCoCgToBGR_Rev_16s8u_P3C4R(const Ipp16s* pYCC[3], int yccStep, Ipp8u* pBGR,
int bgrStep, IppiSize roiSize, Ipp8u aval);
```

### Parameters

<i>pYCC</i>	Array of pointers to the source image ROI in each plane.
<i>yccStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pBGR</i>	Pointer to the destination image ROI.
<i>bgrStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>roiSize</i>	Size of the source and destination ROI in pixels.
<i>aval</i>	Constant value to create the fourth channel.

### Description

The function `ippiYCoCrToBGR_Rev` is declared in the `ippcc.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts the [YCoCg-R](#) image `pYCC` to the 24-bit BGR image `pBGR` according to the following formulas:

$$t = Y - (Cg \gg 1)$$

$$G = Cg + t$$

$$B = t - (Co \gg 1)$$

$$R = B + Co$$

The fourth channel is created by setting channel values to the constant value `aval`.

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointers is <code>NULL</code> .



## YCoCgToSBGR\_Rev

*Converts a YCoCg-R image to the 48-bit BGR image.*

### Syntax

#### Case 1: Conversion to 3-channel image.

```
ippiYCoCgToSBGR_Rev_16s_P3C3R(const Ipp16s* pYCC[3], int yccStep, Ipp16s* pBGR, int bgrStep, IppiSize roiSize);
```

```
IppStatus ippiYCoCgToSBGR_Rev_32s16s_P3C3R(const Ipp32s* pYCC[3], int yccStep, Ipp16s* pBGR, int bgrStep, IppiSize roiSize);
```

#### Case 2: Conversion to 4-channel image

```
IppStatus ippiYCoCgToSBGR_Rev_16s_P3C4R(const Ipp16s* pYCC[3], int yccStep, Ipp16s* pBGR, int bgrStep, IppiSize roiSize, Ipp16s aval);
```

```
IppStatus ippiYCoCgToSBGR_Rev_32s16s_P3C4R(const Ipp32s* pYCC[3], int yccStep, Ipp16s* pBGR, int bgrStep, IppiSize roiSize, Ipp16s aval);
```

### Parameters

<i>pYCC</i>	Array of pointers to the source image ROI in each plane.
<i>yccStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pBGR</i>	Pointer to the destination image ROI.
<i>bgrStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>roiSize</i>	Size of the source and destination ROI in pixels.
<i>aval</i>	Constant value to create the fourth channel.

### Description

The function `ippiYCoCrToBGR_Rev` is declared in the `ippcc.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts the **YCoCg-R** image *pYCC* to the 48-bit BGR image *pBGR* according to the following formulas:

$$t = Y - (Cg \gg 1)$$

$$G = Cg + t$$

$$B = t - (Co \gg 1)$$

$$R = B + Co$$

The fourth channel is created by setting channel values to the constant value *aval*.

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointers is <code>NULL</code> .

## Color - Gray Scale Conversions

### RGBToGray

Converts an RGB image to gray scale using fixed transform coefficients.

#### Syntax

```
IppStatus ippiRGBToGray_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pDst,
int dstStep, IppiSize roiSize);
```

Supported values for mod:

8u_C3C1R	16u_C3C1R	16s_C3C1R	32f_C3C1R
8u_AC4C1R	16u_AC4C1R	16s_AC4C1R	32f_AC4C1R

#### Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>roiSize</i>	Size of the source and destination ROI in pixels.

#### Description

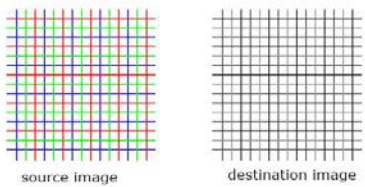
The function `ippiRGBToGray` is declared in the `ippcc.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts an RGB image to gray scale (see [Figure Converting an RGB Image to Gray Scale](#)) using the following basic equation to compute luma from nonlinear gamma-corrected red, green, and blue values:

$$Y' = 0.299 * R' + 0.587 * G' + 0.114 * B'$$

Note that the transform coefficients conform to the standard for the NTSC red, green, and blue CRT phosphors.

#### Converting an RGB Image to Gray Scale



## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <code>pSrc</code> or <code>pDst</code> pointer is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <code>roiSize</code> has a field with a zero or negative value.

## Example

The code example below demonstrates how to use the function `ippiRGBToGray_8u_C3C1R`.

```

Ipp8u src[12*3] = { 255, 0, 0, 255, 0, 0, 255, 0, 0, 255, 0, 0,
                    0, 255, 0, 0, 255, 0, 0, 255, 0, 0, 255, 0,
                    0, 0, 255, 0, 0, 255, 0, 0, 255, 0, 0, 255};

Ipp8u dst[4*3];
IppiSize srcRoi = { 4, 3 };

ippiRGBToGray_8u_C3C1R ( src, 12, dst, 4, srcRoi );

```

Result:

```

255 0 0 255 0 0 255 0 0 255 0 0
0 255 0 0 255 0 0 255 0 0 255 0      src
0 0 255 0 0 255 0 0 255 0 0 255

76 76 76 76
149 149 149 149      dst
29 29 29 29

```

## ColorToGray

*Converts an RGB image to gray scale using custom transform coefficients.*

### Syntax

```

IppStatus ippiColorToGray_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>*
pDst, int dstStep, IppiSize roiSize, const Ipp32f coeffs[3]);

```

Supported values for `mod`:

8u_C3C1R	16u_C3C1R	16s_C3C1R	32f_C3C1R	64f_C3C1R
8u_AC4C1R	16u_AC4C1R	16s_AC4C1R	32f_AC4C1R	64f_AC4C1R

### Parameters

<code>pSrc</code>	Pointer to the source image ROI.
<code>srcStep</code>	Distance in bytes between starts of consecutive lines in the source image.
<code>pDst</code>	Pointer to the destination image ROI.
<code>dstStep</code>	Distance in bytes between starts of consecutive lines in the destination image.
<code>roiSize</code>	Size of the source and destination ROI in pixels.

*coeffs*

Transform coefficients.

## Description

The function `ippiColorToGray` is declared in the `ippcc.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function uses the following equation to convert an RGB image to gray scale:

$$Y = coeffs[0] * R + coeffs[1] * G + coeffs[2] * B,$$

where the *coeffs* array contains user-defined transform coefficients which must be non-negative and satisfy the condition

$$coeffs[0] + coeffs[1] + coeffs[2] \leq 1.$$

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pSrc</i> or <i>pDst</i> is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with a zero or negative value.

## CFAToRGB

Restores the RGB image from the gray-scale CFA image.

## Syntax

```
ippStatus ippiCFAToRGB_8u_C1C3R(const Ipp8u* pSrc, IppiRect srcRoi, IppiSize srcSize, int srcStep, Ipp8u* pDst, int dstStep, IppiBayerGrid grid, int interpolation);
```

```
ippStatus ippiCFAToRGB_16u_C1C3R(const Ipp16u* pSrc, IppiRect srcRoi, IppiSize srcSize, int srcStep, Ipp16u* pDst, int dstStep, IppiBayerGrid grid, int interpolation);
```

## Parameters

<i>pSrc</i>	Pointer to the source image origin.
<i>srcSize</i>	Size of the source image.
<i>srcRoi</i>	Region of interest in the source image (of the <code>IppiRect</code> type).
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>grid</i>	Specifies the configuration of the Bayer grid in the source image. The following values are possible: <code>ippiBayerBGGR</code> <code>ippiBayerRGGB</code> <code>ippiBayerGBRG</code> <code>ippiBayerGRBG</code>

*interpolation*

Interpolation method, reserved, must be 0.

## Description

The function `ippiCFAToRGB` is declared in the `ippcc.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function transforms the one-channel gray-scale image *pSrc* that is produced by applying the color filter array (CFA) - an array of Bayer filters, to 24-bit three-channel RGB image. The order of the color component in the source image - Bayer grid - is specified by the parameter *grid*. Four possible values of this parameter correspond to the allowed variants of the Bayer grid (see Figure below).

## Possible Configurations of the Bayer Grids



Each element of the source image contains an intensity value for only one color component, two others are interpolated using neighbor elements. R and B values are interpolated linearly from the nearest neighbors of the same color. When interpolating R and B values on green pixel, the average values of the two nearest neighbors (above and below, or left and right) of the same colors are used. When interpolating R or B values on the blue or red pixel respectively, the average values of the four nearest blue (red) pixels cornering the red (blue) pixel are used. G values are interpolated using an adaptive interpolation [Sak98] from a pair of nearest neighbors (vertical or horizontal) and taking into account the correlation in the red (or blue) component. The pair is chosen depending on the values of the difference between the red (blue) pixels in the vertical and horizontal directions. If the difference is smaller in the vertical direction - a vertical pair of green pixels is used, if it is smaller in the horizontal direction - a horizontal pair is used. If the difference is the same, all four neighbors are used.

This interpolation requires border pixels for the input pixels near the horizontal or vertical edge of the image. The function uses the mirrored border of two edge rows or columns of the input image. In this case the G values is calculated as the average of four nearest green pixels.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointer is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if the <code>srcSize</code> has a field that is less than 2, or if the <code>roiSize</code> has a field with negative or zero value.
<code>ippStsBadArgErr</code>	Indicates an error condition if <i>grid</i> has an illegal value.

## DemosaicAHD

*Restores the RGB image from the gray-scale CFA image using AHD algorithm.*

---

### Syntax

```
IppStatus ippiDemosaicAHD_8u_C1C3R(const Ipp8u* pSrc, IppiRect srcRoi, IppiSize srcSize,
int srcStep, Ipp8u* pDst, int dstStep, IppiBayerGrid grid, Ipp8u* pTmp, int tmpStep);

IppStatus ippiDemosaicAHD_16u_C1C3R(const Ipp16u* pSrc, IppiRect srcRoi, IppiSize srcSize,
int srcStep, Ipp16u* pDst, int dstStep, IppiBayerGrid grid, Ipp16u* pTmp, int tmpStep);
```

### Parameters

<i>pSrc</i>	Pointer to the source image.
<i>srcRoi</i>	Region of interest in the source image (of the <code>IppiRect</code> type).
<i>srcSize</i>	Size of the source image.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>grid</i>	Specifies the configuration of the Bayer grid in the source image. The following values are possible (see <a href="#">Figure Possible Configurations of the Bayer Grids</a> ): <code>ippiBayerBGGR</code> <code>ippiBayerRGGB</code> <code>ippiBayerGBRG</code> <code>ippiBayerGRBG</code>
<i>pTmp</i>	Pointer to the temporary image of ( <code>srcRoi.width + 6, 30</code> ) size.
<i>tmpStep</i>	Distance in bytes between starts of consecutive lines in the temporary image.

### Description

The function `ippiDemosaicAHD` is declared in the `ippcc.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function transforms the one-channel gray-scale image *pSrc* that is produced by applying the color filter array (CFA) to 24-bit three-channel RGB image using the adaptive homogeneity-directed demosaicing (AHD) algorithm [\[Hir05\]](#). The algorithm requires the temporary image *pTmp* of size `srcRoi.width + 6, 30`.

The type of the Bayer grid (see [Figure Possible Configurations of the Bayer Grids](#)) is specified by the parameter *grid*.

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointer is <code>NULL</code> .

<code>ippStsSizeErr</code>	Indicates an error condition if the <i>srcSize</i> has a field that is less than 5, or if the <i>roiSize</i> has a field with negative or zero value.
<code>ippStsBadArgErr</code>	Indicates an error condition if <i>grid</i> has an illegal value.

## Format Conversion

This section describes Intel IPP functions that perform image color conversion without changing the color space. These functions convert pixel-order images to planar format and vice versa, change the number of channels or planes, alter sampling formats and sequences of samples and planes. Several functions additionally perform filtering - deinterlacing and upsampling.

Intel IPP format conversion functions are specified mainly in the YCbCr color space, but as they do not transform color model they may be used to perform described types of conversion for any other color spaces with decoupled luminance and chrominance coordinates (YUV type).

### RGBToRGB565, BGRTOBGR565

Convert 24-bit per pixel image to the 16-bit image.

#### Syntax

```
IppStatus ippRGBToRGB565_8u16u_C3R(const Ipp8u* pSrc, int srcStep, Ipp16u* pDst, int dstStep, IppiSize roiSize);
```

```
IppStatus ippBGRTOBGR565_8u16u_C3R(const Ipp8u* pSrc, int srcStep, Ipp16u* pDst, int dstStep, IppiSize roiSize);
```

#### Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>roiSize</i>	Size of the source and destination ROI in pixels.

#### Description

The functions `ippRGBToRGB565` and `ippBGRTOBGR565` are declared in the `ippcc.h` file. They operate with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts the 24-bit per pixel gamma-corrected R'G'B' (B'G'R') image *pSrc* to the packed 16-bit image *pDst*: all 3 channel intensities for a pixel are packed into two consecutive bytes (16u data type). The destination image has format RGB565 (BGR565) - 5 bits for blue, 6 bits for green, 5 bits for red (see [Figure Converting an RGB Image to Gray Scale](#) for more details).

#### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
--------------------------	---

<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pSrc</i> or <i>pDst</i> is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with a zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if one of <i>srcStep</i> or <i>dstStep</i> is less than or equal to 0.

## RGB565ToRGB, BGR565ToBGR

Convert 16-bit per pixel image to the 24-bit image.

### Syntax

```
IppStatus ippRGB565ToRGB_16u8u_C3R(const Ipp16u* pSrc, int srcStep, Ipp8u* pDst, int dstStep, IppiSize roiSize);
```

```
IppStatus ippBGR565ToBGR_16u8u_C3R(const Ipp16u* pSrc, int srcStep, Ipp8u* pDst, int dstStep, IppiSize roiSize);
```

### Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>roiSize</i>	Size of the source and destination ROI in pixels.

### Description

The functions `ippRGB565ToRGB` and `ippBGR565ToBGR` are declared in the `ippcc.h` file. They operate with ROI (see [Regions of Interest in Intel IPP](#)).

These functions convert the 16-bit per pixel gamma-corrected R'G'B' (B'G'R') image *pSrc* to the 24-bit image *pDst*. The source image has a packed format RGB565 (BGR565), that is all 3 channel intensities for a pixel are packed into two consecutive bytes (16u data type) in the following order: 5 bits for blue, 6 bits for green, 5 bits for red (see [Figure Converting an RGB Image to Gray Scale](#) for more details).

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pSrc</i> or <i>pDst</i> is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with a zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if one of <i>srcStep</i> or <i>dstStep</i> is less than or equal to 0.



## YCbCr422

*Converts 4:2:2 YCbCr image.*

---

### Syntax

```
IppStatus ippiYCbCr422_8u_C2P3R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);
```

```
IppStatus ippiYCbCr422_8u_P3C2R(const Ipp8u* pSrc[3], int srcStep[3], Ipp8u* pDst, int dstStep, IppiSize roiSize);
```

### Parameters

<i>pSrc</i>	Pointer to the ROI in the pixel-order source image. Array of pointers to the ROI in each plane of the planar source image.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image. Array of distance values for the source image planes.
<i>pDst</i>	Pointer to the ROI in the pixel-order destination image. Array of pointers to the ROI in each plane of the planar destination image.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image. Array of distance values for the destination image planes.
<i>roiSize</i>	Size of the source and destination ROI in pixels.

### Description

The function `ippiYCbCr422` is declared in the `ippcc.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts the 4:2:2 two-channel source image *pSrc* to the 4:2:2 three-plane image *pDst* and vice versa (see [Table "Pixel-Order Image Formats"](#) and [Table "Planar Image Formats"](#) for more details on 4:2:2 planar and pixel-order formats).

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if any of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize.width</i> of the first plane is less than 2, or <i>roiSize.height</i> is less than or equal to zero.

## YCbCr422ToYCrCb422

Converts 4:2:2 YCbCr image to 4:2:2 YCrCb image.

### Syntax

```
IppStatus ippiYCbCr422ToYCrCb422_8u_C2R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst, int dstStep, IppiSize roiSize);
```

```
IppStatus ippiYCbCr422ToYCrCb422_8u_P3C2R(const Ipp8u* pSrc[3], int srcStep[3], Ipp8u* pDst, int dstStep, IppiSize roiSize);
```

### Parameters

<i>pSrc</i>	Pointer to the ROI in the pixel-order source image. Array of pointers to the ROI in each plane of the planar source image.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image. Array of distance values for the source image planes.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>roiSize</i>	Size of the ROI in pixels, its width should be multiple of 2.

### Description

The function `ippiYCbCr422ToYCrCb422` is declared in the `ippcc.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts 4:2:2 YCbCr source image *pSrc* to the 4:2:2 YCrCb two-channel image *pDst* that has the following sequence of samples: Y0, Cr0, Y1, Cb0, Y2, Cr1, Y3, Cb1, ... (see [Table "Pixel-Order Image Formats"](#)). The source image can be either two-channel or three-plane (see [Table "Pixel-Order Image Formats"](#) and [Table "Planar Image Formats"](#)).

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if any of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize.width</i> is less than 2.

## YCbCr422ToCbYCr422

Converts 4:2:2 YCbCr image to 4:2:2 CbYCr image.

### Syntax

```
IppStatus ippiYCbCr422ToCbYCr422_8u_C2R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst, int dstStep, IppiSize roiSize);
```

```
IppStatus ippiYCbCr422ToCbYCr422_8u_P3C2R(const Ipp8u* pSrc[3], int srcStep[3], Ipp8u* pDst, int dstStep, IppiSize roiSize);
```

## Parameters

<i>pSrc</i>	Pointer to the ROI in the pixel-order source image. Array of pointers to the ROI in each plane of the planar source image.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image. Array of distance values for the source image planes.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>roiSize</i>	Size of the source and destination ROI in pixels; its width should be multiple of 2.

## Description

The functions `ippiYCbCr422ToCbYCr422` is declared in the `ippcc.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts 4:2:2 YCbCr source image *pSrc* to the 4:2:2 CbYCr two-channel image *pDst* that has the following sequence of samples: Cb0, Y0, Cr0, Y1, Cb1, Y2, Cr1, Y3, Cb3, ... (see [Table "Pixel-Order Image Formats"](#)). The source image can be either two-channel or three-plane (see [Table "Pixel-Order Image Formats"](#) and [Table "Planar Image Formats"](#)).

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if any of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize.width</i> of the first plane is less than 2, or <i>roiSize.height</i> is less than or equal to zero.

## YCbCr422ToYCbCr420

*Converts YCbCr image from 4:2:2 sampling format to 4:2:0 format.*

---

## Syntax

### Case 1: Operation on planar data

```
ippStatus ippiYCbCr422ToYCbCr420_8u_P3R(const Ipp8u* pSrc[3], int srcStep[3], Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);
```

```
ippStatus ippiYCbCr422ToYCbCr420_8u_P3P2R(const Ipp8u* pSrc[3], int srcStep[3], Ipp8u* pDstY, int dstYStep, Ipp8u* pDstCbCr, int dstCbCrStep, IppiSize roiSize);
```

### Case 2: Conversion from pixel-order to planar data

```
ippStatus ippiYCbCr422ToYCbCr420_8u_C2P3R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);
```

```
ippStatus ippiYCbCr422ToYCbCr420_8u_C2P2R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDstY, int dstYStep, Ipp8u* pDstCbCr, int dstCbCrStep, IppiSize roiSize);
```

## Parameters

<i>pSrc</i>	Pointer to the ROI in the pixel-order source image. Array of pointers to the ROI in each plane of the planar source image.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image. Array of distance values for the source image planes.
<i>pDst</i>	Array of pointers to the ROI in each plane for a three-plane destination image.
<i>dstStep</i>	Array of distances in bytes between starts of consecutive lines in each plane for a three-plane destination image.
<i>pDstY</i>	Pointer to the ROI in the luminance plane for a two-plane destination image.
<i>dstYStep</i>	Distance in bytes between starts of consecutive lines in the luminance plane of a destination image.
<i>pDstCbCr</i>	Pointer to the ROI in the interleaved chrominance plane for a two-plane destination image.
<i>dstCbCrStep</i>	Distance in bytes between starts of consecutive lines in the chrominance plane of a destination image.
<i>roiSize</i>	Size of the ROI in pixels, height and width should be multiple of 2.

## Description

The function `ippiYCbCr422ToYCbCr420` is declared in the `ippcc.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts the `4:2:2` image *pSrc* to the `4:2:0` image. The source image can be two-channel or three-plane, destination image always is planar with two or three planes (see [Table "Pixel-Order Image Formats"](#) and [Table "Planar Image Formats"](#)). Two-plane image contains luminance samples `Y0, Y1, Y2, ..` in the first plane *pDstY*, and interleaved chrominance samples `Cb0, Cr0, Cb1, Cr1, ...` in the second plane *pDstCbCr*.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if any of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if any field of the <i>roiSize</i> is less than 2.

## Example

The code example below shows how to use the function `ippiYCbCr422ToYCbCr420_8u_C2P3R`.

```
{
    Ipp8u*   ImageI420[3];
    int      stepI420[3];
    Ipp8u*   ImageYUY2;
    int      stepYUY2;
    IppiSize roiSize = { 1024, 768 };
    ImageI420[0] = ippiMalloc_8u_C1( roiSize.width, roiSize.height, &(stepI420[0]));
    ImageI420[1] = ippiMalloc_8u_C1( roiSize.width, roiSize.height, &(stepI420[1]));
}
```

```

ImageI420[2] = ippiMalloc_8u_C1( roiSize.width, roiSize.height, &(stepI420[2]));
ImageYUY2     = ippiMalloc_8u_C2( roiSize.width, roiSize.height, &stepYUY2 );
ippiYCbCr422ToYCbCr420_8u_C2P3R( ImageYUY2, stepYUY2, ImageI420, stepI420, roiSize);

ippiFree(ImageI420[0]);
ippiFree(ImageI420[1]);
ippiFree(ImageI420[2]);
ippiFree(ImageYUY2);
}

```

## YCbCr422To420\_Interlace

*Converts interlaced YCbCr image from 4:2:2 sampling format to 4:2:0 format.*

### Syntax

```

IppStatus ippiYCbCr422To420_Interlace_8u_P3R(const Ipp8u* pSrc[3], int srcStep[3], Ipp8u*
pDst[3], int dstStep[3], IppiSize roiSize);

```

### Parameters

<i>pSrc</i>	Array of pointers to the ROI in each plane for source image.
<i>srcStep</i>	Array of distances in bytes between starts of consecutive lines in each plane for the source image.
<i>pDst</i>	Array of pointers to the ROI in each plane for destination image.
<i>dstStep</i>	Array of distances in bytes between starts of consecutive lines in each plane for destination image.
<i>roiSize</i>	Size of the source and destination ROI in pixels, its width must be multiple of 2, and height must be multiple of 4.

### Description

The function `ippiYCbCr422To420_Interlace` is declared in the `ippcc.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts the interlaced 4:2:2 image *pSrc* to the 4:2:0 image *pDst* (see [Table "Planar Image Formats"](#)).

The conversion is performed in accordance with the following formulas:

$$Y1_{dest} = Y1_{src};$$

$$Cb0(Cr0)_{dest} = (3 * Cb0(Cr0)_{src} + Cb2(Cr2)_{src} + 2) / 4;$$

$$Cb1(Cr1)_{dest} = (Cb1(Cr1)_{src} + 3 * Cb3(Cr3)_{src} + 2) / 4;$$

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if any of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize.width</i> is less than 2, or <i>roiSize.height</i> is less than 4.

`ippStsDoubleSize`

Indicates a warning if `roiSize.width` is not multiple of 2, or `roiSize.height` is not multiple of 4.

## YCbCr422ToYCrCb420

*Converts 4:2:2 YCbCr image to 4:2:0 YCrCb image.*

---

### Syntax

```
IppStatus ippiYCbCr422ToYCrCb420_8u_C2P3R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst[3],
int dstStep[3], IppiSize roiSize);
```

### Parameters

<code>pSrc</code>	Pointer to the source image ROI.
<code>srcStep</code>	Distance in bytes between starts of consecutive lines in the source image.
<code>pDst</code>	Array of pointers to the ROI in each plane of the destination image.
<code>dstStep</code>	Array of distances in bytes between starts of consecutive lines in the destination image planes.
<code>roiSize</code>	Size of the source and destination ROI in pixels, height and width should be multiple of 2.

### Description

The function `ippiYCbCr422ToYCrCb420` is declared in the `ippcc.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts the 4:2:2 two-channel image `pSrc` that has the following sequence of samples: Y0, Cb0, Y1, Cr0, Y2, Cb1, Y3, Cr1, ... to the 4:2:0 three-plane image `pDst` with the following order of pointers: Y-plane, Cr-plane, Cb-plane (see [Table "Pixel-Order Image Formats"](#) and [Table "Planar Image Formats"](#)).

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if any of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if any field of the <code>roiSize</code> is less than 2.

## YCbCr422ToYCbCr411

Converts YCbCr image from 4:2:2 sampling format to 4:1:1 format.

### Syntax

#### Case 1: Operation on planar data

```
IppStatus ippiYCbCr422ToYCbCr411_8u_P3R(const Ipp8u* pSrc[3], int srcStep[3], Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);
```

```
IppStatus ippiYCbCr422ToYCbCr411_8u_P3P2R(const Ipp8u* pSrc[3], int srcStep[3], Ipp8u* pDstY, int dstYStep, Ipp8u* pDstCbCr, int dstCbCrStep, IppiSize roiSize);
```

#### Case 2: Conversion from pixel-order to planar data

```
IppStatus ippiYCbCr422ToYCbCr411_8u_C2P3R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);
```

```
IppStatus ippiYCbCr422ToYCbCr411_8u_C2P2R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDstY, int dstYStep, Ipp8u* pDstCbCr, int dstCbCrStep, IppiSize roiSize);
```

### Parameters

<i>pSrc</i>	Pointer to the ROI in the pixel-order source image. Array of pointers to the ROI in each plane of the planar source image.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image. Array of distance values for the source image planes.
<i>pDst</i>	Array of pointers to the ROI in each plane for a three-plane destination image.
<i>dstStep</i>	Array of distances in bytes between starts of consecutive lines in each plane for a three-plane destination image.
<i>pDstY</i>	Pointer to the ROI in the luminance plane for a two-plane destination image.
<i>dstYStep</i>	Distance in bytes between starts of consecutive lines in the luminance plane of a destination image.
<i>pDstCbCr</i>	Pointer to the ROI in the interleaved chrominance plane for a two-plane destination image.
<i>dstCbCrStep</i>	Distance in bytes between starts of consecutive lines in the chrominance plane of a destination image.
<i>roiSize</i>	Size of the ROI in pixels.

### Description

The function `ippiYCbCr422ToYCbCr411` is declared in the `ippcc.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts the 4:2:2 image *pSrc* to the 4:1:1 image. The source image can be two-channel or three-plane (see [Table “Pixel-Order Image Formats”](#) for more details), destination image always is planar with two or three planes (see [Table “Planar Image Formats”](#) for more details). The two-plane image contains luminance samples *Y0*, *Y1*, *Y2*, .. in the first plane *pDstY*, and interleaved chrominance samples *Cb0*, *Cr0*, *Cb1*, *Cr1*, ... in the second plane *pDstCbCr*.

The value of the fields of the *roiSize* have certain limitations:

- its width should be multiple of 4 and cannot be less than 4 for operation on two-channel images;
- its width should be multiple of 4 and cannot be less than 4, and its height should be multiple of 2 and can not be less than 2 for three-plane to two-plane image conversion;
- both height and width should be multiple of 2 and cannot be less than 2 for operation on three-plane images.

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if any of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if corresponding fields of the <i>roiSize</i> is less than specified above values.

## YCrCb422ToYCbCr422

Converts 4:2:2 YCrCb image to 4:2:2 YCbCr image.

### Syntax

```
IppStatus ippYCrCb422ToYCbCr422_8u_C2P3R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst[3],
int dstStep[3], IppiSize roiSize);
```

### Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Array of pointers to the ROI in each plane of the destination image.
<i>dstStep</i>	Array of distances in bytes between starts of consecutive lines in the destination image planes.
<i>roiSize</i>	Size of the ROI in pixels, its width should be multiple of 2.

### Description

The function `ippYCrCb422ToYCbCr422` is declared in the `ippcc.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts the 4:2:2 YCrCb two-channel image *pSrc* (see [Table “Pixel-Order Image Formats”](#)) to the 4:2:2 YCbCr three-plane image *pDst* (see [Table “Planar Image Formats”](#)).



## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if any of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <code>roiSize.width</code> is less than 2.

## YCrCb422ToYCbCr420

Converts 4:2:2 YCrCb image to 4:2:0 YCbCr image.

## Syntax

```
IppStatus ippYCrCb422ToYCbCr420_8u_C2P3R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst[3],
int dstStep[3], IppiSize roiSize);
```

## Parameters

<code>pSrc</code>	Pointer to the source image ROI.
<code>srcStep</code>	Distance in bytes between starts of consecutive lines in the source image.
<code>pDst</code>	Array of pointers to the ROI in each plane of the destination image.
<code>dstStep</code>	Array of distances in bytes between starts of consecutive lines in the destination image planes.
<code>roiSize</code>	Size of the ROI in pixels, height and width should be multiple of 2.

## Description

The function `ippYCrCb422ToYCbCr420_8u_C2P3R` is declared in the `ippcc.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts the 4:2:2 YCrCb two-channel image `pSrc` (see [Table "Pixel-Order Image Formats"](#)) to the 4:2:0 YCbCr three-plane image `pDst` (see [Table "Planar Image Formats"](#)).

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if any of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if any field of the <code>roiSize</code> is less than 2.

## YCrCb422ToYCbCr411

Converts 4:2:2 YCrCb image to 4:1:1 YCbCr image.

### Syntax

```
IppStatus ippiYCrCb422ToYCbCr411_8u_C2P3R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst[3],
int dstStep[3], IppiSize roiSize);
```

### Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Array of pointers to the ROI in each plane of the destination image.
<i>dstStep</i>	Array of distances in bytes between starts of consecutive lines in the destination image planes.
<i>roiSize</i>	Size of the ROI in pixels, its width should be multiple of 4.

### Description

The function `ippiYCrCb422ToYCbCr411_8u_C2P3R` is declared in the `ippcc.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts the 4:2:2 YCrCb two-channel image *pSrc* (see [Table "Pixel-Order Image Formats"](#)) to the 4:1:1 YCbCr three-plane image *pDst* (see [Table "Planar Image Formats"](#)).

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if any of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize.width</i> is less than 4.

## CbYCr422ToYCbCr422

Converts 4:2:2 CbYCr image to 4:2:2 YCbCr image.

### Syntax

```
IppStatus ippiCbYCr422ToYCbCr422_8u_C2R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst, int
dstStep, IppiSize roiSize);
```

```
IppStatus ippiCbYCr422ToYCbCr422_8u_C2P3R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst[3],
int dstStep[3], IppiSize roiSize);
```

### Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.

<i>pDst</i>	Pointer to the ROI in the pixel-order destination image. Array of pointers to the ROI in each plane of the planar destination image.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image. Array of distance values for the destination image planes.
<i>roiSize</i>	Size of the ROI in pixels, its width should be multiple of 2.

## Description

The function `ippiCbYCr422ToYCbCr422` is declared in the `ippcc.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts the 4:2:2 CbYCr two-channel image *pSrc* to the 4:2:2 YCbCr two-channel or three-plane image *pDst* (see [Table "Pixel-Order Image Formats"](#) and [Table "Planar Image Formats"](#)). The source image has the following sequence of samples: Cb0, Y0, Cr0, Y1, Cb1, Y2, Cr1, Y3, Cb2, ... . Two-channel destination image has different sequence of samples: Y0, Cb0, Y1, Cr0, Y2, Cb1, Y3, Cr1, Y4, ....

## Return Values

<code>ppStsNoErr</code>	Indicates no error. Any other value indicates an error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if any of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize.width</i> is less than 2.

## CbYCr422ToYCbCr420

Converts 4:2:2 CbYCr image to 4:2:0 YCbCr image.

## Syntax

```
ippStatus ippiCbYCr422ToYCbCr420_8u_C2P3R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst[3],
int dstStep[3], IppiSize roiSize);
```

```
ippStatus ippiCbYCr422ToYCbCr420_8u_C2P2R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDstY,
int dstYStep, Ipp8u* pDstCbCr, int dstCbCrStep, IppiSize roiSize);
```

## Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Array of pointers to the ROI in each plane for a three-plane destination image.
<i>dstStep</i>	Array of distances in bytes between starts of consecutive lines in each plane for a three-plane destination image.
<i>pDstY</i>	Pointer to the ROI in the luminance plane for a two-plane destination image.
<i>dstYStep</i>	Distance in bytes between starts of consecutive lines in the luminance plane of a destination image.

<i>pDstCbCr</i>	Pointer to the ROI in the interleaved chrominance plane for a two-plane destination image.
<i>dstCbCrStep</i>	Distance in bytes between starts of consecutive lines in the chrominance plane of a destination image.
<i>roiSize</i>	Size of the source and destination ROI in pixels, height and width should be multiple of 2.

## Description

The function `ippiCbYCr422ToYCbCr420` is declared in the `ippcc.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts the 4:2:2 CbYCr two-channel image *pSrc* to the 4:2:0 YCbCr two- or three-plane image *pDst* (see [Table "Pixel-Order Image Formats"](#) and [Table "Planar Image Formats"](#)). The source image has the following sequence of samples: Cb0, Y0, Cr0, Y1, Cb1, Y2, Cr1, Y3, Cb2, ... . Three-plane destination image has the following order of pointers: Y-plane, Cb-plane, Cr-plane. Two-plane destination image contains luminance samples Y0, Y1, Y2, ... in the first plane *pDstY*, and interleaved chrominance samples Cb0, Cr0, Cb1, Cr1, ... in the second plane *pDstCbCr*.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if any of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if any field of the <i>roiSize</i> is less than 2.

## CbYCr422ToYCbCr420\_Interlace

*Converts interlaced 4:2:2 CbYCr image to 4:2:0 YCbCr image.*

---

## Syntax

```
IppStatus ippiCbYCr422ToYCbCr420_Interlace_8u_C2P3R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);
```

## Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Array of pointers to the ROI in each plane for destination image.
<i>dstStep</i>	Array of distances in bytes between starts of consecutive lines in each plane for a three-plane destination image.
<i>roiSize</i>	Size of the source and destination ROI in pixels, its width must be multiple of 2, and height must be multiple of 4.

## Description

The function `ippiCbYCr422ToYCbCr420_Interlace` is declared in the `ippcc.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#) ).

This function converts interlaced 4:2:2 CbYCr two-channel image `pSrc` to the 4:2:0 YCbCr three-plane image `pDst` (see [Table "Pixel-Order Image Formats"](#) and [Table "Planar Image Formats"](#)). The source image has the following sequence of samples: Cb0, Y0, Cr0, Y1, Cb1, Y2, Cr1, Y3, Cb2, ... . Three-plane destination image has the following order of pointers: Y-plane, Cb-plane, Cr-plane.

The conversion is performed in accordance with the following formulas:

$$Y1_{\text{dest}} = Y1_{\text{src}};$$

$$Cb0(Cr0)_{\text{dest}} = (3 * Cb0(Cr0)_{\text{src}} + Cb2(Cr2)_{\text{src}} + 2) / 4;$$

$$Cb1(Cr1)_{\text{dest}} = (Cb1(Cr1)_{\text{src}} + 3 * Cb3(Cr3)_{\text{src}} + 2) / 4;$$

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if any of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <code>roiSize.width</code> is less than 2, or <code>roiSize.height</code> is less than 4.
<code>ippStsDoubleSize</code>	Indicates a warning if <code>roiSize.width</code> is not multiple of 2, or <code>roiSize.height</code> is not multiple of 4.

## CbYCr422ToYCrCb420

*Converts 4:2:2 CbYCr image to 4:2:0 YCrCb image.*

## Syntax

```
ippStatus ippiCbYCr422ToYCrCb420_8u_C2P3R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst[3],
int dstStep[3], IppiSize roiSize);
```

## Parameters

<code>pSrc</code>	Pointer to the source image ROI.
<code>srcStep</code>	Distance in bytes between starts of consecutive lines in the source image.
<code>pDst</code>	Array of pointers to the ROI in each plane of the destination image.
<code>dstStep</code>	Array of distances in bytes between starts of consecutive lines in the destination image planes.
<code>roiSize</code>	Size of the source and destination ROI in pixels, height and width should be multiple of 2.

## Description

The function `ippiCbYCr422ToYCrCb420` is declared in the `ippcc.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts the 4:2:2 CbYCr two-channel image *pSrc* to the 4:2:0 YCrCb three-plane image *pDst*. The source image has the following sequence of samples: Cb0, Y0, Cr0, Y1, Cb1, Y2, Cr1, Y3, Cb2, ... . The destination image has the following order of pointers: Y-plane, Cr-plane, Cb-plane (see [Table "Pixel-Order Image Formats"](#) and [Table "Planar Image Formats"](#)).

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if any of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if any field of the <i>roiSize</i> is less than 2.

## CbYCr422ToYCbCr411

Converts 4:2:2 CbYCr image to 4:1:1 YCbCr image.

### Syntax

```
IppStatus ippCbYCr422ToYCbCr411_8u_C2P3R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst[3],
int dstStep[3], IppiSize roiSize);
```

### Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Array of pointers to the ROI in each plane of the destination image.
<i>dstStep</i>	Array of distances in bytes between starts of consecutive lines in the destination image planes.
<i>roiSize</i>	Size of the ROI in pixels, its width should be multiple of 4.

### Description

The function `ippCbYCr422ToYCbCr411_8u_C2P3R` are declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts the 4:2:2 CbYCr two-channel image *pSrc* to the 4:1:1 YCbCr three-plane image *pDst*. The source image has the following sequence of samples: Cb0, Y0, Cr0, Y1, Cb1, Y2, Cr1, Y3, Cb2, ... . The destination image has the following order of pointers: Y-plane, Cb-plane, Cr-plane (see [Table "Pixel-Order Image Formats"](#) and [Table "Planar Image Formats"](#)).

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if any of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize.width</i> is less than 4.

## YCbCr420

*Converts 4:2:0 YCbCr image.*

---

### Syntax

```
IppStatus ippiYCbCr420_8u_P3P2R(const Ipp8u* pSrc[3], int srcStep[3], Ipp8u* pDstY, int
dstYStep, Ipp8u* pDstCbCr, int dstCbCrStep, IppiSize roiSize);

IppStatus ippiYCbCr420_8u_P2P3R(const Ipp8u* pSrcY, int srcYStep, const Ipp8u* pSrcCbCr,
int srcCbCrStep, Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);
```

### Parameters

<i>pSrc</i>	Array of pointers to the ROI in each plane for a three-plane source image.
<i>srcStep</i>	Array of distances in bytes between starts of consecutive lines in each plane for a three-plane source image.
<i>pSrcY</i>	Pointer to the ROI in the luminance plane for a two-plane source image.
<i>srcYStep</i>	Distance in bytes between starts of consecutive lines in the luminance plane of a source image.
<i>pSrcCbCr</i>	Pointer to the ROI in the interleaved chrominance plane for a two-plane source image.
<i>srcCbCrStep</i>	Distance in bytes between starts of consecutive lines in the interleaved chrominance plane of the source image.
<i>pDst</i>	Array of pointers to the ROI in each plane for a three-plane destination image.
<i>dstStep</i>	Array of distances in bytes between starts of consecutive lines in each plane for a three-plane destination image.
<i>pDstY</i>	Pointer to the ROI in the luminance plane for a two-plane destination image.
<i>dstYStep</i>	Distance in bytes between starts of consecutive lines in the luminance plane of a destination image.
<i>pDstCbCr</i>	Pointer to the ROI in the interleaved chrominance plane for a two-plane destination image.
<i>dstCbCrStep</i>	Distance in bytes between starts of consecutive lines in the chrominance plane of a destination image.
<i>roiSize</i>	Size of the source and destination ROI in pixels, height and width should be multiple of 2.

### Description

The function `ippiYCbCr420` is declared in the `ippcc.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts the 4:2:0 three-plane (see [Table “Planar Image Formats”](#)) source image *pSrc* to the 4:2:0 two-plane image and vice versa. Two-plane image contains luminance samples Y0, Y1, Y2, .. in the first plane, and interleaved chrominance samples Cb0, Cr0, Cb1, Cr1,... in the second plane.

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if any of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if any field of the <i>roiSize</i> is less than 2.

## YCbCr420ToYCbCr422

*Converts YCbCr image from 4:2:0 sampling format to 4:2:2 format.*

---

### Syntax

```
IppStatus ippYCbCr420ToYCbCr422_8u_P3R(const Ipp8u* pSrc[3], int srcStep[3], Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);
```

```
IppStatus ippYCbCr420ToYCbCr422_8u_P2P3R(const Ipp8u* pSrcY, int srcYStep, const Ipp8u* pSrcCbCr, int srcCbCrStep, Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);
```

```
IppStatus ippYCbCr420ToYCbCr422_8u_P2C2R(const Ipp8u* pSrcY, int srcYStep, const Ipp8u* pSrcCbCr, int srcCbCrStep, Ipp8u* pDst, int dstStep, IppiSize roiSize);
```

### Parameters

<i>pSrc</i>	Array of pointers to the ROI in each plane for a three-plane source image.
<i>srcStep</i>	Array of distances in bytes between starts of consecutive lines in each plane for a three-plane source image.
<i>pSrcY</i>	Pointer to the ROI in the luminance plane for a two-plane source image.
<i>srcYStep</i>	Distance in bytes between starts of consecutive lines in the luminance plane of a source image.
<i>pSrcCbCr</i>	Pointer to the ROI in the interleaved chrominance plane for a two-plane source image.
<i>srcCbCrStep</i>	Distance in bytes between starts of consecutive lines in the interleaved chrominance plane of the source image.
<i>pDst</i>	Pointer to the ROI in the pixel-order destination image. Array of pointers to the ROI in each plane of the planar destination image.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image. Array of distance values for the destination image planes.
<i>roiSize</i>	Size of the ROI in pixels, height and width should be multiple of 2.



## Description

The function `ippiYCbCr420ToYCbCr422` is declared in the `ippcc.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts the 4:2:0 planar source image `pSrc` to the 4:2:2 image `pDst`. The source image can be two- or three-plane image (see [Table "Planar Image Formats"](#)). The first plane of the two-plane source image `pSrcY` contains luminance samples `Y0, Y1, Y2, ...`, the second plane `pSrcCbCr` contains interleaved chrominance samples `Cb0, Cr0, Cb1, Cr1, ...`. The destination image `pDst` can be three-plane or two-channel (see [Table "Pixel-Order Image Formats"](#) and [Table "Planar Image Formats"](#)).

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if any of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if any field of the <code>roiSize</code> is less than 2.

## YCbCr420ToYCbCr422\_Filter

*Convert 4:2:0 image to 4:2:2 image with additional filtering.*

## Syntax

```
IppStatus ippiYCbCr420ToYCbCr422_Filter_8u_P3R(const Ipp8u* pSrc[3], int srcStep[3], Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);
```

```
IppStatus ippiYCbCr420ToYCbCr422_Filter_8u_P2P3R(const Ipp8u* pSrcY, int srcYStep, const Ipp8u* pSrcCbCr, int srcCbCrStep, Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);
```

```
IppStatus ippiYCbCr420ToYCbCr422_Filter_8u_P2C2R(const Ipp8u* pSrcY, int srcYStep, const Ipp8u* pSrcCbCr, int srcCbCrStep, Ipp8u* pDst, int dstStep, IppiSize roiSize, int layout);
```

## Parameters

<code>pSrc</code>	Array of pointers to the ROI in each plane for a three-plane source image.
<code>srcStep</code>	Array of distances in bytes between starts of consecutive lines in each plane for a three-plane source image.
<code>pSrcY</code>	Pointer to the ROI in the luminance plane for a two-plane source image.
<code>srcYStep</code>	Distance in bytes between starts of consecutive lines in the luminance plane of a source image.
<code>pSrcCbCr</code>	Pointer to the ROI in the interleaved chrominance plane for a two-plane source image.
<code>srcCbCrStep</code>	Distance in bytes between starts of consecutive lines in the interleaved chrominance plane of the source image.

<i>pDst</i>	Pointer to the ROI in the pixel-order destination image. Array of pointers to the ROI in each plane of the planar destination image.												
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image. Array of distance values for the destination image planes.												
<i>roiSize</i>	Size of the ROI in pixels.												
<i>layout</i>	Slice layout. Possible values: <table> <tr> <td>IPP_UPPER</td><td>Upper (first) slice</td></tr> <tr> <td>IPP_CENTER</td><td>Middle slices</td></tr> <tr> <td>IPP_LOWER</td><td>Lowermost (last) slice</td></tr> <tr> <td>IPP_LOWER &amp;&amp;</td><td>Image is not sliced</td></tr> <tr> <td>IPP_UPPER &amp;&amp;</td><td></td></tr> <tr> <td>IPP_CENTER</td><td></td></tr> </table>	IPP_UPPER	Upper (first) slice	IPP_CENTER	Middle slices	IPP_LOWER	Lowermost (last) slice	IPP_LOWER &&	Image is not sliced	IPP_UPPER &&		IPP_CENTER	
IPP_UPPER	Upper (first) slice												
IPP_CENTER	Middle slices												
IPP_LOWER	Lowermost (last) slice												
IPP_LOWER &&	Image is not sliced												
IPP_UPPER &&													
IPP_CENTER													

## Description

The function `ippiYCbCr420ToYCbCr422_Filter` is declared in the `ippcc.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts the 4:2:0 planar source image *pSrc* to the 4:2:2 image *pDst* and performs additional filtering. The source image can be two- or three-plane image (see [Table "Planar Image Formats"](#)). The first plane of the two-plane source image *pSrcY* contains luminance samples *Y0, Y1, Y2, ...*, the second plane *pSrcCbCr* contains interleaved chrominance samples *Cb0, Cr0, Cb1, Cr1, ...*. The destination image *pDst* can be three-plane or two-channel (see [Table "Pixel-Order Image Formats"](#) and [Table "Planar Image Formats"](#)).

The function flavors `ippiYCbCr420ToYCbCr422_Filter_8u_P3R` and `ippiYCbCr420ToYCbCr422_Filter_8u_P2P3R` additionally perform the vertical upsampling using a Catmull-Rom interpolation (cubic convolution interpolation). In this case *roiSize.width* should be multiple of 2, and *roiSize.height* should be multiple of 8.

The function `ippiYCbCr420ToYCbCr422_Filter_8u_P2C2R` additionally performs deinterlace filtering. Commonly it is used to process images that are divided into slices. In this case slice *layout* should be specified, since the function processes the first (upper), last (lowermost), and intermediate (middle) slices differently. The height of slices should be a multiple of 16.




---

**CAUTION.** The image slices should be processed exactly in the following order: the first slice, intermediate slices, the last slice.

---

The function may be applied to a not-sliced image as well. In this case *roiSize.width* and *roiSize.height* should be multiple of 2.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if any of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize.width</i> has wrong value.

## YCbCr420To422\_Interlace

Converts interlaced YCbCr image from 4:2:0 sampling format to 4:2:2 format.

### Syntax

```
IppStatus ippiYCbCr420To422_Interlace_8u_P3R(const Ipp8u* pSrc[3], int srcStep[3], Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);
```

### Parameters

<i>pSrc</i>	Array of pointers to the ROI in each plane for source image.
<i>srcStep</i>	Array of distances in bytes between starts of consecutive lines in each plane for the source image.
<i>pDst</i>	Array of pointers to the ROI in each plane for destination image.
<i>dstStep</i>	Array of distances in bytes between starts of consecutive lines in each plane for the destination image.
<i>roiSize</i>	Size of the source and destination ROI in pixels, its width must be multiple of 2, and height must be multiple of 4.

### Description

The function `ippiYCbCr420To422_Interlace` is declared in the `ippcc.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts the interlaced planar 4:2:0 source image *pSrc* to the 4:2:2 image *pDst*. Three-plane image has the following order of pointers: Y-plane, Cb-plane, Cr-plane.

The conversion is performed in accordance with the following formulas:

$$Y_{n_{\text{dest}}} = Y_{n_{\text{src}}};$$

$$Cb0(Cr0)_{\text{dest}} = (5 * Cb0(Cr0)_{\text{src}} + 3 * Cb2(Cr2)_{\text{src}} + 4) / 8;$$

$$Cb1(Cr1)_{\text{dest}} = (7 * Cb1(Cr1)_{\text{src}} + Cb3(Cr3)_{\text{src}} + 4) / 8;$$

$$Cb2(Cr2)_{\text{dest}} = (Cb0(Cr0)_{\text{src}} + 7 * Cb2(Cr2)_{\text{src}} + 4) / 8;$$

$$Cb3(Cr3)_{\text{dest}} = (3 * Cb1(Cr1)_{\text{src}} + 5 * Cb3(Cr3)_{\text{src}} + 4) / 8;$$

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if any of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize.width</i> is less than 2, or <i>roiSize.height</i> is less than 4.
<code>ippStsDoubleSize</code>	Indicates a warning if <i>roiSize.width</i> is not multiple of 2, or <i>roiSize.height</i> is not multiple of 4.

## YCbCr420ToCbYCr422

*Converts 4:2:0 YCbCr image to 4:2:2 CbYCr image.*

---

### Syntax

```
ippStatus ippiYCbCr420ToCbYCr422_8u_P2C2R(const Ipp8u* pSrcY, int srcYStep, const Ipp8u*
pSrcCbCr, int srcCbCrStep, Ipp8u* pDst, int dstStep, IppiSize roiSize);
```

### Parameters

<i>pSrcY</i>	Pointer to the ROI in the luminance plane of the source image.
<i>srcYStep</i>	Distance in bytes between starts of consecutive lines in the luminance plane of the source image.
<i>pSrcCbCr</i>	Pointer to the ROI in the interleaved chrominance plane of the source image.
<i>srcCbCrStep</i>	Distance in bytes between starts of consecutive lines in the interleaved chrominance plane of the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>roiSize</i>	Size of the source and destination ROI in pixels, height and width should be multiple of 2.

### Description

The function `ippiYCbCr420ToCbYCr422` is declared in the `ippcc.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts the planar 4:2:0 two-plane source image to the pixel-order 4:2:2 two-channel image. The first plane of the source image *pSrcY* contains luminance samples *Y0, Y1, Y2, ..*, the second plane *pSrcCbCr* contains interleaved chrominance samples *Cb0, Cr0, Cb1, Cr1, ...*. The destination image *pDst* has the following sequence of samples: *Cb0, Y0, Cr0, Y1, Cb1, Y2, Cr1, Y3, Cb2, ...*

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if any of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if any field of the <i>roiSize</i> is less than 2.

## YCbCr420ToCbYCr422\_Interlace

*Converts interlaced 4:2:0 YCbCr image to 4:2:2 CbYCr image.*

---

### Syntax

```
ippStatus ippiYCbCr420ToCbYCr422_Interlace_8u_P3C2R(const Ipp8u* pSrc[3], int srcStep[3],
Ipp8u* pDst, int dstStep, IppiSize roiSize);
```

## Parameters

<i>pSrc</i>	Array of pointers to the ROI in each plane for source image.
<i>srcStep</i>	Array of distances in bytes between starts of consecutive lines in each plane for the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>roiSize</i>	Size of the source and destination ROI in pixels, its width must be multiple of 2, and height must be multiple of 4.

## Description

The function `ippiYCbCr420ToCbYCr422_Interlace` is declared in the `ippcc.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts the interlaced planar 4:2:0 image to the pixel-order 4:2:2 two-channel image. Three-plane source image has the following order of pointers: Y-plane, Cb-plane, Cr-plane. The destination image `pDst` has the following sequence of samples: Cb0, Y0, Cr0, Y1, Cb1, Y2, Cr1, Y3, Cb2, ...

The conversion is performed in accordance with the following formulas:

```

Yn_dest = Yn_src;
Cb0(Cr0)_dest = (5*Cb0(Cr0)_src + 3*Cb2(Cr2)_src + 4)/8;
Cb1(Cr1)_dest = (7*Cb1(Cr1)_src + Cb3(Cr3)_src + 4)/8;
Cb2(Cr2)_dest = (Cb0(Cr0)_src + 7*Cb2(Cr2)_src + 4)/8;
Cb3(Cr3)_dest = (3*Cb1(Cr1)_src + 5*Cb3(Cr3)_src + 4)/8;

```

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if any of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <code>roiSize.width</code> is less than 2, or <code>roiSize.height</code> is less than 4.
<code>ippStsDoubleSize</code>	Indicates a warning if <code>roiSize.width</code> is not multiple of 2, or <code>roiSize.height</code> is not multiple of 4.

## YCbCr420ToYCrCb420

Converts 4:2:0 YCbCr image to 4:2:0 YCrCb image.

### Syntax

```

IppStatus ippiYCbCr420ToYCrCb420_8u_P2P3R(const Ipp8u* pSrcY, int srcYStep, const Ipp8u*
pSrcCbCr, int srcCbCrStep, Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);

```

## Parameters

<i>pSrcY</i>	Pointer to the ROI in the luminance plane of the source image.
<i>srcYStep</i>	Distance in bytes between starts of consecutive lines in the luminance plane of the source image.
<i>pSrcCbCr</i>	Pointer to the ROI in the interleaved chrominance plane of the source image.
<i>srcCbCrStep</i>	Distance in bytes between starts of consecutive lines in the interleaved chrominance plane of the source image.
<i>pDst</i>	Array of pointers to the ROI in each plane of the destination image.
<i>dstStep</i>	Array of distances in bytes between starts of consecutive lines in the destination image planes.
<i>roiSize</i>	Size of the source and destination ROI in pixels, height and width should be multiple of 2.

## Description

The function `ippiYCbCr420ToYCrCb420` is declared in the `ippcc.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts the 4:2:0 two-plane source image *pSrc* to the 4:2:0 three-plane image *pDst*. The first plane of the source image *pSrcY* contains luminance samples  $Y_0, Y_1, Y_2, \dots$ , the second plane *pSrcCbCr* contains interleaved chrominance samples  $Cb_0, Cr_0, Cb_1, Cr_1, \dots$ . The destination image *pDst* has the following order of pointers: Y-plane, Cr-plane, Cb-plane (see [Table "Planar Image Formats"](#)).

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if any of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if any field of the <i>roiSize</i> is less than 2.

## YCbCr420ToYCrCb420\_Filter

*Convert 4:2:0 YCbCr image to 4:2:0 YCrCb image with deinterlace filtering.*

---

## Syntax

```
IppStatus ippiYCbCr420ToYCrCb420_Filter_8u_P2P3R(const Ipp8u* pSrcY, int srcYStep, const Ipp8u* pSrcCbCr, int srcCbCrStep, Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize, int layout);
```

## Parameters

<i>pSrcY</i>	Pointer to the ROI in the luminance plane of the source image.
<i>srcYStep</i>	Distance in bytes between starts of consecutive lines in the luminance plane of the source image.

<i>pSrcCbCr</i>	Pointer to the ROI in the interleaved chrominance plane of the source image.												
<i>srcCbCrStep</i>	Distance in bytes between starts of consecutive lines in the interleaved chrominance plane of the source image.												
<i>pDst</i>	Array of pointers to the ROI in each plane of the destination image.												
<i>dstStep</i>	Array of distances in bytes between starts of consecutive lines in the destination image planes.												
<i>roiSize</i>	Size of the source and destination ROI in pixels, height and width should be multiple of 2.												
<i>layout</i>	Slice layout. Possible values: <table> <tr> <td>IPP_UPPER</td><td>Upper (first) slice</td></tr> <tr> <td>IPP_CENTER</td><td>Middle slices</td></tr> <tr> <td>IPP_LOWER</td><td>Lowermost (last) slice</td></tr> <tr> <td>IPP_LOWER &amp;&amp;</td><td>Image is not sliced</td></tr> <tr> <td>IPP_UPPER &amp;&amp;</td><td></td></tr> <tr> <td>IPP_CENTER</td><td></td></tr> </table>	IPP_UPPER	Upper (first) slice	IPP_CENTER	Middle slices	IPP_LOWER	Lowermost (last) slice	IPP_LOWER &&	Image is not sliced	IPP_UPPER &&		IPP_CENTER	
IPP_UPPER	Upper (first) slice												
IPP_CENTER	Middle slices												
IPP_LOWER	Lowermost (last) slice												
IPP_LOWER &&	Image is not sliced												
IPP_UPPER &&													
IPP_CENTER													

## Description

The functions `ippiYCbCr420ToYCrCb420_Filter` is declared in the `ippcc.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts the 4:2:0 two-plane source image to the 4:2:0 three-plane image. The first plane of the source image *pSrcY* contains luminance samples *Y0*, *Y1*, *Y2*, ..., the second plane *pSrcCbCr* contains interleaved chrominance samples *Cb0*, *Cr0*, *Cb1*, *Cr1*, .... The destination image *pDst* has the following order of pointers: Y-plane, Cr-plane, Cb-plane. The function additionally performs deinterlace filtering. Commonly it is used to process sliced images. In this case the slice *layout* should be specified, since the function processes the first (upper), last (lowermost), and intermediate (middle) slices differently. The height of slices should be a multiple of 16. The function may be applied to a not-sliced image as well.



**CAUTION.** The image slices should be processed exactly in the following order: the first slice, intermediate slices, the last slice.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if any of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if any field of the <i>roiSize</i> is less than 2.

## YCbCr420ToYCbCr411

*Converts YCbCr image from 4:2:0 sampling format to 4:1:1 format.*

---

### Syntax

```
IppStatus ippiYCbCr420ToYCbCr411_8u_P3P2R(const Ipp8u* pSrc[3], int srcStep[3], Ipp8u* pDstY, int dstYStep, Ipp8u* pDstCbCr, int dstCbCrStep, IppiSize roiSize);

IppStatus ippiYCbCr420ToYCbCr411_8u_P2P3R(const Ipp8u* pSrcY, int srcYStep, const Ipp8u* pSrcCbCr, int srcCbCrStep, Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);
```

### Parameters

<i>pSrc</i>	Array of pointers to the ROI in each plane for a three-plane source image.
<i>srcStep</i>	Array of distances in bytes between starts of consecutive lines in each plane for a three-plane source image.
<i>pSrcY</i>	Pointer to the ROI in the luminance plane for a two-plane source image.
<i>srcYStep</i>	Distance in bytes between starts of consecutive lines in the luminance plane of a source image.
<i>pSrcCbCr</i>	Pointer to the ROI in the interleaved chrominance plane for a two-plane source image.
<i>srcCbCrStep</i>	Distance in bytes between starts of consecutive lines in the interleaved chrominance plane of the source image.
<i>pDst</i>	Array of pointers to the ROI in each plane for a three-plane destination image.
<i>dstStep</i>	Array of distances in bytes between starts of consecutive lines in each plane for a three-plane destination image.
<i>pDstY</i>	Pointer to the ROI in the luminance plane for a two-plane destination image.
<i>dstYStep</i>	Distance in bytes between starts of consecutive lines in the luminance plane of a destination image.
<i>pDstCbCr</i>	Pointer to the ROI in the interleaved chrominance plane for a two-plane destination image.
<i>dstCbCrStep</i>	Distance in bytes between starts of consecutive lines in the chrominance plane of a destination image.
<i>roiSize</i>	Size of the ROI in pixels, its width should be multiple of 4, its height should be multiple of 2.

### Description

The function `ippiYCbCr420ToYCbCr411` is declared in the `ippcc.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).



This function converts the 4:2:0 source image to the 4:1:1 destination image. The source two-plane image is converted to destination three-plane image and vice versa. The first plane of the two-plane image contains luminance samples  $Y_0, Y_1, Y_2, \dots$ , the second plane contains interleaved chrominance samples  $Cb_0, Cr_0, Cb_1, Cr_1, \dots$ . The three-plane image has the following order of pointers: Y-plane, Cb-plane, Cr-plane (see [Table “Planar Image Formats”](#)).

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if any of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <code>roiSize.width</code> is less than 4 or <code>roiSize.height</code> is less than 2.

## YCrCb420ToYCbCr422

*Converts 4:2:0 YCrCb image to 4:2:2 YCbCr image.*

## Syntax

```
IppStatus ippYCrCb420ToYCbCr422_8u_P3R(const Ipp8u* pSrc[3], int srcStep[3], Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);
```

```
IppStatus ippYCrCb420ToYCbCr422_8u_P3C2R(const Ipp8u* pSrc[3], int srcStep[3], Ipp8u* pDst, int dstStep, IppiSize roiSize);
```

## Parameters

<code>pSrc</code>	Array of pointers to the ROI in each plane of the source image.
<code>srcStep</code>	Array of distances in bytes between starts of consecutive lines in each plane of the source image.
<code>pDst</code>	Pointer to the ROI in the pixel-order destination image. Array of pointers to the ROI in each plane of the planar destination image.
<code>dstStep</code>	Distance in bytes between starts of consecutive lines in the destination image. Array of distance values for the destination image planes.
<code>roiSize</code>	Size of the ROI in pixels, height and width should be multiple of 2.

## Description

The function `ippYCrCb420ToYCbCr422` is declared in the `ippcc.h` file. This function converts the 4:2:0 YCrCb three-plane image `pSrc` to the 4:2:2 YCbCr three-plane or two-channel image `pDst` (see [Table “Pixel-Order Image Formats”](#) and [Table “Planar Image Formats”](#)).

This function operates with ROI (see [Regions of Interest in Intel IPP](#)).

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
--------------------------	---

<code>ippStsNullPtrErr</code>	Indicates an error condition if any of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if any field of the <code>roiSize</code> is less than 2.

## YCrCb420ToYCbCr422\_Filter

Converts 4:2:0 YCrCb image to 4:2:2 YCbCr image with additional filtering.

---

### Syntax

```
IppStatus ippYCrCb420ToYCbCr422_Filter_8u_P3R(const Ipp8u* pSrc[3], int srcStep[3], Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);
```

### Parameters

<code>pSrc</code>	Array of pointers to the ROI in each plane of the source image.
<code>srcStep</code>	Array of distances in bytes between starts of consecutive lines in each plane of the source image.
<code>pDst</code>	Array of pointers to the ROI in each plane of the destination image.
<code>dstStep</code>	Array of distances in bytes between starts of consecutive lines in each plane of the destination image.
<code>roiSize</code>	Size of the ROI in pixels, its width should be multiple of 2, its height should be multiple of 8.

### Description

The function `ippYCrCb420ToYCbCr422_Filter` is declared in the `ippcc.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts the 4:2:0 YCrCb three-plane image `pSrc` to the 4:2:2 YCbCr three-plane image `pDst` (see [Table "Planar Image Formats"](#)).

Additionally, this function performs the vertical upsampling using a Catmull-Rom interpolation (cubic convolution interpolation).

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if any of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <code>roiSize.width</code> is less than 2 or <code>roiSize.height</code> is less than 8.

## YCrCb420ToCbYCr422

Converts 4:2:0 YCrCb image to 4:2:2 CbYCr image.

### Syntax

```
IppStatus ippiYCrCb420ToCbYCr422_8u_P3C2R(const Ipp8u* pSrc[3], int srcStep[3], Ipp8u* pDst, int dstStep, IppiSize roiSize);
```

### Parameters

<i>pSrc</i>	Array of pointers to the ROI in each plane of the source image.
<i>srcStep</i>	Array of distances in bytes between starts of consecutive lines in each plane of the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>roiSize</i>	Size of the source and destination ROI in pixels, height and width should be multiple of 2.

### Description

The function `ippiYCrCb420ToCbYCr422` is declared in the `ippcc.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts the 4:2:0 YCrCb three-plane image *pSrc* (see [Table "Planar Image Formats"](#)) to the 4:2:2 CbYCr two-channel image *pDst* with the following sequence of samples: Cb0, Y0, Cr0, Y1, Cb1, Y2, Cr1, Y3, Cb2, ... (see [Table "Pixel-Order Image Formats"](#)).

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if any of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if any field of the <i>roiSize</i> is less than 2.

## YCrCb420ToYCbCr420

Converts 4:2:0 YCrCb image to 4:2:0 YCbCr image.

### Syntax

```
IppStatus ippiYCrCb420ToYCbCr420_8u_P3P2R(const Ipp8u* pSrc[3], int srcStep[3], Ipp8u* pDstY, int dstYStep, Ipp8u* pDstCbCr, int dstCbCrStep, IppiSize roiSize);
```

### Parameters

<i>pSrc</i>	Array of pointers to the ROI in each plane of the source image.
<i>srcStep</i>	Array of distances in bytes between starts of consecutive lines in each plane of the source image.

<i>pDstY</i>	Pointer to the ROI in the luminance plane of a destination image.
<i>dstYStep</i>	Distance in bytes between starts of consecutive lines in the luminance plane of the destination image.
<i>pDstCbCr</i>	Pointer to the ROI in the interleaved chrominance plane of the destination image.
<i>dstCbCrStep</i>	Distance in bytes between starts of consecutive lines in the interleaved chrominance plane of a destination image.
<i>roiSize</i>	Size of the source and destination ROI in pixels, height and width should be multiple of 2.

## Description

The function `ippiYCrCb420ToYCbCr420` is declared in the `ippcc.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts the 4:2:0 YCrCb three-plane image *pSrc* (see [Table "Planar Image Formats"](#)) to the 4:2:0 YCbCr two-plane image that contains luminance samples Y0, Y1, Y2, .. in the first plane *pDstY*, and interleaved chrominance samples Cb0, Cr0, Cb1, Cr1, ... in the second plane *pDstCbCr*.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if any of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if any field of the <i>roiSize</i> is less than 2.

## YCrCb420ToYCbCr411

Converts 4:2:0 YCrCb image to 4:1:1 YCbCr image.

## Syntax

```
ippStatus ippiYCrCb420ToYCbCr411_8u_P3P2R(const Ipp8u* pSrc[3], int srcStep[3], Ipp8u* pDstY, int dstYStep, Ipp8u* pDstCbCr, int dstCbCrStep, IppiSize roiSize);
```

## Parameters

<i>pSrc</i>	Array of pointers to the ROI in each plane of the source image.
<i>srcStep</i>	Array of distances in bytes between starts of consecutive lines in each plane of the source image.
<i>pDstY</i>	Pointer to the ROI in the luminance plane of a destination image.
<i>dstYStep</i>	Distance in bytes between starts of consecutive lines in the luminance plane of the destination image.
<i>pDstCbCr</i>	Pointer to the ROI in the interleaved chrominance plane of the destination image.
<i>dstCbCrStep</i>	Distance in bytes between starts of consecutive lines in the interleaved chrominance plane of a destination image.

*roiSize* Size of the ROI in pixels, its width should be multiple of 4, its height should be multiple of 2.

## Description

The function `ippiYCrCb420ToYCbCr411` is declared in the `ippcc.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts the `4:2:0` YCrCb three-plane image *pSrc* (see [Table "Planar Image Formats"](#)) to the `4:1:1` YCbCr two-plane image that contains luminance samples *Y0*, *Y1*, *Y2*, .. in the first plane *pDstY*, and interleaved chrominance samples *Cb0*, *Cr0*, *Cb1*, *Cr1*, ... in the second plane *pDstCbCr*..

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if any of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize.width</i> is less than 4 or <i>roiSize.height</i> is less than 2.

## YCbCr411

*Converts 4:1:1 YCbCr image.*

## Syntax

```
IppStatus ippiYCbCr411_8u_P3P2R(const Ipp8u* pSrc[3], int srcStep[3], Ipp8u* pDstY, int dstYStep, Ipp8u* pDstCrCb, int dstCbCrStep, IppiSize roiSize);
```

```
IppStatus ippiYCbCr411_8u_P2P3R(const Ipp8u* pSrcY, int srcYStep, const Ipp8u* pSrcCrCb, int srcCrCbStep, Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);
```

## Parameters

<i>pSrc</i>	Array of pointers to the ROI in each plane for a three-plane source image.
<i>srcStep</i>	Array of distances in bytes between starts of consecutive lines in each plane for a three-plane source image.
<i>pSrcY</i>	Pointer to the ROI in the luminance plane for a two-plane source image.
<i>srcYStep</i>	Distance in bytes between starts of consecutive lines in the luminance plane of a source image.
<i>pSrcCbCr</i>	Pointer to the ROI in the interleaved chrominance plane for a two-plane source image.
<i>srcCbCrStep</i>	Distance in bytes between starts of consecutive lines in the interleaved chrominance plane of the source image.
<i>pDst</i>	Array of pointers to the ROI in each plane for a three-plane destination image.
<i>dstStep</i>	Array of distances in bytes between starts of consecutive lines in each plane for a three-plane destination image.

<i>pDstY</i>	Pointer to the ROI in the luminance plane for a two-plane destination image.
<i>dstYStep</i>	Distance in bytes between starts of consecutive lines in the luminance plane of a destination image.
<i>pDstCbCr</i>	Pointer to the ROI in the interleaved chrominance plane for a two-plane destination image.
<i>dstCbCrStep</i>	Distance in bytes between starts of consecutive lines in the chrominance plane of a destination image.
<i>roiSize</i>	Size of the ROI in pixels, its width should be multiple of 4.

### Description

The function `ippiYCbCr411` is declared in the `ippcc.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts the 4:1:1 three-plane (see [Table "Planar Image Formats"](#)) source image *pSrc* to the 4:1:1 two-plane image and vice versa. Two-plane image contains luminance samples Y0, Y1, Y2, .. in the first plane, and interleaved chrominance samples Cb0, Cr0, Cb1, Cr1, ... in the second plane.

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if any of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize.width</i> is less than 4.

## YCbCr411ToYCbCr422

Converts 4:1:1 YCbCr image to 4:2:2 YCbCr image.

### Syntax

```
ippStatus ippiYCbCr411ToYCbCr422_8u_P3R(const Ipp8u* pSrc[3], int srcStep[3], Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);
```

```
ippStatus ippiYCbCr411ToYCbCr422_8u_P3C2R(const Ipp8u* pSrc[3], int srcStep[3], Ipp8u* pDst, int dstStep, IppiSize roiSize);
```

```
ippStatus ippiYCbCr411ToYCbCr422_8u_P2P3R(const Ipp8u* pSrcY, int srcYStep, const Ipp8u* pSrcCbCr, int srcCbCrStep, Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);
```

```
ippStatus ippiYCbCr411ToYCbCr422_8u_P2C2R(const Ipp8u* pSrcY, int srcYStep, const Ipp8u* pSrcCrCb, int srcCrCbStep, Ipp8u* pDst, int dstStep, IppiSize roiSize);
```

### Parameters

<i>pSrc</i>	Array of pointers to the ROI in each plane for a three-plane source image.
<i>srcStep</i>	Array of distances in bytes between starts of consecutive lines in each plane for a three-plane source image.

<i>pSrcY</i>	Pointer to the ROI in the luminance plane for a two-plane source image.
<i>srcYStep</i>	Distance in bytes between starts of consecutive lines in the luminance plane of a source image.
<i>pSrcCbCr</i>	Pointer to the ROI in the interleaved chrominance plane for a two-plane source image.
<i>srcCbCrStep</i>	Distance in bytes between starts of consecutive lines in the interleaved chrominance plane of the source image.
<i>pDst</i>	Pointer to the ROI in the pixel-order destination image. Array of pointers to the ROI in each plane of the planar destination image.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image. Array of distance values for the destination image planes.
<i>roiSize</i>	Size of the ROI in pixels, its width should be multiple of 4.

## Description

The function `ippiYCbCr411ToYCbCr422` is declared in the `ippcc.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts 4:1:1 planar source image *pSrc* to the 4:2:2 image *pDst*. The first plane of the two-plane source image *pSrcY* contains luminance samples *Y0*, *Y1*, *Y2*, ..., the second plane *pSrcCbCr* contains interleaved chrominance samples *Cb0*, *Cr0*, *Cb1*, *Cr1*, .... The destination image *pDst* can be either three-plane (see [Table "Planar Image Formats"](#)) or two-channel image (see [Table "Pixel-Order Image Formats"](#)).

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if any of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize.width</i> is less than 4.

## YCbCr411ToYCrCb422

Converts 4:1:1 YCbCr image to 4:2:2 YCrCb image.

## Syntax

```
ippStatus ippiYCbCr411ToYCrCb422_8u_P3R(const Ipp8u* pSrc[3], int srcStep[3], Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);
```

```
ippStatus ippiYCbCr411ToYCrCb422_8u_P3C2R(const Ipp8u* pSrc[3], int srcStep[3], Ipp8u* pDst, int dstStep, IppiSize roiSize);
```

## Parameters

<i>pSrc</i>	Array of pointers to the ROI in each plane for a three-plane source image.
-------------	--

<i>srcStep</i>	Array of distances in bytes between starts of consecutive lines in each plane for a three-plane source image.
<i>pDst</i>	Pointer to the ROI in the pixel-order destination image. Array of pointers to the ROI in each plane of the planar destination image.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image. Array of distance values for the destination image planes.
<i>roiSize</i>	Size of the ROI in pixels, its width should be multiple of 4.

## Description

The function `ippiYCbCr411ToYCbCr422` is declared in the `ippcc.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts the 4:1:1 three-plane image *pSrc* to the 4:2:2 two-channel or three-plane image *pDst* with different order of components. The source image has the following order of pointers: Y-plane, Cb-plane, Cr-plane (see [Table "Planar Image Formats"](#)). The three-plane destination image has the following order of pointers: Y-plane, Cr-plane, Cb-plane (see [Table "Planar Image Formats"](#)), and two-channel destination image has the following sequence of samples: Y0, Cr0, Y1, Cb0, Y2, Cr1, Y3, Cb1, ... (see [Table "Pixel-Order Image Formats"](#)).

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if any of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize.width</i> is less than 4.

## YCbCr411ToYCbCr420

Converts 4:1:1 YCbCr image to 4:2:0 YCbCr image.

## Syntax

```
IppStatus ippiYCbCr411ToYCbCr420_8u_P3R(const Ipp8u* pSrc[3], int srcStep[3], Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);
```

```
IppStatus ippiYCbCr411ToYCbCr420_8u_P3P2R(const Ipp8u* pSrc[3], int srcStep[3], Ipp8u* pDstY, int dstYStep, Ipp8u* pDstCbCr, int dstCbCrStep, IppiSize roiSize);
```

```
IppStatus ippiYCbCr411ToYCbCr420_8u_P2P3R(const Ipp8u* pSrcY, int srcYStep, const Ipp8u* pSrcCbCr, int srcCbCrStep, Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);
```

## Parameters

<i>pSrc</i>	Array of pointers to the ROI in each plane for a three-plane source image.
<i>srcStep</i>	Array of distances in bytes between starts of consecutive lines in each plane for a three-plane source image.



<i>pSrcY</i>	Pointer to the ROI in the luminance plane for a two-plane source image.
<i>srcYStep</i>	Distance in bytes between starts of consecutive lines in the luminance plane of a source image.
<i>pSrcCbCr</i>	Pointer to the ROI in the interleaved chrominance plane for a two-plane source image.
<i>srcCbCrStep</i>	Distance in bytes between starts of consecutive lines in the interleaved chrominance plane of the source image.
<i>pDst</i>	Array of pointers to the ROI in each plane for a three-plane destination image.
<i>dstStep</i>	Array of distances in bytes between starts of consecutive lines in each plane for a three-plane destination image.
<i>pDstY</i>	Pointer to the ROI in the luminance plane for a two-plane destination image.
<i>dstYStep</i>	Distance in bytes between starts of consecutive lines in the luminance plane of a destination image.
<i>pDstCbCr</i>	Pointer to the ROI in the interleaved chrominance plane for a two-plane destination image.
<i>dstCbCrStep</i>	Distance in bytes between starts of consecutive lines in the chrominance plane of a destination image.
<i>roiSize</i>	Size of the ROI in pixels, its width should be multiple of 4, its height should be multiple of 2.

## Description

The function `ippiYCbCr411ToYCbCr420` is declared in the `ippcc.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts the 4:1:1 planar source image *pSrc* (see [Table "Planar Image Formats"](#)) to the 4:2:0 planar image *pDst*. Both source and destination images can be three- or two-plane. Three-plane images has the following order of pointers: Y-plane, Cb-plane, Cr-plane (see [Table "Planar Image Formats"](#)). Two-plane images contain luminance samples *Y0*, *Y1*, *Y2*, .. in the first plane, and interleaved chrominance samples *Cb0*, *Cr0*, *Cb1*, *Cr1*, ... in the second plane.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if any of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize.width</i> is less than 4 or <i>roiSize.height</i> is less than 2.

## YCbCr411ToYCrCb420

*Converts 4:1:1 YCbCr image to 4:2:0 YCrCb image.*

---

### Syntax

```
IppStatus ippiYCbCr411ToYCrCb420_8u_P2P3R(const Ipp8u* pSrcY, int srcYStep, const Ipp8u*
pSrcCbCr, int srcCbCrStep, Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);
```

### Parameters

<i>pSrcY</i>	Pointer to the ROI in the luminance plane of the source image.
<i>srcYStep</i>	Distance in bytes between starts of consecutive lines in the luminance plane of the source image.
<i>pSrcCbCr</i>	Pointer to the ROI in the interleaved chrominance plane of the source image.
<i>srcCbCrStep</i>	Distance in bytes between starts of consecutive lines in the interleaved chrominance plane of the source image.
<i>pDst</i>	Array of pointers to the ROI in each plane of the destination image.
<i>dstStep</i>	Array of distances in bytes between starts of consecutive lines in each plane of the destination image.
<i>roiSize</i>	Size of the ROI in pixels, its width should be multiple of 4, its height should be multiple of 2.

### Description

The function `ippiYCbCr411ToYCrCb420` is declared in the `ippcc.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts the 4:1:1 two-plane source image *pSrc* to the 4:2:0 three-plane image *pDst* with a different order of components. The first plane of the source image *pSrcY* contains luminance samples *Y0*, *Y1*, *Y2*, ..., the second plane *pSrcCbCr* contains interleaved chrominance samples *Cb0*, *Cr0*, *Cb1*, *Cr1*, ... .The destination image has the following order of pointers: Y-plane, Cr-plane, Cb-plane (see [Table "Planar Image Formats"](#)),

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if any of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize.width</i> is less than 4 or <i>roiSize.height</i> is less than 2.

## Color Twist

Color twist conversion functions use values of all color channels of a source pixel to compute the resultant destination channel value. The destination channel value is obtained as the result of multiplying the corresponding row of the color-twist matrix by the vector of source pixel channel values.

For example, if  $(r, g, b)$  is a source pixel, then the destination pixel values  $(R, G, B)$  are computed as follows:

$$R = t_{11} * r + t_{12} * g + t_{13} * b + t_{14}$$

$$G = t_{21} * r + t_{22} * g + t_{23} * b + t_{24}$$

$$B = t_{31} * r + t_{32} * g + t_{33} * b + t_{34}$$

where

$$T = \begin{bmatrix} t_{11} & t_{12} & t_{13} & t_{14} \\ t_{21} & t_{22} & t_{23} & t_{24} \\ t_{31} & t_{32} & t_{33} & t_{34} \end{bmatrix}$$

is the color twist matrix. The color twist matrix used by the Intel IPP functions is a matrix of size 3x4, or 4x4 with floating-point elements. The matrix elements are specific for each particular type of color conversion.

## ColorTwist

*Applies a color twist matrix to an image with floating-point pixel values.*

### Syntax

#### Case 1: Not-in-place operation on pixel-order data

```
IppStatus ippiColorTwist_<mod>(const Ipp32f* pSrc, int srcStep, Ipp32f* pDst, int dstStep,
IppiSize roiSize, const Ipp32f twist[3][4]);
```

Supported values for mod:

```
32f_C3R
32f_AC4R
```

```
IppStatus ippiColorTwist_32f_C4R(const Ipp32f* pSrc, int srcStep, Ipp32f* pDst, int dstStep,
IppiSize roiSize, const Ipp32f twist[3][4]);
```

#### Case 2: Not-in-place operation on planar data

```
IppStatus ippiColorTwist_32f_P3R(const Ipp32f* const pSrc[3], int srcStep, Ipp32f* const
pDst[3], int dstStep, IppiSize roiSize, const Ipp32f twist[3][4]);
```

#### Case 3: In-place operation on pixel-order data

```
IppStatus ippiColorTwist_<mod>(Ipp32f* pSrcDst, int srcDstStep, IppiSize roiSize, const
Ipp32f twist[3][4]);
```

Supported values for mod:

```
32f_C3IR
32f_AC4IR
```

#### Case 4: In-place operation on planar data

```
IppStatus ippiColorTwist_32f_IP3R(Ipp32f* const pSrcDst[3], int srcDstStep, IppiSize
roiSize, const Ipp32f twist[3][4]);
```

## Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>pSrcDst</i>	Pointer to the source and destination image ROI for the in-place operation.
<i>srcDstStep</i>	Distance in bytes between starts of consecutive lines in the source and destination image for the in-place operation.
<i>roiSize</i>	Size of the source and destination ROI in pixels.
<i>twist</i>	The array containing color-twist matrix elements.

## Description

The function `ippiColorTwist` is declared in the `ippcc.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function applies the color-twist matrix to all three color channels in the source image with floating-point pixel values to obtain the resulting data in the destination image. The destination channel value is obtained as the result of multiplying the corresponding row of the color-twist matrix by the vector of source pixel channel values.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pSrc</i> , <i>pDst</i> , or <i>pSrcDst</i> is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with a zero or negative value.

## ColorTwist32f

*Applies a color twist matrix to an image with integer pixel values.*

---

## Syntax

### Case 1: Not-in-place operation on pixel-order data

```
IppStatus ippiColorTwist32f_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, const Ipp32f twist[3][4]);
```

Supported values for `mod`:

8u_C3R	8s_C3R	16u_C3R	16s_C3R
8u_AC4R	8s_AC4R	16u_AC4R	16s_AC4R

**Case 2: Not-in-place operation on planar data**

```
IppStatus ippiColorTwist32f_<mod>(const Ipp<datatype>* const pSrc[3], int srcStep,
Ipp<datatype>* const pDst[3], int dstStep, IppiSize roiSize, const Ipp32f twist[3][4]);
```

Supported values for mod :

8u\_P3R            8s\_P3R            16u\_P3R            16s\_P3R

**Case 3: In-place operation on pixel-order data**

```
IppStatus ippiColorTwist32f_<mod>(Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize,
const Ipp32ftwist[3][4]);
```

Supported values for mod :

8u\_C3IR            8s\_C3IR            16u\_C3IR            16s\_C3IR  
8u\_AC4IR            8s\_AC4IR            16u\_AC4IR            16s\_AC4IR

**Case 4: In-place operation on planar data**

```
IppStatus ippiColorTwist32f_<mod>(Ipp<datatype>* const pSrcDst[3], int srcDstStep, IppiSize
roiSize, const Ipp32f twist[3][4]);
```

Supported values for mod :

8u\_IP3R            8s\_IP3R            16u\_IP3R            16s\_IP3R

**Parameters**

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>pSrcDst</i>	Pointer to the source and destination image ROI for the in-place operation.
<i>srcDstStep</i>	Distance in bytes between starts of consecutive lines in the source and destination image for the in-place operation.
<i>roiSize</i>	Size of the source and destination ROI in pixels.
<i>twist</i>	The array containing color-twist matrix elements.

**Description**

The function `ippiColorTwist32f` is declared in the `ippcc.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function applies the color-twist matrix to all three color channel values in the integer source image to obtain the resulting data in the destination image. For example, the conversion from the RGB to the YCbCr format can be done as

$$Y = 0.299 * R + 0.587 * G + 0.114 * B$$

$$Cb = -0.16874 * R - 0.33126 * G + 0.5 * B + 0.5$$

$$Cr = 0.5 * R - 0.41869 * G - 0.08131 * B + 0.5$$

which can be described in terms of the following color twist matrix:

```
0.29900f  0.58700f  0.11400f  0.000f
-0.16874f -0.33126f  0.50000f 128.0f
0.50000f  -0.41869f -0.08131f 128.0f
```

Color-twist matrices may also be used to perform many other color conversions.

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pSrc</i> , <i>pDst</i> , or <i>pSrcDst</i> is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with a zero or negative value.

## Color Keying

### CompColorKey

*Performs color keying of two images.*

---

#### Syntax

##### Case 1: Operation on one-channel data

```
IppStatus ippCompColorKey_<mod>(const Ipp<datatype>* pSrc1, int src1Step, const
Ipp<datatype>* pSrc2, int src2Step, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize,
Ipp<datatype> colorKey);
```

Supported values for mod:

```
8u_C1R      16u_C1R      16s_C1R
```

##### Case 2: Operation on multi-channel data

```
IppStatus ippCompColorKey_<mod>(const Ipp<datatype>* pSrc1, int src1Step, const
Ipp<datatype>* pSrc2, int src2Step, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize,
Ipp<datatype> colorKey[3]);
```

Supported values for mod:

```
8u_C3R      16u_C3R      16s_C3R
```

```
IppStatus ippCompColorKey_<mod>(const Ipp<datatype>* pSrc1, int src1Step, const
Ipp<datatype>* pSrc2, int src2Step, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize,
Ipp<datatype> colorKey[4]);
```

Supported values for mod:

```
8u_C4R      16u_C4R      16s_C4R
```

## Parameters

<i>pSrc1, pSrc2</i>	Pointer to the source images ROI.
<i>src1Step, src2Step</i>	Distances in bytes between starts of consecutive lines in the source images.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>roiSize</i>	Size of the source and destination ROI in pixels.
<i>colorKey</i>	Value of the key color for 1-channel images, array of color values for multi-channel images.

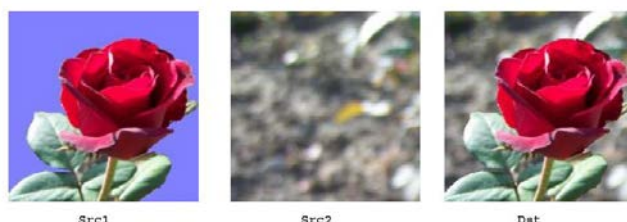
## Description

The function `ippiCompColorKey` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function replaces all areas of the source image *pSrc1* containing the specified key color *colorKey* with the corresponding pixels of the background image *pSrc2* and stores the result in the destination image *pDst*.

The [Figure Applying the Function `ippiCompColorKey` to Sample Images](#) shows an example of how the function `ippiCompColorKey` works.

## Applying the Function `ippiCompColorKey` to Sample Images



## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with a zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if one of the step values is less than or equal to 0.

## AlphaCompColorKey

*Performs color keying and alpha composition of two images.*

---

### Syntax

```
IppStatus ippAlphaCompColorKey_8u_AC4R(const Ipp8u* pSrc1, int src1Step, Ipp8u alpha1,
const Ipp8u* pSrc2, int src2Step, Ipp8u alpha2, Ipp8u* pDst, int dstStep, IppiSize roiSize,
Ipp8u colorKey[4], IppiAlphaType alphaType);
```

### Parameters

<i>pSrc1, pSrc2</i>	Pointer to the source images ROI.
<i>src1Step, src2Step</i>	Distances in bytes between starts of consecutive lines in the source images.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>roiSize</i>	Size of the source and destination ROI in pixels.
<i>colorKey</i>	Array of color values.
<i>alphaType</i>	The type of composition to perform (without pre-multiplying). See <a href="#">Table “Possible Values of alphaType Parameter”</a> for more details.

### Description

The function `ippAlphaCompColorKey` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function replaces all areas of the source image *pSrc1* containing the specified key color *colorKey* with the corresponding pixels of the background image *pSrc2* and additionally performs alpha composition (see supported [Table “Types of Image Compositing Operations”](#)) in accordance with the parameter *alphaType*. Note the alpha channel in the *pDst* is not changed after color keying.

The parameter *alphaType* should not be set to the values intended for operations with pre-multiplying.

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with a zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if one of the step values is less than or equal to 0.
<code>ippStsAlphaTypeErr</code>	Indicates an error condition if <i>alphaType</i> specifies the unsupported type of composition.



## Gamma Correction

### GammaFwd

*Performs gamma-correction of the source image with RGB data.*

---

#### Syntax

##### Case 1: Not-in-place operation on integer pixel-order data

```
IppStatus ippiGammaFwd_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pDst,
int dstStep, IppiSize roiSize);
```

Supported values for mod:

8u_C3R	16u_C3R
8u_AC4R	16u_AC4R

##### Case 2: Not-in-place operation on integer planar data

```
IppStatus ippiGammaFwd_<mod>(const Ipp<datatype>* const pSrc[3], int srcStep, Ipp<datatype>*
const pDst[3], int dstStep, IppiSize roiSize);
```

Supported values for mod:

8u_P3R	16u_P3R
--------	---------

##### Case 3: Not-in-place operation on floating-point pixel-order data

```
IppStatus ippiGammaFwd_<mod>(const Ipp32f* pSrc, int srcStep, Ipp32f* pDst, int dstStep,
IppiSize roiSize, Ipp32f vMin, Ipp32f vMax);
```

Supported values for mod:

32f_C3R
32f_AC4R

##### Case 4: Not-in-place operation on floating-point planar data

```
IppStatus ippiGammaFwd_32f_P3R (const Ipp32f* const pSrc[3], int srcStep, Ipp32f* const
pDst[3], int dstStep, IppiSize roiSize, Ipp32f vMin, Ipp32f vMax);
```

##### Case 5: In-place operation on integer pixel-order data

```
IppStatus ippiGammaFwd_<mod>(Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize);
```

Supported values for mod:

8u_C3IR	16u_C3IR
8u_AC4IR	16u_AC4IR

**Case 6: In-place operation on integer planar data**

```
IppStatus ippiGammaFwd_<mod>(Ipp<datatype>* const pSrcDst[3], int srcDstStep, IppiSize roiSize);
```

Supported values for mod:

8u\_IP3R      16u\_IP3R

**Case 7: In-place operation on floating-point pixel-order data**

```
IppStatus ippiGammaFwd_<mod>(Ipp32f* pSrcDst, int srcDstStep, IppiSize roiSize, Ipp32f vMin, Ipp32f vMax);
```

Supported values for mod:

32f\_C3IR  
32f\_AC4IR

**Case 8: In-place operation on floating-point planar data**

```
IppStatus ippiGammaFwd_32f_IP3R (Ipp32f* const pSrcDst[3], int srcDstStep, IppiSize roiSize, Ipp32f vMin, Ipp32f vMax);
```

**Parameters**

<i>pSrc</i>	Pointer to the ROI in the pixel-order source image. Array of pointers to the ROI in each plane of the planar source image.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the ROI in the pixel-order destination image. Array of pointers to the ROI in each plane of the planar destination image.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>pSrcDst</i>	Pointer to the source and destination image ROI for the in-place operation.
<i>srcDstStep</i>	Distance in bytes between starts of consecutive lines in the source and destination image for the in-place operation.
<i>roiSize</i>	Size of the source and destination ROI in pixels.
<i>vMin, vMax</i>	Minimum and maximum values of the input floating-point data.

**Description**

The function `ippiGammaFwd` is declared in the `ippcc.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function performs [gamma-correction](#) of the source image with RGB data. It uses the following basic equations to convert an RGB image to the gamma-corrected R'G'B' image:

for  $R, G, B < 0.018$

$$R' = 4.5 * R$$

$$G' = 4.5 * G$$

$$B' = 4.5 * B$$

for  $R, G, B \geq 0.018$

$$R' = 1.099 * R^{0.45} - 0.099$$

$$G' = 1.099 * G^{0.45} - 0.099$$

$$B' = 1.099 * B^{0.45} - 0.099$$

Note that the channel intensity values are normalized to fit in the range of [0..1]. The gamma value is equal to  $1/0.45 = 2.22$  in conformity with [\[ITU709\]](#) specification.

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <code>pSrc</code> , <code>pDst</code> , or <code>pSrcDst</code> is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <code>roiSize</code> has a field with a zero or negative value.
<code>ippStsGammaRangeErr</code>	Indicates an error condition if the input data bounds are incorrect, that is <code>vMax</code> is less than or equal to <code>vMin</code> .

## Gammaln

*Converts a gamma-corrected R'G'B' image back to the original RGB image.*

### Syntax

#### Case 1: Not-in-place operation on integer pixel-order data

```
IppStatus ippGammaInv_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pDst,
int dstStep, IppiSize roiSize);
```

Supported values for `mod`:

8u_C3R	16u_C3R
8u_AC4R	16u_AC4R

#### Case 2: Not-in-place operation on integer planar data

```
IppStatus ippGammaInv_<mod>(const Ipp<datatype>* const pSrc[3], int srcStep, Ipp<datatype>*
const pDst[3], int dstStep, IppiSize roiSize);
```

Supported values for `mod`:

8u_P3R	16u_P3R
--------	---------

**Case 3: Not-in-place operation on floating-point pixel-order data**

```
IppStatus ippiGammaInv_<mod>(const Ipp32f* pSrc, int srcStep, Ipp32f* pDst, int dstStep,
IppiSize roiSize, Ipp32f vMin, Ipp32f vMax);
```

Supported values for mod:

```
32f_C3R
32f_AC4R
```

**Case 4: Not-in-place operation on floating-point planar data**

```
IppStatus ippiGammaInv_32f_P3R (const Ipp32f* const pSrc[3], int srcStep, Ipp32f* const
pDst[3], int dstStep, IppiSize roiSize, Ipp32f vMin, Ipp32f vMax);
```

**Case 5: In-place operation on integer pixel-order data**

```
IppStatus ippiGammaInv_<mod>(Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize);
```

Supported values for mod:

```
8u_C3IR    16u_C3IR
8u_AC4IR    16u_AC4IR
```

**Case 6: In-place operation on integer planar data**

```
IppStatus ippiGammaInv_<mod>(Ipp<datatype>* const pSrcDst[3], int srcDstStep, IppiSize
roiSize);
```

Supported values for mod:

```
8u_IP3R    16u_IP3R
```

**Case 7: In-place operation on floating-point pixel-order data**

```
IppStatus ippiGammaInv_<mod>(Ipp32f* pSrcDst, int srcDstStep, IppiSize roiSize, Ipp32f
vMin, Ipp32f vMax);
```

Supported values for mod:

```
32f_C3IR
32f_AC4IR
```

**Case 8: In-place operation on floating-point planar data**

```
IppStatus ippiGammaInv_32f_IP3R (Ipp32f* const pSrcDst[3], int srcDstStep, IppiSize roiSize,
Ipp32f vMin, Ipp32f vMax);
```

**Parameters**

<i>pSrc</i>	Pointer to the ROI in the pixel-order source image. Array of pointers to the ROI in each plane of the planar source image.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.

<i>pDst</i>	Pointer to the ROI in the pixel-order destination image. Array of pointers to the ROI in each plane of the planar destination image.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>pSrcDst</i>	Pointer to the source and destination image ROI for the in-place operation.
<i>srcDstStep</i>	Distance in bytes between starts of consecutive lines in the source and destination image for the in-place operation.
<i>roiSize</i>	Size of the source and destination ROI in pixels.
<i>vMin, vMax</i>	Minimum and maximum values of the input floating-point data.

## Description

The function `ippiGammaInv` is declared in the `ippcc.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts a [gamma-corrected](#) R'G'B' image back to the original RGB image. It uses the following equations:

for  $R', G', B' < 0.0812$

$$R = R' / 4.5$$

$$G = G' / 4.5$$

$$B = B' / 4.5$$

for  $R', G', B' \geq 0.0812$

$$R = \left( \frac{R' + 0.099}{1.099} \right)^{2.22}$$

$$G = \left( \frac{G' + 0.099}{1.099} \right)^{2.22}$$

$$B = \left( \frac{B' + 0.099}{1.099} \right)^{2.22}$$

Note that the channel intensity values are normalized to fit in the range of [0..1]. The gamma value is equal to  $1/0.45 = 2.22$  in conformity with [ITU709](#) specification.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pSrc</i> , <i>pDst</i> , or <i>pSrcDst</i> is NULL.

<code>ippStsSizeErr</code>	Indicates an error condition if <code>roiSize</code> has a field with a zero or negative value.
<code>ippStsGammaRangeErr</code>	Indicates an error condition if the input data bounds are incorrect, that is, <code>vMax</code> is less than or equal to <code>vMin</code> .

## Intensity Transformation

The functions described in this section perform different types of intensity transformation including reduction of the intensity levels in each channel of the image, intensity transformation using lookup tables, and mapping high dynamic range image (HDRI) into low dynamic range image (LDRI).

### ReduceBits

*Reduces the bit resolution of an image.*

---

#### Syntax

##### Case 1: Operation on data of the same source and destination bit depths

```
IppStatus ippiReduceBits_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, int noise, IppiDitherType dtype, int levels);
```

Supported values for `mod`:

<code>8u_C1R</code>	<code>16u_C1R</code>	<code>16s_C1R</code>
<code>8u_C3R</code>	<code>16u_C3R</code>	<code>16s_C3R</code>
<code>8u_C4R</code>	<code>16u_C4R</code>	<code>16s_C4R</code>
<code>8u_AC4R</code>	<code>16u_AC4R</code>	<code>16s_AC4R</code>

##### Case 2: Operation on data of different source and destination bit depths

```
IppStatus ippiReduceBits_8ulu_C1R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst, int dstStep, int dstBitOffset, IppiSize roiSize, int noise, int seed, IppiDitherType dtype, Ipp8u threshold);
```

```
IppStatus ippiReduceBits_<mod>(const Ipp<srcDatatype>* pSrc, int srcStep, Ipp<dstDatatype>* pDst, int dstStep, IppiSize roiSize, int noise, IppiDitherType dtype, int levels);
```

Supported values for `mod`:

<code>16u8u_C1R</code>	<code>16s8u_C1R</code>	<code>32f8u_C1R</code>	<code>32f16u_C1R</code>	<code>32f16s_C1R</code>
<code>16u8u_C3R</code>	<code>16s8u_C3R</code>	<code>32f8u_C3R</code>	<code>32f16u_C3R</code>	<code>32f16s_C3R</code>
<code>16u8u_C4R</code>	<code>16s8u_C4R</code>	<code>32f8u_C4R</code>	<code>32f16u_C4R</code>	<code>32f16s_C4R</code>
<code>16u8u_AC4R</code>	<code>16s8u_AC4R</code>	<code>32f8u_AC4R</code>	<code>32f16u_AC4R</code>	<code>32f16s_AC4R</code>

#### Parameters

<code>pSrc</code>	Pointer to the source image ROI.
<code>srcStep</code>	Distance in bytes between starts of consecutive lines in the source image.

<i>pDst</i>	Pointer to the destination image ROI.										
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.										
<i>dstBitOffset</i>	Offset (in bits) in the first byte of the destination image row.										
<i>roiSize</i>	Size of the source and destination ROI in pixels.										
<i>noise</i>	The number specifying the amount of noise added. This parameter is set as a percentage of range [0..100].										
<i>seed</i>	The seed value used by the pseudo-random number generation, should be set to 0.										
<i>dtype</i>	The type of dithering to be used. The following types are supported: <table> <tr> <td><code>ippDitherNone</code></td><td>No dithering is done</td></tr> <tr> <td><code>ippDitherStucki</code></td><td>The Stucki's error diffusion dithering algorithm is used</td></tr> <tr> <td><code>ippDitherFS</code></td><td>The Floyd-Steinberg error diffusion dithering algorithm is used</td></tr> <tr> <td><code>ippDitherJJN</code></td><td>The Jarvice-Judice-Ninke error diffusion dithering algorithm is used</td></tr> <tr> <td><code>ippDitherBayer</code></td><td>The Bayer's threshold dithering algorithm is used</td></tr> </table>	<code>ippDitherNone</code>	No dithering is done	<code>ippDitherStucki</code>	The Stucki's error diffusion dithering algorithm is used	<code>ippDitherFS</code>	The Floyd-Steinberg error diffusion dithering algorithm is used	<code>ippDitherJJN</code>	The Jarvice-Judice-Ninke error diffusion dithering algorithm is used	<code>ippDitherBayer</code>	The Bayer's threshold dithering algorithm is used
<code>ippDitherNone</code>	No dithering is done										
<code>ippDitherStucki</code>	The Stucki's error diffusion dithering algorithm is used										
<code>ippDitherFS</code>	The Floyd-Steinberg error diffusion dithering algorithm is used										
<code>ippDitherJJN</code>	The Jarvice-Judice-Ninke error diffusion dithering algorithm is used										
<code>ippDitherBayer</code>	The Bayer's threshold dithering algorithm is used										
<i>levels</i>	The number of output levels for halftoning (dithering); can be varied in the range $[2..(1 < depth)]$ , where <i>depth</i> is the bit depth of the destination image.										
<i>threshold</i>	Threshold level for Stucki's dithering for the function <code>ippiReduceBits_8u1u_C1R</code> .										

## Description

The function `ippiReduceBits` is declared in the `ippcc.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function reduces the number of intensity levels in each channel of the source image *pSrc* and places the results in respective channels of the destination image *pDst*. Note that for floating point source data type, RGB values must be in the range [0..1].

The *levels* parameter sets the resultant number of intensity levels in each channel of the destination image.

If the *noise* value is greater than 0, some random noise is added to the threshold level used in computations. The amplitude of the noise signal is specified by the *noise* parameter set as a percentage of the destination image luminance range. For the 4x4 ordered dithering mode, the threshold value is determined by the dither matrix used, whereas for the error diffusion dithering mode the input threshold is set as half of the *range* value, where

$$range = ((1 < depth) - 1) / (levels - 1)$$

and *depth* is the bit depth of the source image.

For floating-point data type,  $range = 1.0 / (levels - 1)$ .

**8u to 1u conversion.** Source image is converted to a bitonal image. The function `ippiReduceBits_8u1u_C1R` supports only one dithering algorithm - Stucki's error diffusion. The destination image has a 8u data type, where each byte represents eight consecutive pixels of the bitonal image (1 bit per pixel). In this case, additional parameter `dstBitOffset` is required to specify the start position of the destination ROI buffer.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <code>pSrc</code> or <code>pDst</code> is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <code>roiSize</code> has a field with a zero or negative value.
<code>ippStsNoiseValErr</code>	Indicates an error condition if <code>noise</code> has an illegal value.
<code>ippStsDitherTypeErr</code>	Indicates an error condition if the specified dithering type is not supported.
<code>ippStsDitherLevelsErr</code>	Indicates an error condition if <code>levels</code> value is out of admissible range.
<code>ippStsMemAllocErr</code>	Indicates an error condition if memory allocation fails.

## LUT

Maps an image by applying intensity transformation.

### Syntax

#### Case 1: Not-in-place operation on one-channel integer data

```
IppStatus ippiLUT_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, const Ipp32s* pValues, const Ipp32s* pLevels, int nLevels);
```

Supported values for `mod`:

8u\_C1R            16u\_C1R            16s\_C1R

#### Case 2: Not-in-place operation on multi-channel integer data

```
IppStatus ippiLUT_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, const Ipp32s* pValues[3], const Ipp32s* pLevels[3], int nLevels[3]);
```

Supported values for `mod`:

8u\_C3R            16u\_C3R            16s\_C3R  
8u\_AC4R           16u\_AC4R           16s\_AC4R

```
IppStatus ippiLUT_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, const Ipp32s* pValues[4], const Ipp32s* pLevels[4], int nLevels[4]);
```

Supported values for `mod`:

8u\_C4R            16u\_C4R            16s\_C4R



**Case 3: Not-in-place operation on one-channel floating-point data**

```
IppStatus ippiLUT_32f_C1R(const Ipp32f* pSrc, int srcStep, Ipp32f* pDst, int dstStep,
IppiSize roiSize, const Ipp32f* pValues, const Ipp32f* pLevels, int nLevels);
```

**Case 4: Not-in-place operation on multi-channel floating-point data**

```
IppStatus ippiLUT_<mod>(const Ipp32f* pSrc, int srcStep, Ipp32f* pDst, int dstStep, IppiSize
roiSize, const Ipp32f* pValues[3], const Ipp32f* pLevels[3], int nLevels[3]);
```

Supported values for mod:

```
32f_C3R
32f_AC4R
```

```
IppStatus ippiLUT_32f_C4R(const Ipp32f* pSrc, int srcStep, Ipp32f* pDst, int dstStep,
IppiSize roiSize, const Ipp32f* pValues[4], const Ipp32f* pLevels[4], int nLevels[4]);
```

**Case 5: In-place operation on one-channel integer data**

```
IppStatus ippiLUT_<mod>(Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize, const
Ipp32s* pValues, const Ipp32s* pLevels, int nLevels);
```

Supported values for mod:

```
8u_C1IR      16u_C1IR      16s_C1IR
```

**Case 6: In-place operation on multi-channel integer data**

```
IppStatus ippiLUT_<mod>(Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize, const
Ipp32s* pValues[3], const Ipp32s* pLevels[3], int nLevels[3]);
```

Supported values for mod:

```
8u_C3IR      16u_C3IR      16s_C3IR
8u_AC4IR      16u_AC4IR      16s_AC4IR
```

```
IppStatus ippiLUT_<mod>(Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize, const
Ipp32s* pValues[4], const Ipp32s* pLevels[4], int nLevels[4]);
```

Supported values for mod:

```
8u_C4IR      16u_C4IR      16s_C4IR
```

**Case 7: In-place operation on one-channel floating-point data**

```
IppStatus ippiLUT_32f_C1R(Ipp32f* pSrcDst, int srcDstStep, IppiSize roiSize, const Ipp32f*
pValues, const Ipp32f* pLevels, int nLevels);
```

**Case 8: In-place operation on multi-channel floating-point data**

```
IppStatus ippiLUT_<mod>(Ipp32f* pSrcDst, int srcDstStep, IppiSize roiSize, const Ipp32f*
pValues[3], const Ipp32f* pLevels[3], int nLevels[3]);
```

Supported values for mod:

```
32f_C3IR
32f_AC4IR
```

```
IppStatus ippiLUT_32f_C4IR(Ipp32f* pSrcDst, int srcDstStep, IppiSize roiSize, const Ipp32f*
pValues[4], const Ipp32f* pLevels[4], int nLevels[4]);
```

## Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>pSrcDst</i>	Pointer to the source and destination image ROI for the in-place operation.
<i>srcDstStep</i>	Distance in bytes between starts of consecutive lines in the source and destination image for the in-place operation.
<i>roiSize</i>	Size of the source ROI in pixels.
<i>pValues</i>	Pointer to the array of intensity values. In case of multi-channel data, <i>pValues</i> is an array of pointers to the intensity values array for each channel.
<i>pLevels</i>	Pointer to the array of level values. In case of multi-channel data, <i>pLevels</i> is an array of pointers to the level values array for each channel.
<i>nLevels</i>	Number of levels, separate for each channel.

## Description

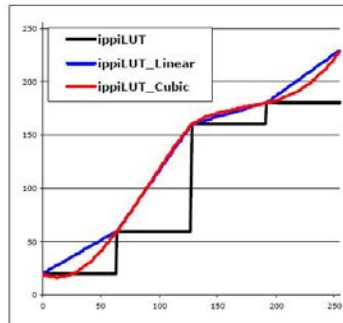
The function `ippiLUT` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function performs intensity transformation of the source image *pSrc* using the lookup table (LUT) specified by the arrays *pLevels* and *pValues*. Every source pixel *pSrc(x,y)* from the range [*pLevels*[*k*], *pLevels*[*k*+1]) is mapped to the destination pixel *pDst(x,y)* whose value is equal to the intensity *pValues*[*k*].

Length of the *pLevels* and *pValues* arrays is defined by the *nLevels* parameter. Number of level and intensity values is less than *nLevels* by one. Pixels in the *pSrc* image that are not in the range [*pLevels*[0], *pLevels*[*nLevels*-1]) are copied to the *pDst* image without any transformation.

Figure Example Mapping Curves Used by the `ippiLUT` Function Flavors shows particular curves that are used in all the `ippiLUT` function flavors for mapping. The level values are 0, 64, 128, 192, 256; the intensity values are 20, 60, 160, 180, 230.

### Example Mapping Curves Used by the `ippiLUT` Function Flavors



### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error when any of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <code>roiSize</code> has a field with a zero or negative value.
<code>ippStsMemAllocErr</code>	Indicates an error when there is not enough memory for inner buffers.
<code>ippStsLUTNoLevelsErr</code>	Indicates an error when the <code>nLevels</code> is less than 2.

### LUT\_Linear

*Maps an image by applying intensity transformation with linear interpolation.*

### Syntax

#### Case 1: Not-in-place operation on one-channel integer data

```
ippStatus ippiLUT_Linear_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>*
pDst, int dstStep, IppiSize roiSize, const Ipp32s* pValues, const Ipp32s* pLevels, int
nLevels);
```

Supported values for `mod`:

`8u_C1R`      `16u_C1R`      `16s_C1R`

## Case 2: Not-in-place operation on multi-channel integer data.

```
ippStatus ippiLUT_Linear_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>*
pDst, int dstStep, IppiSize roiSize, const Ipp32s* pValues[3], const Ipp32s* pLevels[3],
int nLevels[3]);
```

Supported values for mod:

8u_C3R	16u_C3R	16s_C3R
8u_AC4R	16u_AC4R	16s_AC4R

```
ippStatus ippiLUT_Linear_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>*
pDst, int dstStep, IppiSize roiSize, const Ipp32s* pValues[4], const Ipp32s* pLevels[4],
int nLevels[4]);
```

Supported values for mod:

8u_C4R	16u_C4R	16s_C4R
--------	---------	---------

## Case 3: Not-in-place operation on one-channel floating-point data

```
ippStatus ippiLUT_Linear_32f_C1R(const Ipp32f* pSrc, int srcStep, Ipp32f* pDst, int dstStep,
IppiSize roiSize, const Ipp32f* pValues, const Ipp32f* pLevels, int nLevels);
```

## Case 4: Not-in-place operation on multi-channel floating-point data

```
ippStatus ippiLUT_Linear_<mod>(const Ipp32f* pSrc, int srcStep, Ipp32f* pDst, int dstStep,
IppiSize roiSize, const Ipp32f* pValues[3], const Ipp32f* pLevels[3], int nLevels[3]);
```

Supported values for mod:

32f_C3R
32f_AC4R

```
ippStatus ippiLUT_Linear_32f_C4R(const Ipp32f* pSrc, int srcStep, Ipp32f* pDst, int dstStep,
IppiSize roiSize, const Ipp32f* pValues[4], const Ipp32f* pLevels[4], int nLevels[4]);
```

## Case 5: In-place operation on one-channel integer data

```
ippStatus ippiLUT_Linear_<mod>(Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize,
const Ipp32s* pValues, const Ipp32s* pLevels, int nLevels);
```

Supported values for mod:

8u_C1IR	16u_C1IR	16s_C1IR
---------	----------	----------

## Case 6: In-place operation on multi-channel integer data.

```
ippStatus ippiLUT_Linear_<mod>(Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize,
const Ipp32s* pValues[3], const Ipp32s* pLevels[3], int nLevels[3]);
```

Supported values for mod:

8u_C3IR	16u_C1IR	16s_C1IR
8u_AC4IR	16u_C1IR	16s_C1IR

```
IppStatus ippiLUT_Linear_<mod>(Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize,
const Ipp32s* pValues[4], const Ipp32s* pLevels[4], int nLevels[4]);
```

Supported values for mod:

```
8u_C4IR      16u_C4IR      16s_C4IR
```

### Case 7: In-place operation on one-channel floating-point data

```
IppStatus ippiLUT_Linear_32f_C1R(Ipp32f* pSrcDst, int srcDstStep, IppiSize roiSize, const
Ipp32f* pValues, const Ipp32f* pLevels, int nLevels);
```

### Case 8: In-place operation on multi-channel floating-point data

```
IppStatus ippiLUT_Linear_<mod>(Ipp32f* pSrcDst, int srcDstStep, IppiSize roiSize, const
Ipp32f* pValues[3], const Ipp32f* pLevels[3], int nLevels[3]);
```

Supported values for mod:

```
32f_C3IR
32f_AC4IR
```

```
IppStatus ippiLUT_Linear_32f_C4R(Ipp32f* pSrcDst, int srcDstStep, IppiSize roiSize, const
Ipp32f* pValues[4], const Ipp32f* pLevels[4], int nLevels[4]);
```

## Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>pSrcDst</i>	Pointer to the source and destination image ROI for the in-place operation.
<i>srcDstStep</i>	Distance in bytes between starts of consecutive lines in the source and destination image for the in-place operation.
<i>roiSize</i>	Size of the source ROI in pixels.
<i>pValues</i>	Pointer to the array of intensity values. In case of multi-channel data, <i>pValues</i> is an array of pointers to the intensity values array for each channel.
<i>pLevels</i>	Pointer to the array of level values. In case of multi-channel data, <i>pLevels</i> is an array of pointers to the level values array for each channel.
<i>nLevels</i>	Number of levels, separate for each channel.

## Description

The function `ippiLUT_Linear` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function performs intensity transformation of the source image *pSrc* using the lookup table (LUT) with linear interpolation between two neighbor levels. LUT is specified by the arrays *pLevels* and *pValues*. Every source pixel *pSrc(x,y)* from the range [*pLevels*[*k*], *pLevels*[*k*+1]) is mapped to the destination pixel *pDst(x,y)* whose value is computed according to the following relation:

$$pDst(x, y) = pValues[k] + (pSrc(x, y) - pLevels[k]) \cdot \frac{pValues[k+1] - pValues[k]}{pLevels[k+1] - pLevels[k]}$$

Length of the *pLevels* and *pValues* arrays is defined by the *nLevels* parameter. Pixels in *pSrc* image that are not in the range [*pLevels*[0], *pLevels*[*nLevels*-1]) are copied to *pDst* image without any transformation.

[Figure Example Mapping Curves Used by the ippiLUT Function Flavors](#) shows particular curves that are used in all ippiLUT function flavors for mapping. The level values are 0, 64, 128, 192, 256; the intensity values are 20, 60, 160, 180, 230.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error when any of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with a zero or negative value.
<code>ippStsMemAllocErr</code>	Indicates an error when there is not enough memory for inner buffers.
<code>ippStsLUTNoLevelsErr</code>	Indicates an error when the <i>nLevels</i> is less than 2.

## Example

The code example below shows how to use the function `ippiLUT_Linear_32f_C1R`.

```
void func_LUTLinear()
{
    Ipp32f pSrc[8*8];
    int srcStep = 8*sizeof(Ipp32f);
    IppiSize roiSize = {8, 8};

    Ipp32f pDst[8*8];
    int dstStep = 8*sizeof(Ipp32f);
    Ipp32f pLevels[5] = {0.0, 0.128, 0.256, 0.512, 1.0};
    Ipp32f pValues[5] = {0.2, 0.4, 0.6, 0.8, 1.0};
    int nLevels = 5;

    ippiImageJaehne_32f_C1R(pSrc, srcStep, roiSize);

    ippiLUT_Linear_32f_C1R(pSrc, srcStep, pDst, dstStep, roiSize,
        pValues, pLevels, nLevels);
}
```

**Result:**

```
pSrc
0.00 0.26 0.65 0.82 0.82 0.65 0.26 0.00
0.26 0.82 1.00 0.98 0.98 1.00 0.82 0.26
0.65 1.00 0.89 0.74 0.74 0.89 1.00 0.65
0.82 0.98 0.74 0.55 0.55 0.74 0.98 0.82
0.82 0.98 0.74 0.55 0.55 0.74 0.98 0.82
```

```
0.65 1.00 0.89 0.74 0.74 0.89 1.00 0.65
0.26 0.82 1.00 0.98 0.98 1.00 0.82 0.26
0.00 0.26 0.65 0.82 0.82 0.65 0.26 0.00
```

```
pDst
0.20 0.61 0.85 0.93 0.93 0.85 0.61 0.20
0.61 0.93 1.00 0.99 0.99 1.00 0.93 0.61
0.85 1.00 0.95 0.89 0.89 0.95 1.00 0.85
0.93 0.99 0.89 0.82 0.82 0.89 0.99 0.93
0.93 0.99 0.89 0.82 0.82 0.89 0.99 0.93
0.85 1.00 0.95 0.89 0.89 0.95 1.00 0.85
0.61 0.93 1.00 0.99 0.99 1.00 0.93 0.61
0.20 0.61 0.85 0.93 0.93 0.85 0.61 0.20
```

## LUT\_Cubic

*Maps an image by applying intensity transformation with cubic interpolation.*

### Syntax

#### Case 1: Not-in-place operation on one-channel integer data

```
IppStatus ippiLUT_Cubic_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pDst,
int dstStep, IppiSize roiSize, const Ipp32s* pValues, const Ipp32s* pLevels, int nLevels);
```

Supported values for mod:

```
8u_C1R      16u_C1R      16s_C1R
```

#### Case 2: Not-in-place operation on multi-channel integer data

```
IppStatus ippiLUT_Cubic_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pDst,
int dstStep, IppiSize roiSize, const Ipp32s* pValues[3], const Ipp32s* pLevels[3], int
nLevels[3]);
```

Supported values for mod:

```
8u_C3R      16u_C3R      16s_C3R
8u_AC4R      16u_AC4R      16s_AC4R
```

```
IppStatus ippiLUT_Cubic_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pDst,
int dstStep, IppiSize roiSize, const Ipp32s* pValues[4], const Ipp32s* pLevels[4], int
nLevels[4]);
```

Supported values for mod:

```
8u_C4R      16u_C4R      16s_C4R
```

#### Case 3: Not-in-place operation on one-channel floating-point data

```
IppStatus ippiLUT_Cubic_32f_C1R(const Ipp32f* pSrc, int srcStep, Ipp32f* pDst, int dstStep,
IppiSize roiSize, const Ipp32f* pValues, const Ipp32f* pLevels, int nLevels);
```

**Case 4: Not-in-place operation on multi-channel floating-point data**

```
IppStatus ippiLUT_Cubic_<mod>(const Ipp32f* pSrc, int srcStep, Ipp32f* pDst, int dstStep,
IppiSize roiSize, const Ipp32f* pValues[3], const Ipp32f* pLevels[3], int nLevels[3]);
```

Supported values for mod:

32f\_C3R  
32f\_AC4R

```
IppStatus ippiLUT_Cubic_32f_C4R(const Ipp32f* pSrc, int srcStep, Ipp32f* pDst, int dstStep,
IppiSize roiSize, const Ipp32f* pValues[4], const Ipp32f* pLevels[4], int nLevels[4]);
```

**Case 5: In-place operation on one-channel integer data**

```
IppStatus ippiLUT_Cubic_<mod>(Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize,
const Ipp32s* pValues, const Ipp32s* pLevels, int nLevels);
```

Supported values for mod:

8u\_C1IR            16u\_C1IR            16s\_C1IR

**Case 6: In-place operation on multi-channel integer data**

```
IppStatus ippiLUT_Cubic_<mod>(Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize,
const Ipp32s* pValues[3], const Ipp32s* pLevels[3], int nLevels[3]);
```

Supported values for mod:

8u\_C3IR            16u\_C3IR            16s\_C3IR  
8u\_AC4IR           16u\_AC4IR           16s\_AC4IR

```
IppStatus ippiLUT_Cubic_<mod>(Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize,
const Ipp32s* pValues[4], const Ipp32s* pLevels[4], int nLevels[4]);
```

Supported values for mod:

8u\_C4IR            16u\_C4IR            16s\_C4IR

**Case 7: In-place operation on one-channel floating-point data**

```
IppStatus ippiLUT_Cubic_32f_C1R(Ipp32f* pSrcDst, int srcDstStep, IppiSize roiSize, const
Ipp32f* pValues, const Ipp32f* pLevels, int nLevels);
```

**Case 8: In-place operation on multi-channel floating-point data**

```
IppStatus ippiLUT_Cubic_<mod>(Ipp32f* pSrcDst, int srcDstStep, IppiSize roiSize, const
Ipp32f* pValues[3], const Ipp32f* pLevels[3], int nLevels[3]);
```

Supported values for mod:

32f\_C3IR  
32f\_AC4IR

```
IppStatus ippiLUT_Cubic_32f_C4IR(Ipp32f* pSrcDst, int srcDstStep, IppiSize roiSize, const
Ipp32f* pValues[4], const Ipp32f* pLevels[4], int nLevels[4]);
```



## Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>pSrcDst</i>	Pointer to the source and destination image ROI for the in-place operation.
<i>srcDstStep</i>	Distance in bytes between starts of consecutive lines in the source and destination image for the in-place operation.
<i>roiSize</i>	Size of the source ROI in pixels.
<i>pValues</i>	Pointer to the array of intensity values. In case of multi-channel data, <i>pValues</i> is an array of pointers to the intensity values array for each channel.
<i>pLevels</i>	Pointer to the array of level values. In case of multi-channel data, <i>pLevels</i> is an array of pointers to the level values array for each channel.
<i>nLevels</i>	Number of levels, separate for each channel.

## Description

The function `ippiLUT_Cubic` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function performs intensity transformation of the source image *pSrc* using the lookup table (LUT) with cubic interpolation between neighbor levels. The LUT is specified by the arrays *pLevels* and *pValues*.

Every source pixel *pSrc(x,y)* from the range [*pLevels*[*k*], *pLevels*[*k*+1]) is mapped to the destination pixel *pDst(x,y)* whose value is computed as

$$pDst(x,y) = A * pSrc(x,y)^3 + B * pSrc(x,y)^2 + C * pSrc(x,y) + D.$$

(*pLevels*[*k*-1], *pLevels*[*k*-1])

(*pLevels*[*k*], *pLevels*[*k*])

(*pLevels*[*k*+1], *pLevels*[*k*+1])

(*pLevels*[*k*+2], *pLevels*[*k*+2])

Correspondingly, coefficients A, B, C, D are computed by solving the following set of linear equations:

$$A * pLevels[k-1]^3 + B * pLevels[k-1]^2 + C * pLevels[k-1] + D = pValues[k-1]$$

$$A * pLevels[k]^3 + B * pLevels[k]^2 + C * pLevels[k] + D = pValues[k]$$

$$A * pLevels[k+1]^3 + B * pLevels[k+1]^2 + C * pLevels[k+1] + D = pValues[k+1]$$

$$A * pLevels[k+2]^3 + B * pLevels[k+2]^2 + C * pLevels[k+2] + D = pValues[k+2]$$

Length of the *pLevels* and *pValues* arrays is defined by the *nLevels* parameter. Pixels in the *pSrc* image that are not in the range [*pLevels*[0], *pLevels*[*nLevels*-1]) are copied to the *pDst* image without any transformation.

[Figure Example Mapping Curves Used by the ippiLUT Function Flavors](#) shows particular curves that are used in all the ippiLUT function flavors for mapping. The level values are 0, 64, 128, 192, 256; the intensity values are 20, 60, 160, 180, 230.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error when any of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with a zero or negative value.
<code>ippStsMemAllocErr</code>	Indicates an error when there is not enough memory for inner buffers.
<code>ippStsLUTNoLevelsErr</code>	Indicates an error when the <i>nLevels</i> is less than 2.

## LUTPalette, LUTPaletteSwap

*Maps an image by applying intensity transformation in accordance with a palette table.*

---

### Syntax

#### Case 1: Operations on one-channel data

```
IppStatus ippiLUTPalette_<mod>(const Ipp<srcDatatype>* pSrc, int srcStep, Ipp<dstDatatype>* pDst, int dstStep, IppiSize roiSize, const Ipp<dstDatatype>* pTable, int nBitSize);
```

Supported values for mod:

8u_C1R	16u_C1R
8u32u_C1R	16u8u_C1R
	16u32u_C1R

```
IppStatus ippiLUTPalette_<mod>(const Ipp<srcDatatype>* pSrc, int srcStep, Ipp8u* pDst, int dstStep, IppiSize roiSize, const Ipp8u* pTable, int nBitSize);
```

Supported values for mod:

8u24u_C1R	16u24u_C1R
-----------	------------

#### Case 2: Operations on multi-channel data

```
IppStatus ippiLUTPalette_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, const Ipp<datatype>* const pTable[3], int nBitSize);
```

Supported values for mod:

8u_C3R	16u_C3R
8u_AC4R	16u_AC4R

```
IppStatus ippiLUTPalette_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>*
pDst, int dstStep, IppiSize roiSize, const Ipp<datatype>* const pTable[4], int nBitSize);
```

Supported values for mod:

8u\_C4R            16u\_C4R

```
IppStatus ippiLUTPaletteSwap_<mod>(const Ipp<datatype>* pSrc, int srcStep, int alphaValue,
Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, const Ipp<datatype>* const pTable[3],
int nBitSize);
```

Supported values for mod:

8u\_C3A0C4R       16u\_C3A0C4R

## Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>pSrcDst</i>	Pointer to the source and destination image ROI for the in-place operation.
<i>roiSize</i>	Size of the source ROI in pixels.
<i>pTable</i>	Pointer to the palette table, or an array of pointers to the palette tables for each source channel.
<i>alphaValue</i>	Constant value for the alpha channel.
<i>nBitSize</i>	Number of significant bits in the source image.

## Description

The functions `ippiLUTPalette` and `ippiLUTPaletteSwap` are declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

The function `ippiLUTPalette` performs intensity transformation of the source image *pSrc* using the palette lookup table *pTable*. This table is a vector with  $2^{nBitSize}$  elements that contain intensity values specified by the user. The function uses *nBitSize* lower bits of intensity value of each source pixel as an index in the *pTable* and assigns the correspondent intensity value from the table to the respective pixel in the destination image *pDst*. The number of significant bits *nBitSize* should be in the range [1, 8] for functions that operate on 8u source images, and [1, 16] for functions that operate on 16u source images.

Some function flavors that operate on the 3-channel source image additionally create a 4-th channel - alpha channel - in the destination image and place it at first position. The channel values of the alpha channel can be set to the arbitrary constant value *alphaValue*. If this value is less than 0 or greater than the upper boundary of the data range, the channel values are not set.

The function flavor `ippiLUTPaletteSwap` reverses the order of channels in the destination image.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error when any of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <code>roiSize</code> has a field with a zero or negative value.
<code>ippStsOutOfRangeErr</code>	Indicates an error if <code>nBitSize</code> is out of the range.

## Example

The code example below shows how to use the function `ippiLUTPalette_8u32u_C1R`.

```
void func_LUTPalette()
{
    Ipp32f pSrc[8*8];
    int srcStep = 8*sizeof(Ipp32f);
    IppiSize roiSize = {8, 8};

    Ipp32f pDst[8*8];
    int dstStep = 8*sizeof(Ipp32f);
    int nBitSize = 3;
    Ipp32u pTable[8] = {1, 2, 3, 4, 5, 6, 7, 8} ;

    ippiImageJaehne_8u_C1R( pSrc, srcStep, roiSize);

    ippiLUTPalette_8u32u_C1R( pSrc, srcStep, pDst, dstStep, roiSize,
pTable, nBitSize);
}
```

Result:

pSrc

0	67	165	209	209	165	67	0
67	209	255	250	250	255	209	67
165	255	226	188	188	226	255	165
209	250	188	140	140	188	250	209
209	250	188	140	140	188	250	209
165	255	226	188	188	226	255	165
67	209	255	250	250	255	209	67
0	67	165	209	209	165	67	0

pDst

1	4	6	2	2	6	4	1
4	2	8	3	3	8	2	4
6	8	3	5	5	3	8	6
2	3	5	5	5	5	3	2
2	3	5	5	5	5	3	2
6	8	3	5	5	3	8	6
4	2	8	3	3	8	2	4
1	4	6	2	2	6	4	1

## ToneMapLinear, ToneMapMean

*Maps an HDRI image to the LDRI image.*

---

### Syntax

```
IppStatus ippiToneMapLinear_32f8u_C1R(const Ipp32f* pSrc, int srcStep, Ipp8u* pDst, int dstStep, IppiSize roiSize);
```

```
IppStatus ippiToneMapMean_32f8u_C1R(const Ipp32f* pSrc, int srcStep, Ipp8u* pDst, int dstStep, IppiSize roiSize);
```

### Parameters

<i>pSrc</i>	Pointer to the ROI in the HDRI source image.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the ROI in the LDRI destination image.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>roiSize</i>	Size of the source and destination ROI in pixels.

### Description

The functions `ippiToneMapLinear` and `ippiToneMeanMap` are declared in the `ippcc.h` file. They both operate with ROI (see [Regions of Interest in Intel IPP](#)).

These functions convert the source high dynamic range image (HDRI) *pSrc* into low dynamic range image (LDRI) *pDst*. Pixel values of the source image must be positive.

The function `ippiToneMapLinear` implements the Linear Scale-Factor method converting each source pixel *pSrc[i]* in accordance with the formula:

$$pSrc[i] = pSrc[i] / Lmax, Lmax = \max\{pSrc[i]\}.$$

The function `ippiToneMapMean` implements the Mean Value method converting each source pixel *pSrc[i]* in accordance with the formula:

$$pSrc[i] = 0.5 * pSrc[i] / Lave, Lave = \text{average}\{pSrc[i]\}.$$

If the value of *Lmax* or *Lave* is less than 0, then the function does not perform the operation and returns the warning message.

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pSrc</i> or <i>pDst</i> is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with a zero or negative value.
<code>ippStsNoOperation</code>	Indicates a warning if the values of <i>Lmax</i> or <i>Lave</i> are less than 0.



# Threshold and Compare Operations

This chapter describes the Intel® IPP image processing functions that operate on a pixel-by-pixel basis: threshold and compare functions. The table below lists all functions of this group:

## Image Threshold and Compare Operations

Function Base Name	Operation
<b>Thresholding</b>	
<a href="#">Threshold</a>	Performs thresholding of pixel values in an image, using the specified compare operation
<a href="#">Threshold_GT</a>	Thresholds pixel values of an image, using the comparison for "greater than"
<a href="#">Threshold_LT</a>	Thresholds pixel values of an image, using the comparison for "less than"
<a href="#">Threshold_Val</a>	Thresholds pixel values of an image. Pixels that satisfy the compare condition are set to a specified value
<a href="#">Threshold_GTVal</a>	Thresholds pixel values of an image. Pixels that are greater than threshold are set to a specified value
<a href="#">Threshold_LTVal</a>	Thresholds pixel values of an image. Pixels that are less than threshold are set to a specified value
<a href="#">Threshold_LTValGTVal</a>	Performs double thresholding of pixel values in an image
<a href="#">Threshold_ComputeThreshold_Otsu</a>	Computes the value of the Otsu threshold
<b>Comparison</b>	
<a href="#">Compare</a>	Compares pixel values of two images using a specified compare operation
<a href="#">CompareC</a>	Compares pixel values of a source image to a given value using a specified compare operation
<a href="#">CompareEqualEps</a>	Compares two images with floating-point data, testing whether pixel values are equal within a certain tolerance <i>eps</i>
<a href="#">CompareEqualEpsC</a>	Tests whether floating-point pixel values of an image are equal to a given value within a certain tolerance <i>eps</i>

## Thresholding

The threshold functions change pixel values depending on whether they are less or greater than the specified *threshold*.

The type of comparison operation used to threshold pixel values is specified by the *ippCmpOp* parameter; this operation can be either “greater than” or “less than” (see [Structures and Enumerators](#) in Chapter 2 for more information). For some thresholding functions the type of comparison operation is fixed.

If an input pixel value satisfies the compare condition, the corresponding output pixel is set to the fixed value that is specific for a given threshold function flavor. Otherwise, it is either not changed, or set to another fixed value, which is defined in a particular function description.

For images with multi-channel data, the compare conditions should be set separately for each channel.

### Threshold

*Performs thresholding of pixel values in an image buffer.*

#### Syntax

##### Case 1: Not-in-place operation on one-channel data

```
IppStatus ippThreshold_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pDst,
int dstStep, IppiSize roiSize, Ipp<datatype> threshold, IppCmpOp ippCmpOp);
```

Supported values for *mod*:

8u_C1R	16u_C1R	16s_C1R	32f_C1R
--------	---------	---------	---------

##### Case 2: Not-in-place operation on multi-channel data

```
IppStatus ippThreshold_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pDst,
int dstStep, IppiSize roiSize, const Ipp<datatype> threshold[3], IppCmpOp ippCmpOp);
```

Supported values for *mod*:

8u_C3R	16u_C3R	16s_C3R	32f_C3R
8u_AC4R	16u_AC4R	16s_AC4R	32f_AC4R

##### Case 3: In-place operation on one-channel data

```
IppStatus ippThreshold_<mod>(Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize,
Ipp<datatype> threshold, IppCmpOp ippCmpOp);
```

Supported values for *mod*:

8u_C1IR	16u_C1IR	16s_C1IR	32f_C1IR
---------	----------	----------	----------



**Case 4: In-place operation on multi-channel data**

```
IppStatus ippiThreshold_<mod>(Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize,
const Ipp<datatype> threshold[3], IppCmpOp ippCmpOp);
```

Supported values for `mod`:

8u_C3IR	16u_C3IR	16s_C3IR	32f_C3IR
8u_AC4IR	16u_AC4IR	16s_AC4IR	32f_AC4IR

**Parameters**

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>pSrcDst</i>	Pointer to the source and destination image ROI (for the in-place operation).
<i>srcDstStep</i>	Distance in bytes between starts of consecutive lines in the source and destination image buffer (for the in-place operation).
<i>roiSize</i>	Size of the source and destination ROI in pixels.
<i>threshold</i>	The threshold level value to use for each pixel. In case of multi-channel data, an array of threshold values for each color channel is used.
<i>ippCmpOp</i>	The operation specified for comparing pixel values and the <i>threshold</i> . Comparison for either "less than" or "greater than" can be used.

**Description**

The function `ippiThreshold` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function thresholds pixels in the source image *pSrc* using the specified level *threshold*. Pixel values in the source image are compared to the *threshold* value using the *ippCmpOp* comparison operation.

If the result of the compare is true, the corresponding output pixel is set to the *threshold* value. Otherwise, it is set to the source pixel value.

**Return Values**

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointer is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.

<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> , <i>dstStep</i> , or <i>srcDstStep</i> has a zero or negative value
<code>ippStsNotSupportedModeErr</code>	Indicates an error if the comparison mode is not supported.

### Example

The code example below shows how to use the `ippiThreshold_8u_C1R` function.

```
void func_threshold()
{
    IppiSize ROI = {5,4};
    Ipp8u src[9*4] = {1, 2, 4, 8, 16, 8, 4, 2, 1,
                     1, 2, 4, 8, 16, 8, 4, 2, 1,
                     1, 2, 4, 8, 16, 8, 4, 2, 1,
                     1, 2, 4, 8, 16, 8, 4, 2, 1};
    Ipp8u dst[9*4];
    Ipp8u threshold = 6;
    ippiThreshold_8u_C1R(src, 9, dst, 9, ROI, threshold, ippCmpGreater);
}
```

Result:

```
      dst
1  2  4  6  6  8  4  2  1
1  2  4  6  6  8  4  2  1
1  2  4  6  6  8  4  2  1
1  2  4  6  6  8  4  2  1
```

## Threshold\_GT

*Performs thresholding of pixel values in an image, using the comparison for "greater than".*

---

### Syntax

#### Case 1: Not-in-place operation on one-channel data

```
IppStatus ippiThreshold_GT_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>*
pDst, int dstStep, IppiSize roiSize, Ipp<datatype> threshold);
```

Supported values for `mod`:

8u_C1R	16u_C1R	16s_C1R	32f_C1R
--------	---------	---------	---------

#### Case 2: Not-in-place operation on multi-channel data

```
IppStatus ippiThreshold_GT_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>*
pDst, int dstStep, IppiSize roiSize, const Ipp<datatype> threshold[3]);
```

Supported values for `mod`:

8u_C3R	16u_C3R	16s_C3R	32f_C3R
8u_AC4R	16u_AC4R	16s_AC4R	32f_AC4R

**Case 3: In-place operation on one-channel data**

```
IppStatus ippiThreshold_GT_<mod>(Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize,
Ipp<datatype> threshold);
```

Supported values for `mod`:

8u\_C1IR      16u\_C1IR      16s\_C1IR      32f\_C1IR

**Case 4: In-place operation on multi-channel data**

```
IppStatus ippiThreshold_GT_<mod>(Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize,
const Ipp<datatype> threshold[3]);
```

Supported values for `mod`:

8u\_C3IR      16u\_C3IR      16s\_C3IR      32f\_C3IR  
8u\_AC4IR      16u\_AC4IR      16s\_AC4IR      32f\_AC4IR

**Parameters**

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>pSrcDst</i>	Pointer to the source and destination image ROI (for the in-place operation).
<i>srcDstStep</i>	Distance in bytes between starts of consecutive lines in the source and destination image buffer (for the in-place operation).
<i>roiSize</i>	Size of the source and destination ROI in pixels.
<i>threshold</i>	The threshold level value to use for each pixel. In case of multi-channel data, an array of threshold values for each color channel is used.

**Description**

The function `ippiThreshold_GT` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function performs thresholding of pixels in the source image *pSrc* using the specified level *threshold*. Pixel values in the source image are compared to the *threshold* value for “greater than”.

If the result of the compare is true, the corresponding output pixel is set to the *threshold* value. Otherwise, it is set to the source pixel value.

**Return Values**

`ippStsNoErr`      Indicates no error. Any other value indicates an error or a warning.

<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointer is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <code>roiSize</code> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <code>srcStep</code> , <code>dstStep</code> , or <code>srcDstStep</code> has a zero or negative value.

## Threshold\_LT

*Performs thresholding of pixel values in an image buffer, using the comparison for "less than".*

---

### Syntax

#### Case 1: Not-in-place operation on one-channel data

```
IppStatus ippiThreshold_LT<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, Ipp<datatype> threshold);
```

Supported values for `mod`:

8u_C1R	16u_C1R	16s_C1R	32f_C1R
--------	---------	---------	---------

#### Case 2: Not-in-place operation on multi-channel data

```
IppStatus ippiThreshold_LT<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, const Ipp<datatype> threshold[3]);
```

Supported values for `mod`:

8u_C3R	16u_C3R	16s_C3R	32f_C3R
8u_AC4R	16u_AC4R	16s_AC4R	32f_AC4R

#### Case 3: In-place operation on one-channel data

```
IppStatus ippiThreshold_LT<mod>(Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize, Ipp<datatype> threshold);
```

Supported values for `mod`:

8u_C1IR	16u_C1IR	16s_C1IR	32f_C1IR
---------	----------	----------	----------

#### Case 4: In-place operation on multi-channel data

```
IppStatus ippiThreshold_LT<mod>(Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize, const Ipp<datatype> threshold[3]);
```

Supported values for `mod`:

8u_C3IR	16u_C3IR	16s_C3IR	32f_C3IR
8u_AC4IR	16u_AC4IR	16s_AC4IR	32f_AC4IR

## Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>pSrcDst</i>	Pointer to the source and destination image ROI (for the in-place operation).
<i>srcDstStep</i>	Distance in bytes between starts of consecutive lines in the source and destination image buffer (for the in-place operation).
<i>roiSize</i>	Size of the source and destination ROI in pixels.
<i>threshold</i>	The threshold level value to use for each pixel. In case of multi-channel data, an array of threshold values for each color channel is used.

## Description

The function `ippiThreshold_LT` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function performs thresholding of pixels in the source image *pSrc* using the specified level *threshold*. Pixel values in the source image are compared to the *threshold* value for "less than". If the result of the compare is true, the corresponding output pixel is set to the *threshold* value. Otherwise, it is set to the source pixel value.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointer is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> , <i>dstStep</i> , or <i>srcDstStep</i> has a zero or negative value.

## Threshold\_Val

*Performs thresholding of pixel values in an image buffer.  
Pixels that satisfy the compare condition are set to a  
specified value.*

---

### Syntax

#### Case 1: Not-in-place operation on one-channel data

```
IppStatus ippiThreshold_Val_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>*  
pDst, int dstStep, IppiSize roiSize, Ipp<datatype> threshold, Ipp<datatype> value,  
IppCmpOp ippCmpOp);
```

Supported values for mod:

8u_C1R	16u_C1R	16s_C1R	32f_C1R
--------	---------	---------	---------

#### Case 2: Not-in-place operation on multi-channel data

```
IppStatus ippiThreshold_Val_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>*  
pDst, int dstStep, IppiSize roiSize, const Ipp<datatype> threshold[3], const Ipp<datatype>  
value[3], IppCmpOp ippCmpOp);
```

Supported values for mod:

8u_C3R	16u_C3R	16s_C3R	32f_C3R
8u_AC4R	16u_AC4R	16s_AC4R	32f_AC4R

#### Case 3: In-place operation on one-channel data

```
IppStatus ippiThreshold_Val_<mod>(Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize,  
Ipp<datatype> threshold, Ipp<datatype> value, IppCmpOp ippCmpOp);
```

Supported values for mod:

8u_C1IR	16u_C1IR	16s_C1IR	32f_C1IR
---------	----------	----------	----------

#### Case 4: In-place operation on multi-channel data

```
IppStatus ippiThreshold_Val_<mod>(Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize,  
const Ipp<datatype> threshold[3], const Ipp<datatype> value[3], IppCmpOp ippCmpOp);
```

Supported values for mod:

8u_C3IR	16u_C3IR	16s_C3IR	32f_C3IR
8u_AC4IR	16u_AC4IR	16s_AC4IR	32f_AC4IR

## Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>pSrcDst</i>	Pointer to the source and destination image ROI (for the in-place operation).
<i>srcDstStep</i>	Distance in bytes between starts of consecutive lines in the source and destination image buffer (for the in-place operation).
<i>roiSize</i>	Size of the source and destination ROI in pixels.
<i>threshold</i>	The threshold value to use for each pixel. In case of multi-channel data, an array of 3 threshold values (one for each color channel) is used.
<i>value</i>	The output value to be set for each pixel that satisfies the compare condition. In case of multi-channel data, an array of 3 output values (one for each color channel) is used.
<i>ippCmpOp</i>	The operation to use for comparing pixel values and the <i>threshold</i> . Comparison for either "less than" or "greater than" can be used.

## Description

The function `ippiThreshold_Val` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function thresholds pixels in the source image *pSrc* using the specified level *threshold*. Pixel values in the source image are compared to the *threshold* value using the *ippCmpOp* comparison operation. If the result of the compare is true, the corresponding output pixel is set to the specified *value*. Otherwise, it is set to the source pixel value.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointer is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> , <i>dstStep</i> , or <i>srcDstStep</i> has a zero or negative value.

## Threshold\_GTVal

*Performs thresholding of pixel values in an image. Pixels that are greater than threshold, are set to a specified value.*

---

### Syntax

#### Case 1: Not-in-place operation on one-channel data

```
IppStatus ippiThreshold_GTVal_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, Ipp<datatype> threshold, Ipp<datatype> value);
```

Supported values for mod:

8u_C1R	16u_C1R	16s_C1R	32f_C1R
--------	---------	---------	---------

#### Case 2: Not-in-place operation on multi-channel data

```
IppStatus ippiThreshold_GTVal_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, const Ipp<datatype> threshold[3], const Ipp<datatype> value[3]);
```

Supported values for mod:

8u_C3R	16u_C3R	16s_C3R	32f_C3R
8u_AC4R	16u_AC4R	16s_AC4R	32f_AC4R

```
IppStatus ippiThreshold_GTVal_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, const Ipp<datatype> threshold[4], const Ipp<datatype> value[4]);
```

Supported values for mod:

8u_C4R	16u_C4R	16s_C4R	32f_C4R
--------	---------	---------	---------

#### Case 3: In-place operation on one-channel data

```
IppStatus ippiThreshold_GTVal_<mod>(Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize, Ipp<datatype> threshold, Ipp<datatype> value);
```

Supported values for mod:

8u_C1IR	16u_C1IR	16s_C1IR	32f_C1IR
---------	----------	----------	----------



**Case 4: In-place operation on multi-channel data**

```
IppStatus ippiThreshold_GTVVal_<mod>(Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize
roiSize, const Ipp<datatype> threshold[3], const Ipp<datatype> value[3]);
```

Supported values for mod:

8u_C3IR	16u_C3IR	16s_C3IR	32f_C3IR
8u_AC4IR	16u_AC4IR	16s_AC4IR	32f_AC4IR

```
IppStatus ippiThreshold_GTVVal_<mod>(Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize
roiSize, const Ipp<datatype> threshold[4], const Ipp<datatype> value[4]);
```

Supported values for mod:

8u_C4IR	16u_C4IR	16s_C4IR	32f_C4IR
---------	----------	----------	----------

**Parameters**

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>pSrcDst</i>	Pointer to the source and destination image ROI (for the in-place operation).
<i>srcDstStep</i>	Distance in bytes between starts of consecutive lines in the source and destination image buffer (for the in-place operation).
<i>roiSize</i>	Size of the source and destination ROI in pixels.
<i>threshold</i>	The threshold value to use for each pixel. In case of multi-channel data, an array of threshold values (one for each channel) is used.
<i>value</i>	The output value to be set for each pixel that satisfies the compare condition. In case of multi-channel data, an array of output values (one for each channel) is used.

**Description**

The function `ippiThreshold_GTVVal` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function thresholds pixels in the source image *pSrc* using the specified level *threshold*. Pixel values in the source image are compared to the *threshold* value for “greater than”.

If the result of the compare is true, the corresponding output pixel is set to the specified *value*. Otherwise, it is set to the source pixel value.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointer is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <code>roiSize</code> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <code>srcStep</code> , <code>dstStep</code> , or <code>srcDstStep</code> has a zero or negative value.

## Threshold\_LTVal

*Performs thresholding of pixel values in an image. Pixels that are less than threshold, are set to a specified value.*

---

### Syntax

#### Case 1: Not-in-place operation on one-channel data

```
IppStatus ippiThreshold_LTVal_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, Ipp<datatype> threshold, Ipp<datatype> value);
```

Supported values for `mod`:

8u_C1R	16u_C1R	16s_C1R	32f_C1R
--------	---------	---------	---------

#### Case 2: Not-in-place operation on multi-channel data

```
IppStatus ippiThreshold_LTVal_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, const Ipp<datatype> threshold[3], const Ipp<datatype> value[3]);
```

Supported values for `mod`:

8u_C3R	16u_C3R	16s_C3R	32f_C3R
8u_AC4R	16u_AC4R	16s_AC4R	32f_AC4R

```
IppStatus ippiThreshold_LTVal_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, const Ipp<datatype> threshold[4], const Ipp<datatype> value[4]);
```

Supported values for `mod`:

8u_C4R	16u_C4R	16s_C4R	32f_C4R
--------	---------	---------	---------

**Case 3: In-place operation on one-channel data**

```
IppStatus ippiThreshold_LTVal_<mod>(Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize
roiSize, Ipp<datatype> threshold, Ipp<datatype> value);
```

Supported values for mod:

8u\_C1IR      16u\_C1IR      16s\_C1IR      32f\_C1IR

**Case 4: In-place operation on multi-channel data**

```
IppStatus ippiThreshold_LTVal_<mod>(Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize
roiSize, const Ipp<datatype> threshold[3], const Ipp<datatype> value[3]);
```

Supported values for mod:

8u\_C3IR      16u\_C3IR      16s\_C3IR      32f\_C3IR  
8u\_AC4IR      16u\_AC4IR      16s\_AC4IR      32f\_AC4IR

```
IppStatus ippiThreshold_LTVal_<mod>(Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize
roiSize, const Ipp<datatype> threshold[4], const Ipp<datatype> value[4]);
```

Supported values for mod:

8u\_C4IR      16u\_C4IR      16s\_C4IR      32f\_C4IR

**Parameters**

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>pSrcDst</i>	Pointer to the source and destination image ROI (for the in-place operation).
<i>srcDstStep</i>	Distance in bytes between starts of consecutive lines in the source and destination image buffer (for the in-place operation).
<i>roiSize</i>	Size of the source and destination ROI in pixels.
<i>threshold</i>	The threshold value to use for each pixel. In case of multi-channel data, an array of threshold values (one for each channel) is used.
<i>value</i>	The output value to be set for each pixel that satisfies the compare condition. In case of multi-channel data, an array of output values (one for each channel) is used.

## Description

The function `ippiThreshold_LTVal` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function thresholds pixels in the source image *pSrc* using the specified level *threshold*. Pixel values in the source image are compared to the *threshold* value for “less than”. If the result of the compare is true, the corresponding output pixel is set to the specified *value*. Otherwise, it is set to the source pixel value.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointer is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> , <i>dstStep</i> , or <i>srcDstStep</i> has a zero or negative value.

## Threshold\_LTValGTVal

*Performs double thresholding of pixel values in an image buffer.*

---

## Syntax

### Case 1: Not-in-place operation on one-channel data

```
ippiStatus ippiThreshold_LTValGTVal_<mod>(const Ipp<datatype>* pSrc, int srcStep,
Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, Ipp<datatype> thresholdLT, Ipp<datatype>
valueLT, Ipp<datatype> thresholdGT, Ipp<datatype> valueGT);
```

Supported values for *mod*:

8u_C1R	16u_C1R	16s_C1R	32f_C1R
--------	---------	---------	---------

### Case 2: Not-in-place operation on multi-channel data

```
ippiStatus ippiThreshold_LTValGTVal_<mod>(const Ipp<datatype>* pSrc, int srcStep,
Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, const Ipp<datatype> thresholdLT[3],
const Ipp<datatype> valueLT[3], const Ipp<datatype> thresholdGT[3], const Ipp<datatype>
valueGT[3]);
```

Supported values for *mod*:

8u_C3R	16u_C3R	16s_C3R	32f_C3R
8u_AC4R	16u_AC4R	16s_AC4R	32f_AC4R

**Case 3: In-place operation on one-channel data**

```
IppStatus ippiThreshold_LTValGTVal_<mod>(Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize
roiSize, Ipp<datatype> thresholdLT, Ipp<datatype> valueLT, Ipp<datatype> thresholdGT,
Ipp<datatype> valueGT);
```

Supported values for mod:

8u\_C1IR      16u\_C1IR      16s\_C1IR      32f\_C1IR

**Case 4: In-place operation on multi-channel data**

```
IppStatus ippiThreshold_LTValGTVal_<mod>(Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize
roiSize, const Ipp<datatype> thresholdLT[3], const Ipp<datatype> valueLT[3], const
Ipp<datatype> thresholdGT[3], const Ipp<datatype> valueGT[3]);
```

Supported values for mod

8u\_C3IR      16u\_C3IR      16s\_C3IR      32f\_C3IR  
8u\_AC4IR      16u\_AC4IR      16s\_AC4IR      32f\_AC4IR

**Parameters**

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>pSrcDst</i>	Pointer to the source and destination image ROI (for the in-place operation).
<i>srcDstStep</i>	Distance in bytes between starts of consecutive lines in the source and destination image buffer (for the in-place operation).
<i>roiSize</i>	Size of the source and destination ROI in pixels.
<i>thresholdLT</i>	The lower threshold value to use for each pixel. In case of multi-channel data, an array of three lower threshold values (one for each color channel) is used.
<i>valueLT</i>	The lower output value to be set for each pixel that is less than <i>thresholdLT</i> . In case of multi-channel data, an array of 3 lower output values (one for each color channel) is used.
<i>thresholdGT</i>	The upper threshold value to use for each pixel. In case of multi-channel data, an array of three upper threshold values (one for each color channel) is used.
<i>valueGT</i>	The upper output value to be set for each pixel that exceeds <i>thresholdGT</i> . In case of multi-channel data, an array of three upper output values (one for each color channel) is used.

## Description

The function `ippiThreshold_LTValGTVal` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function thresholds pixels in the source image *pSrc* using two specified levels *thresholdLT* and *thresholdGT*. Pixel values in the source image are compared to these levels. If the pixel value is less than *thresholdLT*, the corresponding output pixel is set to *valueLT*. If the pixel value is greater than *thresholdGT*, the output pixel is set to *valueGT*. Otherwise, it is set to the source pixel value. The value of *thresholdLT* should be less than or equal to *thresholdGT*.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointer is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.
<code>ippStsThresholdErr</code>	Indicates an error when <i>thresholdLT</i> is greater than <i>thresholdGT</i> .
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> , <i>dstStep</i> , or <i>srcDstStep</i> has a zero or negative value.

## Example

The code example below illustrates thresholding with two levels.

```

IppStatus threshold( void ) {
    Ipp8u x[5*4];
    IppiSize roi = {5,4};
    int i;

    for( i=0; i<5*4; ++i ) x[i] = (Ipp8u)i;

    return ippiThreshold_LTValGTVal_8u_C1R( x, 5, roi, 2,1,6,7 );
}

```

The destination image *x* contains:

```

01 01 02 03 04
05 06 07 07 07
07 07 07 07 07
07 07 07 07 07

```

## ComputeThreshold\_Otsu

*Computes the value of the Otsu threshold.*

---

## Syntax

```

IppStatus ippiComputeThreshold_Otsu_8u_C1R(const Ipp8u* pSrc, int srcStep, IppiSize roiSize,
Ipp8u* pThreshold);

```

## Parameters

*pSrc*                                      Pointer to the source image ROI.

<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>roiSize</i>	Size of the image ROI in pixels.
<i>pThreshold</i>	Pointer to the Otsu threshold value.

## Description

The function `ippiComputeThreshold_Otsu` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function calculates the Otsu threshold ([\[Otsu79\]](#)) for the source image *pSrc* in accordance with the following formula:

$$\min_T (\sigma(pSrc(x, y) \leq T) + \sigma(pSrc(x, y) > T))$$

where  $0 \leq x < roiSize.width$ ,  $0 \leq y < roiSize.height$ , and  $T$  is the Otsu threshold.

$$\sigma(pSrc(x, y)) = \sqrt{\frac{\sum_{\substack{x < roiSize.width \\ y < roiSize.height \\ x, y \geq 0}} (pSrc(x, y) - mean)^2}{roiSize.height \cdot roiSize.width}}$$

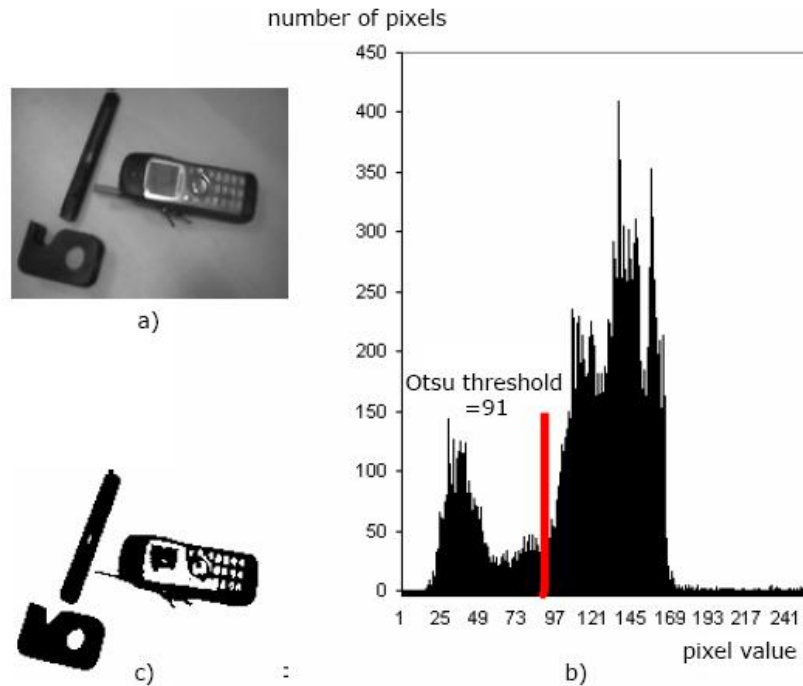
where

$$mean = \frac{\sum_{\substack{x < roiSize.width \\ y < roiSize.height \\ x, y \geq 0}} pSrc(x, y)}{roiSize.height \cdot roiSize.width}$$

The computed Otsu threshold can be used in the thresholding functions described above.

For example, the following figures shows how the Otsu threshold can be used for background/foreground selection. The figure a) below shows the initial image, the figure b) shows the histogram of the initial image with the red line indicating the computed Otsu threshold. The figure c) demonstrates the resulting image obtained by applying the function `ippiThreshold_8u_C1R` with a computed Otsu threshold value to the source image.

## Using Otsu Threshold for Background/Foreground Selection



### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointer is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <code>roiSize</code> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <code>srcStep</code> is less than or equal to 0.

## Compare Operations

This section describes functions that compare images. Each compare function writes its results to a one-channel `Ipp8u` output image. The output pixel is set to a non-zero value if the corresponding input pixel(s) satisfy the compare condition; otherwise, the output pixel is set to 0. You can compare either two images, or an image and a constant value, using the following compare conditions: "greater", "greater or equal", "less", "less or equal", "equal". Compare condition is specified as a function argument of `IppCmpOp` type (see [Structures and Enumerators](#) in Chapter 2 for more information). Images containing floating-point data can also be compared for being equal within a given tolerance `eps`.

For images with multi-channel data, the compare condition for a given pixel is true only when each color channel value of that pixel satisfies this condition.



## Compare

*Compares pixel values of two images using a specified compare operation.*

### Syntax

```
IppStatus ippiCompare_<mod>(const Ipp<datatype>* pSrc1, int src1Step, const Ipp<datatype>* pSrc2, int src2Step, Ipp8u* pDst, int dstStep, IppiSize roiSize, IppCmpOp ippCmpOp);
```

Supported values for mod:

8u_C1R	16u_C1R	16s_C1R	32f_C1R
8u_C3R	16u_C3R	16s_C3R	32f_C3R
8u_C4R	16u_C4R	16s_C4R	32f_C4R
8u_AC4R	16u_AC4R	16s_AC4R	32f_AC4R

### Parameters

<i>pSrc1, pSrc2</i>	Pointers to the source image ROIs.
<i>src1Step, src2Step</i>	Distances in bytes between starts of consecutive lines in the source images.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>roiSize</i>	Size of the source and destination ROI in pixels.
<i>ippCmpOp</i>	Compare operation to be used for comparing the pixel values.

### Description

The function `ippiCompare` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function compares the corresponding pixels of ROI in two source images *pSrc1*, *pSrc2* using the *ippCmpOp* compare operation, and writes the results to a one-channel `Ipp8u` image *pDst*. If the result of the compare is true, the corresponding output pixel is set to an `IPP_MAX_8U` value; otherwise, it is set to 0.

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointer is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <i>src1Step</i> , <i>src2Step</i> , or <i>dstStep</i> has a zero or negative value.

## CompareC

*Compares pixel values of a source image to a given value using a specified compare operation.*

---

### Syntax

#### Case 1: Operation on one-channel data

```
IppStatus ippCompareC_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype> value,
Ipp8u* pDst, int dstStep, IppiSize roiSize, IppCmpOp ippCmpOp);
```

Supported values for mod:

8u_C1R	16u_C1R	16s_C1R	32f_C1R
--------	---------	---------	---------

#### Case 2: Operation on multi-channel data

```
IppStatus ippCompareC_<mod>(const Ipp<datatype>* pSrc, int srcStep, const Ipp<datatype>
value[3], Ipp8u* pDst, int dstStep, IppiSize roiSize, IppCmpOp ippCmpOp);
```

Supported values for mod:

8u_C3R	16u_C3R	16s_C3R	32f_C3R
8u_AC4R	16u_AC4R	16s_AC4R	32f_AC4R

```
IppStatus ippCompareC_<mod>(const Ipp<datatype>* pSrc, int srcStep, const Ipp<datatype>
value[4], Ipp8u* pDst, int dstStep, IppiSize roiSize, IppCmpOp ippCmpOp);
```

Supported values for mod:

8u_C4R	16u_C4R	16s_C4R	32f_C4R
--------	---------	---------	---------

### Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>value</i>	The value to compare each pixel to. In case of multi-channel data, an array of separate values (one for each channel).
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>roiSize</i>	Size of the source and destination ROI in pixels.
<i>ippCmpOp</i>	Compare operation to be used for comparing the pixel values.

## Description

The function `ippiCompareC` is declared in the `ippi.h` file. It operates with ROI ([Regions of Interest in Intel IPP](#)).

This function compares pixels of the each channel of the source image ROI `pSrc` to a given *value* specified for each channel using the `ippCmpOp` compare operation, and writes the results to a one-channel `Ipp8u` image `pDst`. If the result of the compare is true, that is, all pixels of all channels meet the specified condition, then the corresponding output pixel is set to an `IPP_MAX_8U` value; otherwise, it is set to 0.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointer is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <code>roiSize</code> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <code>srcStep</code> or <code>dstStep</code> has a zero or negative value.

## Example

The code example below shows how to use the comparison function and create a mask image:

```

IppStatus compare ( void ) {
    Ipp8u x[5*4], y[5*4];
    IppiSize roi = {5,4};
    int i;
    for( i=0; i<5*4; ++i ) x[i] = (Ipp8u)i;
    return ippiCompareC_8u_C1R ( x, 5, 7, y, 5, roi, ippCmpGreater );
}

```

The mask image `y` contains:

```

00 00 00 00 00
00 00 00 FF FF
FF FF FF FF FF
FF FF FF FF FF

```

## CompareEqualEps

*Compares two images with floating-point data, testing whether pixel values are equal within a certain tolerance `eps`.*

## Syntax

```

IppStatus ippiCompareEqualEps_<mod>(const Ipp32f* pSrc1, int src1Step, const Ipp32f* pSrc2,
int src2Step, Ipp8u* pDst, int dstStep, IppiSize roiSize, Ipp32f eps);

```

Supported values for `mod`:

`32f_C1R`

`32f_C3R`

`32f_C4R``32f_AC4R`

## Parameters

<code>pSrc1, pSrc2</code>	Pointers to the source image ROIs.
<code>src1Step, src2Step</code>	Distances in bytes between starts of consecutive lines in the source images.
<code>pDst</code>	Pointer to the destination image ROI.
<code>dstStep</code>	Distance in bytes between starts of consecutive lines in the destination image.
<code>roiSize</code>	Size of the source and destination ROI in pixels.
<code>eps</code>	The tolerance value.

## Description

The function `ippiCompareEqualEps` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function tests if the corresponding pixels of ROI in two source images `pSrc1, pSrc2` are equal within the tolerance `eps`, and writes the results to a one-channel `Ipp8u` image `pDst`. If the absolute value of difference of the pixel values in `pSrc1` and `pSrc2` is less than or equal to `eps`, then the corresponding pixel in `pDst` is set to an `IPP_MAX_8U` value; otherwise the pixel in `pDst` is set to 0. For multi-channel images, the differences for all color channel values of a pixel must be within the `eps` tolerance for the compare condition to be true.

This function processes images with floating-point data only.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointer is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <code>roiSize</code> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <code>src1Step, src2Step, or dstStep</code> has a zero or negative value.
<code>ippStsEpsValErr</code>	Indicates an error condition if <code>eps</code> has a negative value.

## CompareEqualEpsC

Tests whether floating-point pixel values of an image are equal to a given value within a certain tolerance `eps`.

## Syntax

### Case 1: Operation on one-channel data

```
IppStatus ippiCompareEqualEpsC_32f_C1R(const Ipp32f* pSrc, int srcStep, Ipp32f value,
Ipp8u* pDst, int dstStep, IppiSize roiSize, Ipp32f eps);
```

## Case 2: Operation on multi-channel data

```
IppStatus ippiCompareEqualEpsC_<mod>(const Ipp32f* pSrc, int srcStep, const Ipp32f value[3],
Ipp8u* pDst, int dstStep, IppiSize roiSize, Ipp32f eps);
```

Supported values for `mod`:

32f\_C3R

32f\_AC4R

```
IppStatus ippiCompareEqualEpsC_32f_C4R(const Ipp32f* pSrc, int srcStep, const Ipp32f
value[4], Ipp8u* pDst, int dstStep, IppiSize roiSize, Ipp32f eps);
```

## Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>value</i>	The value to compare each pixel to. In case of multi-channel data, an array of separate values (one for each channel).
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>roiSize</i>	Size of the source and destination ROI in pixels.
<i>eps</i>	The tolerance value.

## Description

The function `ippiCompareEqualEpsC` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function tests if pixel values of the source image ROI *pSrc* are equal to a given constant *value* within the tolerance *eps*, and writes the results to a one-channel `Ipp8u` image *pDst*. If the absolute value of difference between the pixel value in *pSrc* and *value* is less than or equal to *eps*, then the corresponding pixel in *pDst* is set to an `IPP_MAX_8U` value; otherwise the pixel in *pDst* is set to 0. For multi-channel images, the differences between all color channel values of a pixel and the respective components of *value* must be within the tolerance *eps* for the compare condition to be true.

This function processes images with floating-point data only.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointer is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with a zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> or <i>dstStep</i> has a zero or negative value.

`ippStsEpsValErr`Indicates an error condition if *eps* has a negative value.

# Morphological Operations

This chapter describes the Intel® IPP image processing functions that perform morphological operations on images. The table below lists the functions described in more detail later in this chapter:

## Morphological Functions

Function Base Name	Operation
<a href="#">Dilate3x3</a>	Performs dilation of an image using a 3x3 mask.
<a href="#">Erode3x3</a>	Performs erosion of an image using a 3x3 mask.
<a href="#">Dilate</a>	Performs dilation of an image using a general rectangular mask.
<a href="#">Erode</a>	Performs erosion of an image using a general rectangular mask.
<a href="#">MorphologyInitAlloc</a>	Allocates and initializes morphology state structures for the erosion or dilatation operation.
<a href="#">MorphologyFree</a>	Frees memory allocated for the morphology state structure.
<a href="#">MorphologyInit</a>	Initializes morphology state structures for the erosion or dilatation operation.
<a href="#">MorphologyGetSize</a>	Computes the size of the morphology state structures for the erosion or dilatation operation.
<a href="#">DilateBorderReplicate</a>	Performs dilation of an image.
<a href="#">ErodeBorderReplicate</a>	Performs erosion of an image.
<a href="#">MorphAdvInitAlloc</a>	Allocates and initializes morphology state structure for advanced morphology operations.
<a href="#">MorphAdvFree</a>	Frees memory allocated for the advanced morphology state structure.
<a href="#">MorphAdvInit</a>	Initializes morphology state structure for advanced morphology operations.
<a href="#">MorphAdvGetSize</a>	Computes the size of the advanced morphology state structure.
<a href="#">MorphOpenBorder</a>	Performs opening operation of an image.

Function Base Name	Operation
<a href="#">MorphCloseBorder</a>	Performs closing operation of an image.
<a href="#">MorphTophatBorder</a>	Performs top-hat operation of an image.
<a href="#">MorphBlackhatBorder</a>	Performs black-hat operation of an image.
<a href="#">MorphGradientBorder</a>	Calculates morphological gradient of an image.
<a href="#">MorphGrayInitAlloc</a>	Allocates and initializes gray-kernel morphology state structure.
<a href="#">MorphGrayFree</a>	Frees memory allocated for the gray-kernel morphology state structure.
<a href="#">MorphGrayInit</a>	Initializes gray-kernel morphology state structure.
<a href="#">MorphGrayGetSize</a>	Computes the size of the gray-kernel morphology state structure.
<a href="#">GrayDilateBorder</a>	Performs gray-kernel dilation of an image.
<a href="#">GrayErodeBorder</a>	Performs gray-kernel erosion of an image.
<a href="#">MorphReconstructGetBufferSize</a>	Computes the size of the buffer for morphological reconstruction operation.
<a href="#">MorphReconstructDilate</a>	Reconstructs an image by dilation.
<a href="#">MorphReconstructErode</a>	Reconstructs an image by erosion.

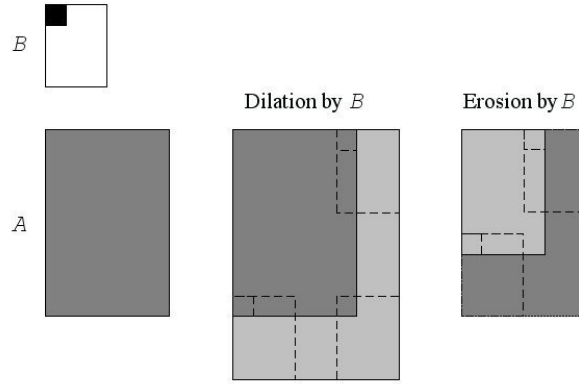
Generally, the *erosion* and *dilation* smooth the boundaries of objects without significantly changing their area. *Opening* and *closing* smooth thin projections or gaps. Morphological operations use a structuring element (SE) that is a user-defined rectangular mask, or for some functions - symmetric 3x3 mask.

In a more general sense, morphological operations involve an image *A* called the *object of interest* and a kernel element *B* called the *structuring element*. The image and structuring element could be in any number of dimensions, but the most common use is with a 2D binary image, or with a 3D gray scale image. The element



$B$  is most often a square or a circle, but it could be of any shape. Just like in convolution,  $B$  is a kernel or template with an anchor point. Figure "Dilation and Erosion of  $A$  by  $B$ " shows dilation and erosion of object  $A$  by  $B$ . In the figure,  $B$  is rectangular with an anchor point at upper left shown as a dark square.

### Dilation and Erosion of $A$ by $B$



Let  $B_t$  is the SE with pixel  $t$  in the anchor position,  $\bar{B}$  is transpose of the SE.

Dilation of binary image  $A$   $\{A(t) = 1, t \in A; 0 - \text{otherwise}\}$  by binary SE  $B$  is

$$A \oplus B = \{t : B_t \cap A \neq \emptyset\}$$

It means that every pixel is in the set, if the intersection is not null. That is, a pixel under the anchor point of  $B$  is marked "on", if at least one pixel of  $B$  is inside of  $A$ .

Erosion of the binary image  $A$  by the binary SE  $B$  is

$$A \ominus B = \{t : B_t \subseteq A\}$$

That is, a pixel under the anchor of  $B$  is marked "on", if  $B$  is entirely within  $A$ .

Generalization of dilation and erosion for the gray-scale image  $A$  and the binary SE  $B$  is

$$A \oplus B = \left\{ \max_{u \in B_t \cap A} A(u) \right\}, \quad A \ominus B = \left\{ \min_{u \in B_t \cap A} A(u) \right\}$$

Generalization of dilation and erosion for the gray-scale image  $A$  and the gray-scale SE  $B$  is

$$A \oplus B = \left\{ \max_{u \in B_t \cap A} (A(u) + B(u - t)) \right\}, \quad A \ominus B = \left\{ \min_{u \in B_t \cap A} (A(u) + B(u - t)) \right\}$$

Opening operation of  $A$  by  $B$  is  $A \circ B = (A \oplus B) \ominus \bar{B}$ .

Closing operation of  $A$  by  $B$  is  $A \bullet B = (A \ominus B) \oplus \bar{B}$ .

Top-hat operation of  $A$  by  $B$  is  $A - A \circ B$ .

Black-hat operation of  $A$  by  $B$  is  $A \bullet B - A$ .

Black-hat operation of  $A$  by  $B$  is  $A \oplus B - A \ominus B$ .

Morphological reconstruction  $\rho_A(c)$  of an image  $A$  from the image  $c$ ,  $A(t) \geq c(t) \forall t$  by dilation with the mask  $B$  is an image

$$C_k : k = \min_i C_i \equiv C_{i-1}, C_0 = C, C_{i+1}(t) = \min\{ (C_i \oplus B)(t), A(t) \}$$

Morphological reconstruction  $\rho_A(c)$  of an image  $A$  from the image  $c$ ,  $A(t) \leq c(t) \forall t$  by erosion with the mask  $B$  is an image

$$C_k : k = \min_i C_i \equiv C_{i-1}, C_0 = C, C_{i+1}(t) = \max\{ (C_i \ominus B)(t), A(t) \}$$

Figure "Morphological Operations Performed by Intel IPP" presents the results of different morphological operations applied to the initial image. In these operations, the SE is a matrix of 3x3 size with the following values:

$$\begin{bmatrix} -8 & 0 & -8 \\ 0 & 8 & 0 \\ -8 & 0 & -8 \end{bmatrix}$$

for common and advanced morphology, and











$$\begin{bmatrix} -5 & 0 & -5 \\ 0 & 5 & 0 \\ -5 & 0 & -5 \end{bmatrix}$$

for gray morphology.

The anchor cell is in the center cell (1,1) of the matrix.

### Morphological Operations Performed by Intel IPP

---

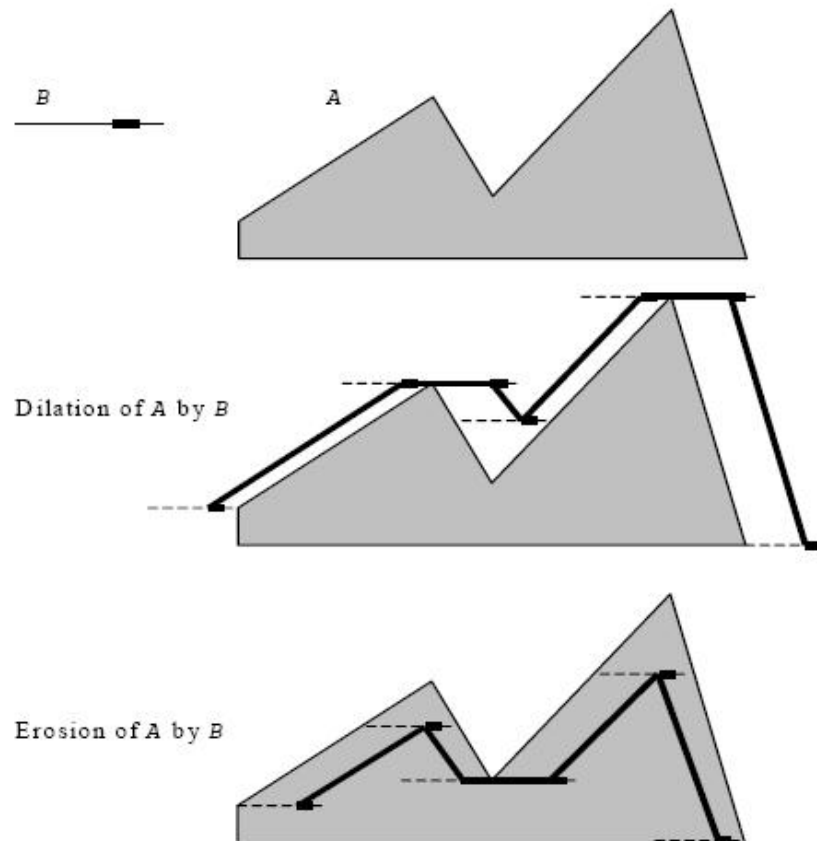
	Initial image
	Erosion
	Dilation
	Opening
	Closing
	Top-hat
	Black-hat
	Morphological gradient
	Gray-kernel erosion
	Gray-kernel dilation

---

## Flat Structuring Elements for Grayscale Image

Erosion and dilation can be done in 3D space, that is, with gray levels. 3D structuring elements can be used, but the simplest and the best way is to use a flat structuring element  $B$ . Figure "1D Cross Section of Dilation and Erosion of  $A$  by  $B$ " is a 1D cross section of dilation and erosion of a grayscale image  $A$  by a flat structuring element  $B$ . In the figure,  $B$  has an anchor slightly to the right of the center as shown by the dark mark on  $B$ .

### 1D Cross Section of Dilation and Erosion of $A$ by $B$



In Figure "1D Cross Section of Dilation and Erosion of  $A$  by  $B$ " above, dilation is mathematically

$$\sup_{y \in B_i} A$$

and erosion is

$$\inf_{y \in B_t} A$$

Many solutions and hints for use of these functions can be found in Intel® IPP Samples. See *Intel® IPP Image Processing and Media Processing Samples* downloadable from <http://www.intel.com/cd/software/products/asmo-na/eng/220046.htm> .

## Dilate3x3

*Performs dilation of an image using a 3x3 mask.*

### Syntax

#### Case 1: Not-in-place operation

```
IppStatus ippiDilate3x3_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pDst,
int dstStep, IppiSize roiSize);
```

Supported values for mod:

8u_C1R	16u_C1R	32f_C1R	64f_C1R
8u_C3R	16u_C3R	32f_C3R	
8u_C4R	16u_C4R	32f_C4R	
8u_AC4R	16u_AC4R	32f_AC4R	

#### Case 2: In-place operation

```
IppStatus ippiDilate3x3_<mod>(Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize);
```

Supported values for mod:

8u_C1IR	16u_C1IR	32f_C1IR
8u_C3IR	16u_C3IR	32f_C3IR
8u_C4IR	16u_C4IR	32f_C4IR
8u_AC4IR	16u_AC4IR	32f_AC4IR

### Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>pSrcDst</i>	Pointer to the source and destination image ROIs for the in-place operation.
<i>srcDstStep</i>	Distance in bytes between starts of consecutive lines in the source and destination image buffer for the in-place operation.
<i>roiSize</i>	Size of the source and destination ROI in pixels.

## Description

The function `ippiDilate3x3` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function performs dilation of a rectangular ROI area inside a one-, three-, or four-channel 2D image using a symmetric 3x3 mask.

Source and destination images can be of different sizes, but the ROI size is the same for both images. The output pixel is set to the maximum of the corresponding input pixel and its eight neighboring pixels.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <code>pSrc</code> , <code>pDst</code> , or <code>pSrcDst</code> is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <code>roiSize</code> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <code>srcStep</code> , <code>dstStep</code> or <code>srcDstStep</code> has a zero or negative value.

## Example

The code example below shows how to use the dilation function.

```

IppStatus dilate( void ) {
    Ipp8u x[7*5];
    IppiSize roi = {7,5};
    ippiSet_8u_C1R( 0, x, 7, roi );
    x[2*7+3] = 1;
    roi.width = roi.width - 2;
    roi.height = roi.height - 2;
    return ippiDilate3x3_8u_C1IR( x+7+1, 7, roi );
}

```

The destination image `x` contains:

```

00 00 00 00 00 00 00
00 00 01 01 01 00 00
00 00 01 01 01 00 00
00 00 01 01 01 00 00
00 00 00 00 00 00 00

```

## Erode3x3

*Performs erosion of an image using a 3x3 mask.*

---

## Syntax

### Case 1: Not-in-place operation

```

IppStatus ippiErode3x3_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pDst,
int dstStep, IppiSize roiSize);

```

Supported values for `mod`:

8u_C1R	16u_C1R	32f_C1R	64f_C1R
8u_C3R	16u_C3R	32f_C3R	

8u_C4R	16u_C4R	32f_C4R
8u_AC4R	16u_AC4R	32f_AC4R

## Case 2: In-place operation

```
IppStatus ippiErode3x3_<mod>(Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize);
```

Supported values for mod:

8u_C1IR	16u_C1IR	32f_C1IR
8u_C3IR	16u_C3IR	32f_C3IR
8u_C4IR	16u_C4IR	32f_C4IR
8u_AC4IR	16u_AC4IR	32f_AC4IR

## Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>pSrcDst</i>	Pointer to the source and destination image ROIs for the in-place operation.
<i>srcDstStep</i>	Distance in bytes between starts of consecutive lines in the source and destination image buffer for the in-place operation.
<i>roiSize</i>	Size of the source and destination ROI in pixels.

## Description

The function `ippiErode3x3` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function performs erosion of a rectangular ROI area inside a one-, three-, or four-channel 2D image using a symmetric 3x3 mask.

Source and destination images can have different size, but the ROI size is the same for both images. The output pixel is set to the minimum of the corresponding input pixel and its 8 neighboring pixels.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pSrc</i> , <i>pDst</i> , or <i>pSrcDst</i> is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with a zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> , <i>dstStep</i> or <i>srcDstStep</i> has a zero or negative value.

## Dilate

*Performs dilation of an image using a specified mask.*

---

### Syntax

#### Case 1: Not-in-place operation

```
IppStatus ippiDilate_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pDst,
int dstStep, IppiSize roiSize, const char* pMask, IppiSize maskSize, IppiPoint anchor);
```

Supported values for mod:

8u_C1R	16u_C1R	32f_C1R
8u_C3R	16u_C3R	32f_C3R
8u_C4R	16u_C4R	32f_C4R
8u_AC4R	16u_AC4R	32f_AC4R

#### Case 2: In-place operation

```
IppStatus ippiDilate_<mod>(Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize, const
char* pMask, IppiSize maskSize, IppiPoint anchor);
```

Supported values for mod:

8u_C1IR	16u_C1IR	32f_C1IR
8u_C3IR	16u_C3IR	32f_C3IR
8u_AC4IR	16u_AC4IR	32f_AC4IR

### Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>pSrcDst</i>	Pointer to the source and destination image ROIs for the in-place operation.
<i>srcDstStep</i>	Distance in bytes between starts of consecutive lines in the source and destination image buffer for the in-place operation.
<i>roiSize</i>	Size of the source and destination ROI in pixels.
<i>pMask</i>	Pointer to mask values. Only pixels that correspond to nonzero mask values are taken into account during operation.
<i>maskSize</i>	Size of the mask in pixels.
<i>anchor</i>	Anchor cell specifying the mask alignment with respect to the position of the input pixel.



## Description

The function `ippiDilate` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function performs dilation of a rectangular ROI area inside a one-, three-, or four-channel 2D image using a specified mask `pMask` of size `maskSize` and alignment `anchor`.

Source and destination images can be of different sizes, but the ROI size is the same for both images. The output pixel is set to the maximum of the corresponding input pixel and its neighboring pixels that are picked out by the nonzero mask values. In the four-channel image the alpha channel is not processed.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <code>pSrc</code> , <code>pDst</code> , or <code>pSrcDst</code> is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <code>roiSize</code> or <code>maskSize</code> has a field with a zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <code>srcStep</code> , <code>dstStep</code> or <code>srcDstStep</code> has a zero or negative value.
<code>ippStsZeroMaskValuesErr</code>	Indicates an error condition if all mask values are equal to zero.

## Erode

Performs erosion of an image using a specified mask.

## Syntax

### Case 1: Not-in-place operation

```
ippStatus ippiErode_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pDst,
int dstStep, IppiSize roiSize, const char* pMask, IppiSize maskSize, IppiPoint anchor);
```

Supported values for `mod`:

8u_C1R	16u_C1R	32f_C1R
8u_C3R	16u_C3R	32f_C3R
8u_C4R	16u_C4R	32f_C4R
8u_AC4R	16u_AC4R	32f_AC4R

### Case 2: In-place operation

```
ippStatus ippiErode_<mod>(Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize, const
char* pMask, IppiSize maskSize, IppiPoint anchor);
```

Supported values for `mod`:

8u_C1IR	16u_C1IR	32f_C1IR
8u_C3IR	16u_C3IR	32f_C3IR
8u_AC4IR	16u_AC4IR	32f_AC4IR

## Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>pSrcDst</i>	Pointer to the source and destination image ROIs for the in-place operation.
<i>srcDstStep</i>	Distance in bytes between starts of consecutive lines in the source and destination image buffer for the in-place operation.
<i>roiSize</i>	Size of the source and destination ROI in pixels.
<i>pMask</i>	Pointer to mask values. Only pixels that correspond to nonzero mask values are taken into account during operation.
<i>maskSize</i>	Size of the mask in pixels.
<i>anchor</i>	Anchor cell specifying the mask alignment with respect to the position of the input pixel.

## Description

The function `ippiErode` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function performs erosion of a rectangular ROI area inside a one-, three-, or four-channel 2D image using a specified mask *pMask* of size *maskSize* and alignment *anchor*.

Source and destination images can be of different sizes, but the ROI size is the same for both images (*roiSize*). The output pixel is set to the minimum of the corresponding input pixel and its neighboring pixels that are picked out by the nonzero mask values. In the four-channel image the alpha channel is not processed.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pSrc</i> , <i>pDst</i> , or <i>pSrcDst</i> is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> or <i>maskSize</i> has a field with a zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> , <i>dstStep</i> or <i>srcDstStep</i> has a zero or negative value.
<code>ippStsZeroMaskValuesErr</code>	Indicates an error condition if all mask values are equal to zero.

## MorphologyInitAlloc

*Allocates and initializes morphology state structure for erosion or dilation operation.*

### Syntax

```
IppStatus ippiMorphologyInitAlloc_<mod>(int roiWidth, const Ipp8u* pMask, IppiSize maskSize,
IppiPoint anchor, IppiMorphState** ppState);
```

Supported values for `mod`:

8u_C1R	32f_C1R
8u_C3R	32f_C3R
8u_C4R	32f_C4R

### Parameters

<code>roiWidth</code>	Maximal width of the image ROI in pixels, that can be processed using the allocated structure.
<code>pMask</code>	Pointer to the mask.
<code>maskSize</code>	Size of the mask in pixels.
<code>anchor</code>	Coordinates of the anchor cell.
<code>ppState</code>	Pointer to the pointer to the morphology state structure.

### Description

The function `ippiMorphologyInitAlloc` is declared in the `ippcv.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function allocates memory, initializes the morphology state structure and returns the pointer `ppState` to it. It is used by the functions [DilateBorderReplicate](#) and [ErodeBorderReplicate](#) that perform morphological operations on the source image pixels corresponding to non-zero values of the structuring element `pMask`. The anchor cell `anchor` is positioned in the arbitrary point in the structuring element and is used for positioning the structuring element.

[Example](#) shows how to use the function `ippiMorphologyInitAlloc_8u_C1R`.



**NOTE.** The structure can be used to process an image with ROI that does not exceed the specified maximum width `roiWidth`.

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error when one of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <code>maskSize</code> has a field with a zero or negative value, or if <code>roiWidth</code> is less than 1.
<code>ippStsAnchorErr</code>	Indicates an error if <code>anchor</code> is outside the mask.

## MorphologyFree

*Frees memory allocated for the morphology state structure.*

---

### Syntax

```
IppStatus ippiMorphologyFree(IppiMorphState* pState);
```

### Parameters

*pState*                                      Pointer to the morphology state structure.

### Description

The function `ippiMorphologyFree` is declared in the `ippcv.h` file. This function frees memory allocated by the function `MorphologyInitAlloc` for the morphology state structure *pState*.

### Return Values

`ippStsNoErr`                                      Indicates no error. Any other value indicates an error or a warning.  
`ippStsNullPtrErr`                                      Indicates an error condition if *pState* is NULL.

## MorphologyInit

*Initializes morphology state structure for erosion or dilation operation.*

---

### Syntax

```
IppStatus ippiMorphologyInit_<mod>(int roiWidth, const Ipp8u* pMask, IppiSize maskSize,  
IppiPoint anchor, IppiMorphState* pState);
```

Supported values for *mod*:

<code>8u_C1R</code>	<code>32f_C1R</code>
<code>8u_C3R</code>	<code>32f_C3R</code>
<code>8u_C4R</code>	<code>32f_C4R</code>

### Parameters

*roiWidth*                                      Maximal width of the image ROI in pixels, that can be processed using the allocated structure.  
*pMask*    Pointer to the mask.  
*maskSize*                                      Size of the mask in pixels.  
*anchor*    Coordinates of the anchor cell.  
*pState*    Pointer to the morphology state structure.

## Description

The function `ippiMorphologyInit` is declared in the `ippcv.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function initializes the morphology state structure `pState` in the external buffer. Its size should be computed by the function `MorphologyGetSize`. This structure is used by the functions `DilateBorderReplicate` and `ErodeBorderReplicate` that perform morphological operations on the source image pixels corresponding to non-zero values of the structuring element (mask) `pMask`. The anchor cell `anchor` is positioned in the arbitrary point in the structuring element and is used for positioning the structuring element.



**NOTE.** The structure can be used to process an image with ROI that does not exceed the specified maximum width `roiWidth`.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error when one of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <code>maskSize</code> has a field with a zero or negative value, or if <code>roiWidth</code> is less than 1.
<code>ippStsAnchorErr</code>	Indicates an error if <code>anchor</code> is outside the mask.

## MorphologyGetSize

*Computes the size of the morphology state structure.*

### Syntax

```
IppStatus ippiMorphologyGetSize_<mod>(int roiWidth, const Ipp8u* pMask, IppiSize maskSize, int* pSize);
```

Supported values for `mod`:

<code>8u_C1R</code>	<code>32f_C1R</code>
<code>8u_C3R</code>	<code>32f_C3R</code>
<code>8u_C4R</code>	<code>32f_C4R</code>

### Parameters

<code>roiWidth</code>	Maximal width of the image ROI in pixels, that can be processed using the allocated structure.
<code>pMask</code>	Pointer to the mask.
<code>maskSize</code>	Size of the mask in pixels.
<code>pSize</code>	Pointer to the size of the morphology state structure.

## Description

The function `ippiMorphologyGetSize` is declared in the `ippcv.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function computes the size of the morphology state structure `pState`. This function should be run prior to the function `MorphologyInit`.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error when one of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <code>maskSize</code> has a field with a zero or negative value, or if <code>roiWidth</code> is less than 1.

## DilateBorderReplicate

*Performs dilation of an image.*

---

## Syntax

```
IppStatus ippiDilateBorderReplicate_<mod>(const Ipp<datatype>* pSrc, int srcStep,
Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, IppiBorderType borderType,
IppiMorphState* pState);
```

Supported values for `mod`:

<code>8u_C1R</code>	<code>32f_C1R</code>
<code>8u_C3R</code>	<code>32f_C3R</code>
<code>8u_C4R</code>	<code>32f_C4R</code>

## Parameters

<code>pSrc</code>	Pointer to the source image ROI.
<code>srcStep</code>	Distance in bytes between starts of consecutive lines in the source image.
<code>pDst</code>	Pointer to the destination image ROI.
<code>dstStep</code>	Distance in bytes between starts of consecutive lines in the destination image.
<code>roiSize</code>	Size of the source and destination image ROI.
<code>borderType</code>	Type of border; the possible value is <code>ippBorderRepl</code> , which means that a replicated border is used.
<code>pState</code>	Pointer to the morphology state structure.

## Description

The function `ippiDilateBorderReplicate` is declared in the `ippcv.h`. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function performs dilation of a rectangular ROI area inside a one-, three-, or four-channel 2D image using a specified in the morphology state structure *pState* mask and the anchor cell. This structure must be initialized by the functions [MorphologyInitAlloc](#) or [MorphologyInit](#) beforehand.

The output pixel is set to the maximum of the corresponding input pixel and its neighboring pixels that are picked out by the nonzero mask values.



**NOTE.** The function can process only images with ROI that does not exceed the specified by the initialization functions maximum width *roiWidth*.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with a zero or negative value, or if <i>roiSize.width</i> is greater than maximum ROI <i>roiWidth</i> passed to the initialization functions.
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> or <i>dstStep</i> is less than <i>roiSize.width</i> * <i>&lt; pixelSize &gt;</i> .
<code>ippStsNotEvenStepErr</code>	Indicates an error condition if one of the step values is not divisible by 4 for floating-point images..
<code>ippStsBorderErr</code>	Indicates an error condition if <i>borderType</i> has an illegal value.

## Example

The code example below shows how to use the function `ippiDilateBorderReplicate_8u_C1R`.

```
void func_morf()
{
    IppiMorphState* ppState;
    IppiSize roiSize = {5, 5};
    Ipp8u pMask[3*3] = {1, 1, 1,
                        1, 0, 1,
                        1, 1, 1};
    IppiSize maskSize = {3, 3};
    IppiPoint anchor = {1, 1};
    Ipp8u pSrc[5*5] = { 1, 2, 4, 1, 2,
                        5, 1, 2, 1, 2,
                        1, 2, 1, 2, 1,
                        2, 1, 5, 1, 2};

    int srcStep = 5;
    int dstStep = 5;
    Ipp8u pDst[5*5];
    int dstSize = 5;

    ippiMorphologyInitAlloc_8u_C1R( roiSize.width, pMask, maskSize, anchor,
                                    &ppState);
    ippiDilateBorderReplicate_8u_C1R( pSrc, srcStep, pDst, dstStep, roiSize,
                                    ippBorderRepl, ppState);
}
```

```
   ippiMorphologyFree(ppState);
}
```

```
Result: pDst-> 5 5 4 4 2
               5 5 4 4 2
               5 5 5 5 2
               2 5 2 5 2
               2 5 5 5 2
```

## ErodeBorderReplicate

*Performs erosion of an image.*

---

### Syntax

```
IppStatus ippiErodeBorderReplicate_<mod>(const Ipp<datatype>* pSrc, int srcStep,
Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, IppiBorderType borderType,
IppiMorphState* pState);
```

Supported values for mod:

8u_C1R	32f_C1R
8u_C3R	32f_C3R
8u_C4R	32f_C4R

### Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>roiSize</i>	Size of the source and destination image ROI.
<i>borderType</i>	Type of border; the possible value is <code>ippiBorderRep1</code> , which means that a replicated border is used.
<i>pState</i>	Pointer to the morphology state structure.

### Description

The function `ippiErodeBorderReplicate` is declared in the `ippcv.h`. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function performs erosion of a rectangular ROI area inside a one-, three-, or four-channel 2D image using a specified in the morphology state structure *pState* mask and the anchor cell. This structure must be initialized by the functions [MorphologyInitAlloc](#) or [MorphologyInit](#) beforehand.

The output pixel is set to the maximum of the corresponding input pixel and its neighboring pixels that are picked out by the nonzero mask values.





**NOTE.** The function can process only images with ROI that does not exceed the specified by the initialization functions maximum width *roiWidth*.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with a zero or negative value, or if <i>roiSize.width</i> is greater than maximum ROI <i>roiWidth</i> passed to the initialization functions.
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> or <i>dstStep</i> is less than <i>roiSize.width</i> * < <i>pixelSize</i> >.
<code>ippStsNotEvenStepErr</code>	Indicates an error condition if one of the step values is not divisible by 4 for floating-point images.
<code>ippStsBorderErr</code>	Indicates an error condition if <i>borderType</i> has an illegal value.

## MorphAdvInitAlloc

*Allocates and initializes morphology state structure for advanced morphology operations.*

### Syntax

```
IppStatus ippMorphAdvInitAlloc_<mod>(IppiMorphAdvState** ppState, IppiSize roiSize, const Ipp8u* pMask, IppiSize maskSize, IppiPoint anchor);
```

Supported values for mod:

<code>8u_C1R</code>	<code>32f_C1R</code>
<code>8u_C3R</code>	<code>32f_C3R</code>
<code>8u_C4R</code>	<code>32f_C4R</code>

### Parameters

<i>ppState</i>	Pointer to the pointer to the advanced morphology state structure.
<i>roiSize</i>	Maximal size of the image ROI in pixels, that can be processed using the allocated structure.
<i>pMask</i>	Pointer to the mask.
<i>maskSize</i>	Size of the mask in pixels.
<i>anchor</i>	Coordinates of the anchor cell.

### Description

The function `ippMorphAdvInitAlloc` is declared in the `ippcv.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function allocates memory, initializes the advanced morphology state structure and returns the pointer *pState* to it. It is used by the functions [MorphOpenBorder](#), [MorphCloseBorder](#), [MorphTophatBorder](#), [MorphBlackhatBorder](#), and [MorphGradientBorder](#) that perform advanced morphological operations (opening, closing, top-hat, black-hat, and gradient) on the source image pixels corresponding to non-zero values of the structuring element *pMask*. The anchor cell *anchor* is positioned in the arbitrary point in the structuring element and is used for positioning the structuring element.




---

**WARNING.** The function can process only images with ROI that does not exceed the specified by the initialization functions maximum width *roiWidth*.

---

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error when one of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <i>maskSize</i> or if <i>roiSize</i> has a field with a zero or negative value.
<code>ippStsAnchorErr</code>	Indicates an error if <i>anchor</i> is outside the mask.

## MorphAdvFree

*Frees memory allocated for the advanced morphology state structure.*

---

## Syntax

```
IppStatus ippMorphAdvFree(IppiMorphAdvState* pState);
```

## Parameters

*pState*                      Pointer to the advanced morphology state structure.

## Description

The function `ippMorphAdvFree` is declared in the `ippcv.h` file. This function frees memory allocated by the function [MorphAdvInitAlloc](#) for the advanced morphology state structure *pState*.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pState</i> is <code>NULL</code> .

## MorphAdvInit

*Initializes morphology state structure for advanced morphology operations.*

### Syntax

```
IppStatus ippMorphologyInit_<mod>(IppiMorphAdvState* pState, IppiSize roiSize, const Ipp8u* pMask, IppiSize maskSize, IppiPoint anchor);
```

Supported values for mod:

8u_C1R	32f_C1R
8u_C3R	32f_C3R
8u_C4R	32f_C4R

### Parameters

<i>pState</i>	Pointer to the advanced morphology state structure.
<i>roiSize</i>	Maximal size of the image ROI in pixels, that can be processed using the allocated structure.
<i>pMask</i>	Pointer to the mask.
<i>maskSize</i>	Size of the mask in pixels.
<i>anchor</i>	Coordinates of the anchor cell.

### Description

The function `ippMorphAdvInit` is declared in the `ippcv.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#) ).

This function initializes the advanced morphology state structure *pState* in the external buffer. Its size should be computed by the function [MorphAdvGetSize](#). `ippMorphAdvInit` is used by the functions [MorphOpenBorder](#), [MorphCloseBorder](#), [MorphTophatBorder](#), [MorphBlackhatBorder](#), and [MorphGradientBorder](#) that perform advanced morphological operations (opening, closing, top-hat, black-hat, and gradient) on the source image pixels corresponding to non-zero values of the structuring element (mask) *pMask*. The anchor cell *anchor* is positioned in the arbitrary point in the structuring element and is used for positioning the structuring element.



**NOTE.** The function can be used to process images with ROI that does not exceed the specified maximum width and height *roiSize*.

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error when one of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <i>maskSize</i> or if <i>roiSize</i> has a field with a zero or negative value.
<code>ippStsAnchorErr</code>	Indicates an error if <i>anchor</i> is outside the mask.

## MorphAdvGetSize

*Computes the size of the advanced morphology state structure.*

---

### Syntax

```
IppStatus ippMorphAdvGetSize_<mod>(IppiSize roiSize, const Ipp8u* pMask, IppiSize maskSize, int* pSize);
```

Supported values for `mod`:

8u_C1R	32f_C1R
8u_C3R	32f_C3R
8u_C4R	32f_C4R

### Parameters

<code>roiSize</code>	Maximal size of the image ROI in pixels, that can be processed using the allocated structure.
<code>pMask</code>	Pointer to the mask.
<code>maskSize</code>	Size of the mask in pixels.
<code>pSize</code>	Pointer to the size of the advanced morphology state structure.

### Description

The function `ippMorphAdvGetSize` is declared in the `ippcv.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function computes the size of the advanced morphology state structure `pState`. This function should be run prior to the function `MorphAdvInit`.

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error when one of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <code>maskSize</code> or if <code>roiSize</code> has a field with a zero or negative value.

## MorphOpenBorder

*Performs opening of an image.*

---

### Syntax

```
IppStatus ippMorphOpenBorder_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, IppiBorderType borderType, IppiMorphAdvState* pState);
```

Supported values for `mod`:

8u_C1R	32f_C1R
--------	---------

8u_C3R	32f_C3R
8u_C4R	32f_C4R

## Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>roiSize</i>	Size of the source and destination image ROI.
<i>borderType</i>	Type of border; the possible value is <code>ippBorderRepl</code> , which means that a replicated border is used.
<i>pState</i>	Pointer to the morphology state structure.

## Description

The function `ippiMorphOpenBorder` is declared in the `ippcv.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function performs opening of a rectangular ROI area inside a one-, three-, or four-channel 2D image using a specified in the advanced morphology state structure *pState* mask and anchor cell. This structure must be initialized by the functions [MorphAdvInitAlloc](#) or [MorphAdvInit](#) beforehand.

The result is equivalent to successive dilation of the source image by the structured element (mask) and erosion by the reverted structured element.



**NOTE.** The function can process only images with ROI that does not exceed the maximum width and height *roiSize* specified by the initialization functions.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with a zero or negative value, or if one of ROI width or height is greater than corresponding size of ROI passed to the initialization functions.
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> or <i>dstStep</i> is less than $roiSize.width * <pixelSize>$ .
<code>ippStsNotEvenStepErr</code>	Indicates an error condition if one of the step values is not divisible by 4 for floating-point images.
<code>ippStsBorderErr</code>	Indicates an error condition if <i>borderType</i> has an illegal value.

## MorphCloseBorder

*Performs closing of an image.*

---

### Syntax

```
IppStatus ippiMorphCloseBorder_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, IppiBorderType borderType, IppiMorphAdvState* pState);
```

Supported values for mod:

8u_C1R	32f_C1R
8u_C3R	32f_C3R
8u_C4R	32f_C4R

### Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>roiSize</i>	Size of the source and destination image ROI.
<i>borderType</i>	Type of border; the possible value is <code>ippiBorderRepl</code> , which means that a replicated border is used.
<i>pState</i>	Pointer to the morphology state structure.

### Description

The function `ippiMorphCloseBorder` is declared in the `ippcv.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function performs closing of a rectangular ROI area inside a one-, three-, or four-channel 2D image using a specified in the advanced morphology state structure *pState* mask and anchor cell. This structure must be initialized by the functions [MorphAdvInitAlloc](#) or [MorphAdvInit](#) beforehand.

The result is equivalent to successive erosion of the source image by the structured element (mask) and dilation by the reverted structured element.



---

**NOTE.** The function can process only images with ROI that does not exceed the maximum width and height *roiSize* specified by the initialization functions.

---

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointers is <code>NULL</code> .

<code>ippStsSizeErr</code>	Indicates an error condition if <code>roiSize</code> has a field with a zero or negative value, or if one of ROI width or height is greater than corresponding size of ROI passed to the initialization functions.
<code>ippStsStepErr</code>	Indicates an error condition if <code>srcStep</code> or <code>dstStep</code> is less than <code>roiSize.width * &lt;pixelSize&gt;</code> .
<code>ippStsNotEvenStepErr</code>	Indicates an error condition if one of the step values is not divisible by 4 for floating-point images.
<code>ippStsBorderErr</code>	Indicates an error condition if <code>borderType</code> has an illegal value.

## MorphTophatBorder

*Performs top-hat operation on an image.*

### Syntax

```
IppStatus ippMorphTophatBorder_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, IppiBorderType borderType, IppiMorphAdvState* pState);
```

Supported values for `mod`:

8u_C1R	32f_C1R
8u_C3R	32f_C3R
8u_C4R	32f_C4R

### Parameters

<code>pSrc</code>	Pointer to the source image ROI.
<code>srcStep</code>	Distance in bytes between starts of consecutive lines in the source image.
<code>pDst</code>	Pointer to the destination image ROI.
<code>dstStep</code>	Distance in bytes between starts of consecutive lines in the destination image.
<code>roiSize</code>	Size of the source and destination image ROI.
<code>borderType</code>	Type of border; the possible value is <code>ippBorderRepl</code> , which means that a replicated border is used.
<code>pState</code>	Pointer to the morphology state structure.

### Description

The function `ippMorphTophatBorder` is declared in the `ippcv.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function performs a top-hat operation on a rectangular ROI area inside a one-, three-, or four-channel 2D image using a specified in the advanced morphology state structure `pState` mask and the anchor cell. This structure must be initialized by the functions [MorphAdvInitAlloc](#) or [MorphAdvInit](#) beforehand.

The result is equivalent to the opening the source image and following subtraction from the initial source image.



**NOTE.** The function can process only images with ROI that does not exceed the maximum width and height *roiSize* specified by the initialization functions.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with a zero or negative value, or if one of ROI width or height is greater than corresponding size of ROI passed to the initialization functions.
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> or <i>dstStep</i> is less than <i>roiSize.width</i> * <i>&lt;pixelSize&gt;</i> .
<code>ippStsNotEvenStepErr</code>	Indicates an error condition if one of the step values is not divisible by 4 for floating-point images.
<code>ippStsBorderErr</code>	Indicates an error condition if <i>borderType</i> has an illegal value.

## MorphBlackhatBorder

*Performs black-hat operation on an image.*

### Syntax

```
IPPStatus ippIMorphBlackhatBorder_<mod>(const Ipp<datatype>* pSrc, int srcStep,
Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, IppiBorderType borderType,
IppiMorphAdvState* pState);
```

Supported values for mod:

<code>8u_C1R</code>	<code>32f_C1R</code>
<code>8u_C3R</code>	<code>32f_C3R</code>
<code>8u_C4R</code>	<code>32f_C4R</code>

### Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>roiSize</i>	Size of the source and destination image ROI.
<i>borderType</i>	Type of border; the possible value is <code>ippBorderRep1</code> , which means that a replicated border is used.
<i>pState</i>	Pointer to the morphology state structure.



## Description

The function `ippiMorphBlackhatBorder` is declared in the `ippcv.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function performs black-hat operation on a rectangular ROI area inside a one-, three-, or four-channel 2D image using a specified in the advanced morphology state structure `pState` mask and anchor cell. This structure must be initialized by the functions `MorphAdvInitAlloc` or `MorphAdvInit` beforehand.

The result is equivalent to the subtraction of the initial source image from the closed source image.



**NOTE.** The function can process only images with ROI that does not exceed the maximum width and height `roiSize` specified by the initialization functions.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <code>roiSize</code> has a field with a zero or negative value, or if one of ROI width or height is greater than corresponding size of ROI passed to the initialization functions.
<code>ippStsStepErr</code>	Indicates an error condition if <code>srcStep</code> or <code>dstStep</code> is less than <code>roiSize.width * &lt;pixelSize&gt;</code> .
<code>ippStsNotEvenStepErr</code>	Indicates an error condition if one of the step values is not divisible by 4 for floating-point images.
<code>ippStsBorderErr</code>	Indicates an error condition if <code>borderType</code> has an illegal value.

## MorphGradientBorder

*Calculates morphological gradient of an image.*

### Syntax

```
ippStatus ippiMorphGradientBorder_<mod>(const Ipp<datatype>* pSrc, int srcStep,
Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, IppiBorderType borderType,
IppiMorphAdvState* pState);
```

Supported values for `mod`:

<code>8u_C1R</code>	<code>32f_C1R</code>
<code>8u_C3R</code>	<code>32f_C3R</code>
<code>8u_C4R</code>	<code>32f_C4R</code>

### Parameters

<code>pSrc</code>	Pointer to the source image ROI.
<code>srcStep</code>	Distance in bytes between starts of consecutive lines in the source image.

<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>roiSize</i>	Size of the source and destination image ROI.
<i>borderType</i>	Type of border; the possible value is <code>ippBorderRepl</code> , which means that a replicated border is used.
<i>pState</i>	Pointer to the morphology state structure.

## Description

The function `ippiMorphGradientBorder` is declared in the `ippcv.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function calculates a morphological gradient of a rectangular ROI area inside a one-, three-, or four-channel 2D image using a specified in the advanced morphology state structure *pState* mask and anchor cell. This structure must be initialized by the functions [MorphAdvInitAlloc](#) or [MorphAdvInit](#) beforehand.

The result is equivalent to the subtraction of an opened source image from a closed source image.



**NOTE.** The function can process only images with ROI that does not exceed the maximum width and height *roiSize* specified by the initialization functions.

---

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with a zero or negative value, or if one of ROI width or height is greater than corresponding size of ROI passed to the initialization functions.
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> or <i>dstStep</i> is less than <i>roiSize.width</i> * <i>&lt;pixelSize&gt;</i> .
<code>ippStsNotEvenStepErr</code>	Indicates an error condition if one of the step values is not divisible by 4 for floating-point images.
<code>ippStsBorderErr</code>	Indicates an error condition if <i>borderType</i> has an illegal value.

## MorphGrayInitAlloc

*Allocates and initializes morphology state structure for gray-kernel morphology operations.*

---

### Syntax

```
IppStatus ippiMorphGrayInitAlloc_8u_C1R(IppiMorphGrayState_8u** ppState, IppiSize roiSize,
const Ipp32s* pMask, IppiSize maskSize, IppiPoint anchor);
```

```
IppStatus ippiMorphGrayInitAlloc_32f_C1R(IppiMorphGrayState_32f** ppState, IppiSize roiSize,
const Ipp32f* pMask, IppiSize maskSize, IppiPoint anchor);
```

## Parameters

<i>ppState</i>	Double pointer to the gray-kernel morphology state structure.
<i>roiSize</i>	Maximal size of the image ROI in pixels, that can be processed using the allocated structure.
<i>pMask</i>	Pointer to the mask.
<i>maskSize</i>	Size of the mask in pixels.
<i>anchor</i>	Coordinates of the anchor cell.

## Description

The function `ippiMorphGrayInitAlloc` is declared in the `ippcv.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function allocates memory, initializes the gray-kernel morphology state structure and returns the pointer *ppState* to it. It is used by the functions [GrayDilateBorder](#) and [GrayErodeBorder](#) that perform gray-kernel dilation and erosion of the source image pixels corresponding to the specified *pMask* of size *maskSize*. The anchor cell *anchor* is positioned in the arbitrary point in the mask and is used for positioning the mask.



**NOTE.** The structure can be used to process images with ROI that does not exceed the specified maximum width and height *roiSize*.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error when one of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <i>maskSize</i> or if <i>roiSize</i> has a field with a zero or negative value.
<code>ippStsAnchorErr</code>	Indicates an error if <i>anchor</i> is outside the mask.

## MorphGrayFree

*Frees memory allocated for the gray-kernel morphology state structure.*

## Syntax

```
IppStatus ippiMorphGrayFree_8u_C1R(IppiMorphAdvState_8u* pState);
IppStatus ippiMorphGrayFree_32f_C1R(IppiMorphAdvState_32f* pState);
```

## Parameters

<i>pState</i>	Pointer to the gray-kernel morphology state structure.
---------------	--

## Description

The function `ippiMorphGrayFree` is declared in the `ippcv.h` file. This function frees memory allocated by the function [MorphGrayInitAlloc](#) for the gray-kernel morphology state structure *pState*.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <code>pState</code> is NULL.

## MorphGrayInit

*Initializes morphology state structure for gray-kernel morphology operations.*

---

### Syntax

```
IppStatus ippMorphGrayInit_8u_C1R(IppiMorphGrayState_8u* pState, IppiSize roiSize, const Ipp32s* pMask, IppiSize maskSize, IppiPoint anchor);
```

```
IppStatus ippMorphGrayInit_32f_C1R(IppiMorphGrayState_32f* pState, IppiSize roiSize, const Ipp32f* pMask, IppiSize maskSize, IppiPoint anchor);
```

### Parameters

<code>pState</code>	Pointer to the gray-kernel morphology state structure.
<code>roiSize</code>	Maximal size of the image ROI in pixels, that can be processed using the allocated structure.
<code>pMask</code>	Pointer to the mask.
<code>maskSize</code>	Size of the mask in pixels.
<code>anchor</code>	Coordinates of the anchor cell.

### Description

The function `ippMorphGrayInit` is declared in the `ippcv.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function initializes the gray-kernel morphology state structure `pState` in the external buffer. Its size should be computed by the function [MorphGrayGetSize](#). It is used by the functions [GrayDilateBorder](#) and [GrayErodeBorder](#) that perform gray-kernel dilation and erosion of the source image pixels corresponding to the specified `pMask` of size `maskSize`. The anchor cell `anchor` is positioned in the arbitrary point in the mask and is used for positioning the mask.




---

**WARNING.** The structure can be used to process images with ROI that does not exceed the specified maximum width and height `roiSize`.

---

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error when one of the specified pointers is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <code>maskSize</code> or if <code>roiSize</code> has a field with a zero or negative value.
<code>ippStsAnchorErr</code>	Indicates an error if <code>anchor</code> is outside the mask.

## MorphGrayGetSize

*Computes the size of the gray-kernel morphology state structure.*

---

### Syntax

```
IppStatus ippiMorphGrayGetSize_8u_C1R(IppiSize roiSize, const Ipp32s* pMask, IppiSize maskSize, int* pSize);
```

```
IppStatus ippiMorphAdvGetSize_32f_C1R(IppiSize roiSize, const Ipp32f* pMask, IppiSize maskSize, int* pSize);
```

### Parameters

<i>roiSize</i>	Maximal size of the image ROI in pixels, that can be processed using the allocated structure.
<i>pMask</i>	Pointer to the mask.
<i>maskSize</i>	Size of the mask in pixels.
<i>pSize</i>	Pointer to the size of the advanced morphology state structure.

### Description

The function `ippiMorphGrayGetSize` is declared in the `ippcv.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function computes the size of the morphology state structure *pState* for gray-kernel dilation and erosion. This function should be run prior to the function [MorphGrayInit](#).

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error when one of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <i>maskSize</i> or if <i>roiSize</i> has a field with a zero or negative value.

## GrayDilateBorder

*Performs gray-kernel dilation of an image.*

---

### Syntax

```
IppStatus ippiGrayDilateBorder_8u_C1R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst, int dstStep, IppiSize roiSize, IppiBorderType borderType, IppiMorphGrayState_8u* pState);
```

```
IppStatus ippiGrayDilateBorder_32f_C1R(const Ipp32f* pSrc, int srcStep, Ipp32f* pDst, int dstStep, IppiSize roiSize, IppiBorderType borderType, IppiMorphGrayState_32f* pState);
```

### Parameters

<i>pSrc</i>	Pointer to the source image ROI.
-------------	----------------------------------

<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>roiSize</i>	Size of the source and destination image ROI.
<i>borderType</i>	Type of border; the possible value is <code>ippBorderRepl</code> , which means that a replicated border is used.
<i>pState</i>	Pointer to the morphology state structure.

### Description

The function `ippiGrayDilateBorder` is declared in the `ippcv.h`. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function performs gray-kernel dilation of a rectangular ROI area inside a one-channel 2D image using a specified in the gray-kernel morphology state structure *pState* mask and the anchor cell. This structure must be initialized by the functions [MorphGrayInitAlloc](#) or [MorphGrayInit](#) beforehand.




---

**NOTE.** The structure can be used to process images with ROI that does not exceed the specified maximum width and height *roiSize*.

---

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with a zero or negative value, or if one of ROI width or height is greater than corresponding size of ROI passed to the initialization functions.
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> or <i>dstStep</i> is less than $roiSize.width * <pixelSize>$ .
<code>ippStsNotEvenStepErr</code>	Indicates an error condition if one of the step values is not divisible by 4 for floating-point images.
<code>ippStsBorderErr</code>	Indicates an error condition if <i>borderType</i> has an illegal value.

## GrayErodeBorder

*Performs gray-kernel erosion of an image.*

---

### Syntax

```
ippStatus ippiGrayErodeBorder_8u_C1R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst, int dstStep, IppiSize roiSize, IppiBorderType borderType, IppiMorphGrayState_8u* pState);
```

```
ippStatus ippiGrayErodeBorder_32f_C1R(const Ipp32f* pSrc, int srcStep, Ipp32f* pDst, int dstStep, IppiSize roiSize, IppiBorderType borderType, IppiMorphGrayState_32f* pState);
```

## Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>roiSize</i>	Size of the source and destination image ROI.
<i>borderType</i>	Type of border; the possible value is <code>ippBorderRepl</code> , which means that a replicated border is used.
<i>pState</i>	Pointer to the morphology state structure.

## Description

The function `ippiGrayErodeBorder` is declared in the `ippcv.h`. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function performs gray-kernel erosion of a rectangular ROI area inside a one-channel 2D image using a specified in the gray-kernel morphology state structure *pState* mask and the anchor cell. This structure must be initialized by the functions [MorphGrayInitAlloc](#) or [MorphGrayInit](#) beforehand.



**NOTE.** The structure can be used to process images with ROI that does not exceed the specified maximum width and height *roiSize*.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with a zero or negative value, or if one of ROI width or height is greater than corresponding size of ROI passed to the initialization functions.
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> or <i>dstStep</i> is less than $roiSize.width * <pixelSize>$ .
<code>ippStsNotEvenStepErr</code>	Indicates an error condition if one of the step values is not divisible by 4 for floating-point images.
<code>ippStsBorderErr</code>	Indicates an error condition if <i>borderType</i> has an illegal value.

## MorphReconstructGetBufferSize

*Computes the size of the buffer for morphological reconstruction operation.*

---

### Syntax

```
IppStatus ippiMorphReconstructGetBufferSize_<mod>(IppiSize roiSize, int* pBufferSize);
```

Supported values for mod:

8u\_C1IR 16u\_C1IR 32f\_C1IR 64f\_C1IR

### Parameters

<i>roiSize</i>	Maximal size of the image ROI in pixels, that can be processed using the buffer.
<i>pBufferSize</i>	Pointer to the size of the buffer (in bytes).

### Description

The function `ippiMorphReconstructBufferGetSize` is declared in the `ippcv.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function computes the size *pBufferSize* of the buffer for the morphological reconstruction of the source image. This buffer can be used by the functions [MorphReconstructDilate](#) and [MorphReconstructErode](#).

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pBufferSize</i> is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with a zero or negative value.

## MorphReconstructDilate

*Reconstructs an image by dilation.*

---

### Syntax

```
IppStatus ippiMorphReconstructDilate_<mod>(const Ipp<datatype>* pSrc, int srcStep,
Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize, Ipp8u* pBuffer, IppiNorm norm);
```

Supported values for mod:

8u\_C1IR 16u\_C1IR 32f\_C1IR 64f\_C1IR

### Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.



---

<i>pSrcDst</i>	Pointer to the decreased and reconstructed image ROI.				
<i>srcDstStep</i>	Distance in bytes between starts of consecutive lines in the decreased and reconstructed image.				
<i>roiSize</i>	Size of the source and destination image ROI.				
<i>norm</i>	Type of norm to form the mask for dilation; the following values are possible: <table><tr><td><i>ippiNormInf</i></td><td>Infinity norm (8-connectivity, 3x3 rectangular mask).</td></tr><tr><td><i>ippiNormL1</i></td><td>L1 norm (4-connectivity, 3x3 cross mask).</td></tr></table>	<i>ippiNormInf</i>	Infinity norm (8-connectivity, 3x3 rectangular mask).	<i>ippiNormL1</i>	L1 norm (4-connectivity, 3x3 cross mask).
<i>ippiNormInf</i>	Infinity norm (8-connectivity, 3x3 rectangular mask).				
<i>ippiNormL1</i>	L1 norm (4-connectivity, 3x3 cross mask).				
<i>pBuffer</i>	Pointer to the buffer.				

## Description

The function `ippiMorphReconstructDilate` is declared in the `ippcv.h`. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

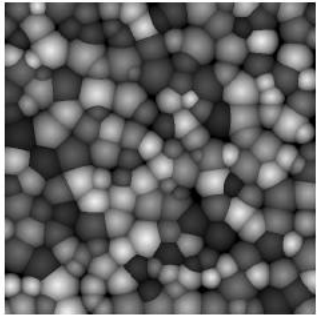
This function performs morphological reconstruction of the decreased source image by dilation [\[Vincent93\]](#). The operation is performed in the working buffer whose size should be computed using the function `MorphReconstructGetBufferSize` beforehand.

This operation enables detection of the regional maximums that can be used as markers for successive watershed segmentation.

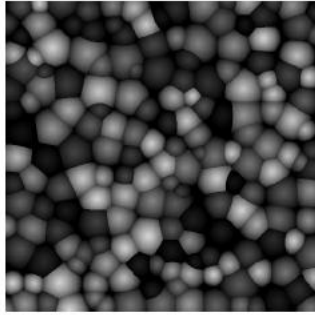
**Example** below shows how the morphological reconstruction can be used to build markers of objects with different brightness. Some value (cap size) is subtracted from the initial image and then the subtracted image is reconstructed to the initial one. Thresholding and opening complete the building of markers. The figure below shows the results of these operations.

### Building Markers for Segmentation by the Morphological Reconstruction

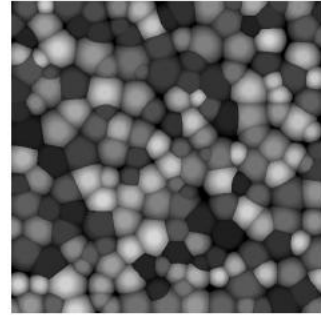
---



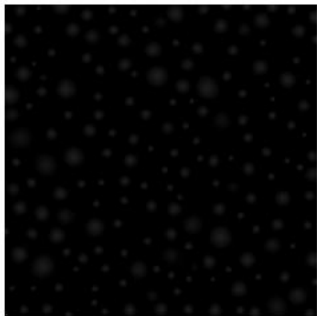
a) Initial Image



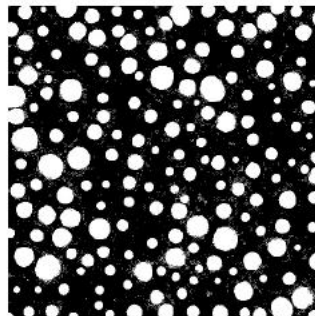
b) Initial Image Decreased by the Cap Value



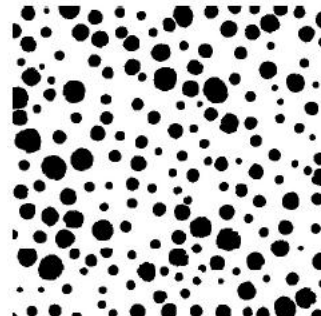
c) Reconstructed Image



d) Cap Image



e) Thresholded Caps



f) Markers

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <code>roiSize</code> has a field with a zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <code>srcStep</code> or <code>srcDstStep</code> is less than <code>roiSize.width * &lt;pixelSize&gt;</code> .
<code>ippStsNotEvenStepErr</code>	Indicates an error condition if one of the step values is not divisible by 4 for floating-point images.
<code>ippStsBadArgErr</code>	Indicates an error condition if <code>norm</code> has an illegal value.

## Example

**Example** below shows how to use the morphological reconstruction to build markers of objects with different brightness.

```

IppiMorphAdvState *state;
IppiSize roi, msize={3,3};
IppiPoint anchor={msize.width/2, msize.height/2};
Ipp8u *src, *dst, *img;
Ipp8u *buf, *mask={1,1,1,1,1,1,1,1};
Int step, size;

...

ippiMorphReconstructGetBufferSize_8u_C1R(roi, &size);
buf = ippsMalloc(size);
ippiMorphAdvInitAlloc_8u_C1R(roi, mask, msize, anchor, &state);

ippiCopy_8u_C1R(src, step, dst, step, roi);
ippiCopy_8u_C1R(src, step, img, step, roi);
// subtract cap size
ippiSubC_8u_C1RSfs(cap, dst, step, roi, 0);
// reconstruct image
ippiMorphReconstructDilate_8u_C1R(src, step, dst, step, roi, buf, norm);
// get caps
ippiSub_8u_C1RSfs(dst, step, img, step, roi, 0);
// caps to white
ippiThreshold_GTVal_8u_C1R(img, step, roi, 0, 255);
// delete noise
ippiMorphOpenBorder_8u_C1R(img, step, dst, step, roi,
                           ipbBorderRepl, state);

// revert to get markers
ippiXorC_8u_C1R(0xff, dst, step, roi, 0);

ippiMorphAdvFree(state);
ippsFree(buf);

```

## MorphReconstructErode

*Reconstructs an image by erosion.*

### Syntax

```

IppStatus ippiMorphReconstructErode_<mod>(const Ipp<datatype>* pSrc, int srcStep,
Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize, Ipp8u* pBuf, IppiNorm norm);

```

Supported values for mod:

8u\_C1R 16u\_C1R 32f\_C1R 64f\_C1R

### Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pSrcDst</i>	Pointer to the decreased and reconstructed image ROI.

<i>srcDstStep</i>	Distance in bytes between starts of consecutive lines in the decreased and reconstructed image.				
<i>roiSize</i>	Size of the source and destination image ROI.				
<i>norm</i>	Type of norm to form the mask for dilation; the following values are possible: <table> <tr> <td><code>ippiNormInf</code></td><td>Infinity norm (8-connectivity, 3x3 rectangular mask).</td></tr> <tr> <td><code>ippiNormL1</code></td><td>L1 norm (4-connectivity, 3x3 cross mask).</td></tr> </table>	<code>ippiNormInf</code>	Infinity norm (8-connectivity, 3x3 rectangular mask).	<code>ippiNormL1</code>	L1 norm (4-connectivity, 3x3 cross mask).
<code>ippiNormInf</code>	Infinity norm (8-connectivity, 3x3 rectangular mask).				
<code>ippiNormL1</code>	L1 norm (4-connectivity, 3x3 cross mask).				
<i>pBuffer</i>	Pointer to the buffer.				

## Description

The function `ippiMorphReconstructErode` is declared in the `ippcv.h`. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function performs morphological reconstruction of the increased source image by erosion [Vincent93]. The operation is performed in the working buffer whose size should be computed using the function `MorphReconstructGetBufferSize` beforehand.

This operation enables detection of the regional minimums that can be used as markers for successive watershed segmentation.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with a zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> or <i>srcDstStep</i> is less than <i>roiSize.width</i> * <i>&lt;pixelSize&gt;</i> .
<code>ippStsNotEvenStepErr</code>	Indicates an error condition if one of the step values is not divisible by 4 for floating-point images.
<code>ippStsBadArgErr</code>	Indicates an error condition if <i>norm</i> has an illegal value.

# Filtering Functions

This chapter describes the Intel® IPP image processing functions that perform linear and non-linear filtering operations on an image.

Filtering can be used in a variety of image processing operations; for example, edge detection, blurring, noise removal, and feature detection.

The table below lists the Intel IPP filtering functions described in more detail later in this chapter:

## Filtering functions

Function Base Name	Operation
<code>FilterBox</code>	Blurs an image using a box filter.
<code>SumWindowRow</code>	Sums pixel values in the row mask applied to the image.
<code>SumWindowColumn</code>	Sums pixel values in the column mask applied to the image.
<code>FilterMin</code>	Filters an image using a <i>min</i> filter.
<code>FilterMax</code>	Filters an image using a <i>max</i> filter.
<code>FilterMinGetBufferSize</code>	Computes the size of the working buffer for the minimum filter.
<code>FilterMaxGetBufferSize</code>	Computes the size of the working buffer for the maximum filter.
<code>FilterMinBorderReplicate</code>	Filters an image using a <i>min</i> filter with border replication.
<code>FilterMaxBorderReplicate</code>	Filters an image using a <i>max</i> filter with border replication.
<code>FilterBilateralGetBufSize</code>	Computes the size of the buffer for the bilateral filter.
<code>FilterBilateralInit</code>	Initializes the bilateral filter structure.
<code>FilterBilateral</code>	Performs bilateral filtering of the image.
<code>DecimateFilterRow</code> , <code>DecimateFilterColumn</code>	Decimates an image using the specified filter.

Function Base Name	Operation
<b>Median Filters</b>	
<code>FilterMedianGetBufferSize</code>	Compute the size of the external buffer.
<code>FilterMedian</code>	Applies a median filter to an image.
<code>FilterMedianHoriz</code>	Filters an image using a median filter with a horizontal mask.
<code>FilterMedianVert</code>	Filters an image using a median filter with a vertical mask.
<code>FilterMedianCross</code>	Filters an image using a cross median filter.
<code>FilterMedianWeightedCenter3x3</code>	Filters an image using a median filter with weighted center pixel.
<code>FilterMedianColor</code>	Applies a color median filter to an image.
<b>General Linear Filters</b>	
<code>Filter</code>	Filters an image using a general rectangular convolution kernel.
<code>FilterGetBufSize</code>	Compute the size of working buffer for 64f general linear filter.
<code>Filter32f</code>	Filters an image with integer data using a floating-point rectangular convolution kernel.
<code>Filter_Round16s, Filter_Round32s, Filter_Round32f</code>	Perform filtering an image and rounding the result.
<code>FilterRoundGetBufSize16s, FilterRoundGetBufSize32s, FilterRoundGetBufSize32f</code>	Computes the size of the working buffer for filters with rounding mode.
<b>Separable Filters</b>	
<code>FilterColumn</code>	Filters an image using an integer column convolution kernel.
<code>FilterColumn32f</code>	Filters an image using a floating-point column convolution kernel.
<code>FilterRow</code>	Filters an image using an integer row convolution kernel.
<code>FilterRow32f</code>	Filters an image using a floating-point row convolution kernel.
<code>FilterRowBorderPipelineGetBufferSize, FilterRowBorderPipelineGetBufferSize_Low</code>	Compute the size of working buffer for rows filter with border.

Function Base Name	Operation
<code>FilterColumnPipelineGetBufferSize, FilterColumnPipelineGetBufferSize_Low</code>	Compute the size of working buffer for columns filter with border.
<code>FilterRowBorderPipeline, FilterRowBorderPipeline_Low</code>	Applies the filter with border to image rows.
<code>FilterColumnPipeline, FilterColumnPipeline_Low</code>	Applies the filter with border to image columns.
<code>DotProdCol</code>	Calculates the dot product of taps vector and columns of the specified set of rows.
<b>Wiener Filters</b>	
<code>FilterWienerGetBufferSize</code>	Computes the size of the external buffer for <code>ippiFilterWiener</code> function.
<code>FilterWiener</code>	Filters an image using the Wiener algorithm.
<b>Convolution</b>	
<code>ConvFull</code>	Performs full convolution of two images.
<code>ConvValid</code>	Performs valid convolution of two images.
<b>Deconvolution</b>	
<code>DeconvFFTInitAlloc</code>	Allocates and initializes state structure for FFT deconvolution.
<code>DeconvFFTFree</code>	Frees memory allocated for the FFT deconvolution state structure.
<code>DeconvFFT</code>	Performs FFT deconvolution of an image.
<code>DeconvLRInitAlloc</code>	Allocates and initializes state structure for LR deconvolution.
<code>DeconvLRFree</code>	Frees memory allocated for the LR deconvolution state structure.
<code>DeconvLR</code>	Performs LR deconvolution of an image.
<b>Fixed Filters</b>	
<code>FilterPrewittHoriz</code>	Filters an image using a horizontal Prewitt operator.
<code>FilterPrewittVert</code>	Filters an image using a vertical Prewitt operator.

Function Base Name	Operation
<code>FilterScharrHoriz</code>	Filters an image using a horizontal Scharr operator.
<code>FilterScharrVert</code>	Filters an image using a vertical Scharr operator.
<code>FilterSobelHoriz</code> , <code>FilterSobelHorizMask</code>	Filters an image using a horizontal Sobel operator.
<code>FilterSobelVert</code> , <code>FilterSobelVertMask</code>	Filters an image using a vertical Sobel operator.
<code>FilterSobelHorizSecond</code>	Filters an image using a second derivative horizontal Sobel operator.
<code>FilterSobelVertSecond</code>	Filters an image using a second derivative vertical Sobel operator.
<code>FilterSobelCross</code>	Filters an image using a second cross derivative Sobel operator.
<code>FilterRobertsDown</code>	Filters an image using a horizontal Roberts cross-gradient operator.
<code>FilterRobertsUp</code>	Filters an image using a vertical Roberts cross-gradient operator.
<code>FilterLaplace</code>	Filters an image using a Laplacian kernel.
<code>FilterGauss</code>	Filters an image using a Gaussian kernel.
<code>FilterHipass</code>	Applies a highpass filter to an image.
<code>FilterLowpass</code>	Applies a lowpass filter to an image.
<code>FilterSharpen</code>	Applies a sharpening filter to an image.
<b>Fixed Filters with Borders</b>	
<code>FilterScharrHorizGetBufferSize</code>	Computes the size of the external buffer for the horizontal Scharr filter with border.
<code>FilterScharrVertGetBufferSize</code>	Computes the size of the external buffer for the vertical Scharr filter with border.
<code>FilterSobelHorizGetBufferSize</code>	Computes the size of the external buffer for the horizontal Sobel filter with border.
<code>FilterSobelVertGetBufferSize</code> , <code>FilterSobelNegVertGet- BufferSize</code>	Computes the size of the external buffer for the vertical Sobel filter with border.



Function Base Name	Operation
<code>FilterSobelHorizSecondGetBufferSize</code>	Computes the size of the external buffer for the second derivative horizontal Sobel filter with border.
<code>FilterSobelVertSecondGetBufferSize</code>	Computes the size of the external buffer for the second derivative vertical Sobel filter with border.
<code>FilterSobelCrossGetBufferSize</code>	Computes the size of the external buffer for the cross Sobel filter with border.
<code>FilterLaplacianGetBufferSize</code>	Computes the size of the external buffer for the Laplace filter with border.
<code>FilterGaussGetBufferSize</code>	Computes the size of the external buffer for the Gaussian filter with border.
<code>FilterLowpassGetBufferSize</code>	Computes the size of the external buffer for the lowpass filter with border.
<code>GenSobelKernel</code>	Computes kernel for the Sobel filter to an image.
<code>FilterScharrHorizBorder</code>	Applies horizontal Scharr filter with border to an image.
<code>FilterScharrVertBorder</code>	Applies vertical Scharr filter with border to an image.
<code>FilterSobelHorizBorder</code>	Applies horizontal Sobel filter with border to an image.
<code>FilterSobelVertBorder, FilterSobelNegVertBorder</code>	Applies vertical Sobel filter with border to an image.
<code>FilterSobelHorizSecondBorder</code>	Applies horizontal (second derivative) Sobel filter with border to an image.
<code>FilterSobelVertSecondBorder</code>	Applies vertical (second derivative) Sobel filter with border to an image.
<code>FilterSobelCrossBorder</code>	Applies second derivative cross Sobel filter with border to an image.
<code>FilterLaplacianBorder</code>	Applies Laplacian filter with border to an image.
<code>FilterGaussBorder</code>	Applies Gauss filter with border.
<code>FilterLowpassBorder</code>	Applies lowpass filter with border to an image.

Most of filtering function operate with ROI (see [Regions of Interest in Intel IPP](#)). The size of the source image ROI is equal to *dstRoiSize*, the destination image ROI size. Most of the functions use different source and destination image buffers, that is, they are not in-place.

## Borders

Filtering functions described later in this chapter perform neighborhood operations (see [Neighborhood Operations](#) in chapter 2). They operate on the assumption that for each pixel to be processed, all neighborhood pixels required for the operation are also available.

The neighborhood for each given pixel is defined by the filter kernel (or mask) size and anchor cell position. Anchor cell (see [Figure "Borders for Neighborhood Operations" \(a\)](#)) is a fixed cell within the kernel, which is used for positioning the kernel with respect to the currently processed pixel of the source image. Specifically, the kernel is placed on the image in such a way that the anchor cell coincides with the input pixel. The anchor cell is specified by its coordinates *anchor.x* and *anchor.y* in the coordinate system associated with the lower right corner of the kernel.

As [Figure "Borders for Neighborhood Operations" \(b\)](#) illustrates, if the input pixel is near the horizontal or vertical edge of the image, the overlaid kernel may refer to neighborhood pixels that do not exist within the source image (that is, are located outside the image area). The set of all boundary source image pixels that require such non-existent pixels to complete the neighborhood operation for the given kernel and anchor is shaded yellow in [Figure "Borders for Neighborhood Operations" \(b\)](#), while the collection of all scanned external pixels (called *border pixels*) is shaded gray.

If you want to apply some filtering operation to the whole source image, you must figure out what additional border pixels are required for the operation, and define in one way or another these non-existent pixels. To do this, you can use the Intel IPP functions [ippiCopyConstBorder](#), [ippiCopyReplicateBorder](#), or [ippiCopyWrapBorder](#), which fill the border of the extended image with the pixel values that you define, or you can apply your own extension method.



**Note.** If the required border pixels are not defined prior to the filtering function call, you may get memory violation errors.

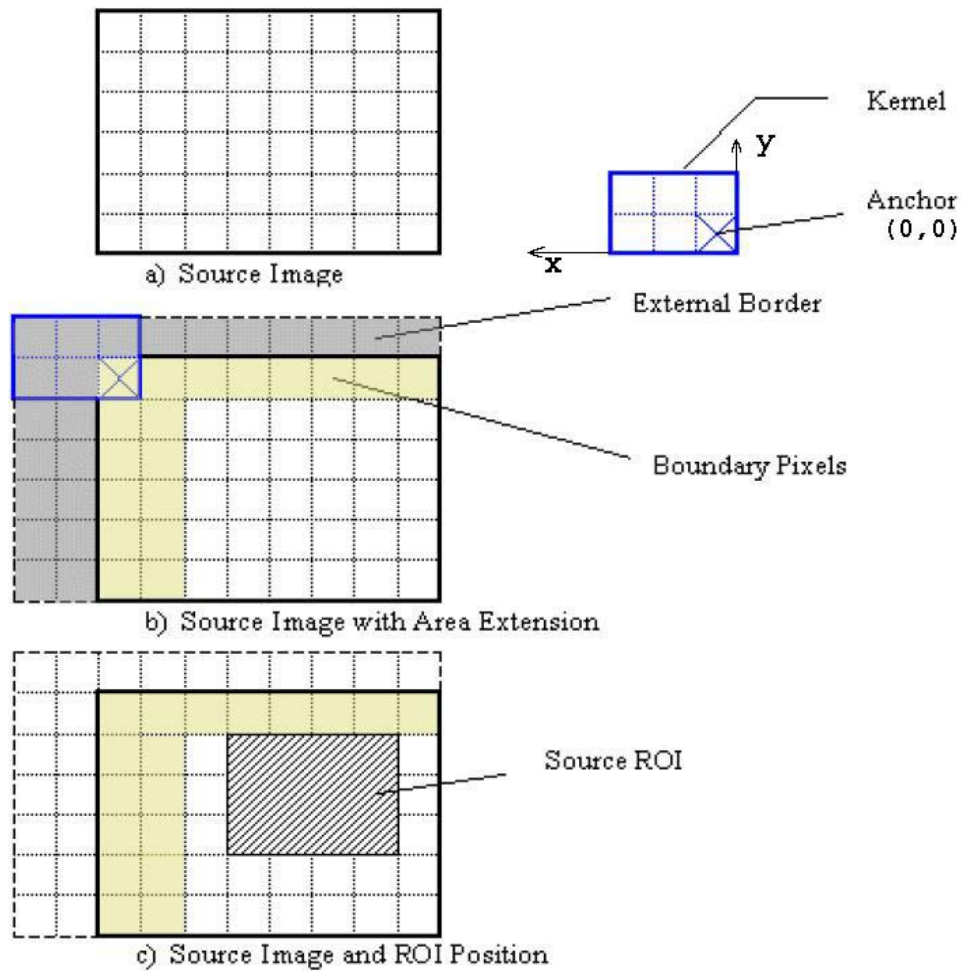
---

If you want to apply the filtering operation to the part of the source image, or ROI (see [Regions of Interest in Intel IPP](#) in chapter 2), then the necessity of extending the image area with border pixels depends upon the ROI size and position within the image. [Figure "Borders for Neighborhood Operations" \(c\)](#) shows that if ROI does not cover yellow (internal boundary) pixels, then no external pixels are scanned, and border extension is not required.

If boundary pixels are part of ROI, then some area extension of the source image is still necessary.

The bottom-line is that to provide valid results of filtering operations, the application must check that ROI parameters passed to the filtering function have such values that all required neighborhood pixels actually exist in the image and define the missing pixels when necessary.

### Borders for Neighborhood Operations



## FilterBox

*Blurs an image using a simple box filter.*

---

### Syntax

#### Case 1: Not-in-place operation

```
IppStatus ippiFilterBox_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pDst,
int dstStep, IppiSize dstRoiSize, IppiSize maskSize, IppiPoint anchor);
```

Supported values for mod:

8u_C1R	16u_C1R	16s_C1R	32f_C1R	64f_C1R
8u_C3R	16u_C3R	16s_C3R	32f_C3R	
8u_C4R	16u_C4R	16s_C4R	32f_C4R	
8u_AC4R	16u_AC4R	16s_AC4R	32f_AC4R	

#### Case 2: In-place operation

```
IppStatus ippiFilterBox_<mod>(Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize,
IppiSize maskSize, IppiPoint anchor);
```

Supported values for mod:

8u_C1IR	16u_C1IR	16s_C1IR	32f_C1IR
8u_C3IR	16u_C3IR	16s_C3IR	32f_C3IR
8u_C4IR	16u_C4IR	16s_C4IR	32f_C4IR
8u_AC4IR	16u_AC4IR	16s_AC4IR	32f_AC4IR

### Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>dstRoiSize</i>	Size of the destination ROI in pixels.
<i>roiSize</i>	Size of the source and destination ROI in pixels for the in-place operation.
<i>pSrcDst</i>	Pointer to the source and destination image ROIs for the in-place operation.
<i>srcDstStep</i>	Distance in bytes between starts of consecutive lines in the source and destination image buffer for the in-place operation.
<i>maskSize</i>	Size of the mask in pixels.
<i>anchor</i>	Anchor cell specifying the mask alignment with respect to the position of the input pixel.

## Description

The function `ippiFilterBox` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function sets each pixel in the destination image as the average of all the input image pixels in the rectangular neighborhood of size `maskSize` with the anchor cell at that pixel. This has the effect of smoothing or blurring the input image. To ensure valid operation when image boundary pixels are processed, the application must correctly define additional border pixels (see [Borders](#)).

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <code>pSrc</code> , <code>pDst</code> , or <code>pSrcDst</code> is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <code>dstRoiSize</code> or <code>roiSize</code> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <code>srcStep</code> , <code>dstStep</code> , or <code>srcDstStep</code> has a zero or negative value.
<code>ippStsMaskSizeErr</code>	Indicates an error condition if <code>maskSize</code> has a field with zero or negative value.
<code>ippStsAnchorErr</code>	Indicates an error condition if <code>anchor</code> is outside the mask size.
<code>ippStsMemAllocErr</code>	Indicates a memory allocation error.

## SumWindowRow

Sums pixel values in the row mask applied to the image.

## Syntax

```
ippiStatus ippiSumWindowRow_<mod>(const Ipp<srcDatatype>* pSrc, int srcStep, Ipp32f* pDst,
int dstStep, IppiSize dstRoiSize, int maskSize, int xAnchor);
```

Supported values for mod:

<code>8u32f_C1R</code>	<code>16u32f_C1R</code>	<code>16s32f_C1R</code>
<code>8u32f_C3R</code>	<code>16u32f_C3R</code>	<code>16s32f_C3R</code>
<code>8u32f_C4R</code>	<code>16u32f_C4R</code>	<code>16s32f_C4R</code>

## Parameters

<code>pSrc</code>	Pointer to the source image ROI.
<code>srcStep</code>	Distance in bytes between starts of consecutive lines in the source image.
<code>pDst</code>	Pointer to the destination image ROI.
<code>dstStep</code>	Distance in bytes between starts of consecutive lines in the destination image.
<code>dstRoiSize</code>	Size of the destination ROI in pixels.
<code>maskSize</code>	Size of the horizontal row mask in pixels.

*xAnchor* Anchor cell specifying the row mask alignment with respect to the position of the input pixel.

## Description

The function `ippiSumWindowRow` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function sets each pixel in the destination image ROI *pDst* as the sum of all the source image pixels in the horizontal row mask of size *maskSize* with the anchor cell *xAnchor* at the corresponding pixel in the source image ROI *pSrc*. To ensure valid operation when image boundary pixels are processed, the application must correctly define additional border pixels (see [Borders](#)).

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pSrc</i> , <i>pDst</i> is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>dstRoiSize</i> has a field with a zero or negative value.
<code>ippStsMaskSizeErr</code>	Indicates an error condition if <i>maskSize</i> has a field with a zero or negative value.
<code>ippStsAnchorErr</code>	Indicates an error condition if <i>xAnchor</i> is outside the mask size.
<code>ippStsMemAllocErr</code>	Indicates a memory allocation error.

## SumWindowColumn

*Sums pixel values in the column mask applied to the image.*

---

## Syntax

```
ippStatus ippiSumWindowColumn_<mod>(const Ipp<srcDatatype>* pSrc, int srcStep, Ipp32f* pDst, int dstStep, IppiSize dstRoiSize, int maskSize, int yAnchor);
```

Supported values for `mod`:

8u32f_C1R	16u32f_C1R	16s32f_C1R
8u32f_C3R	16u32f_C3R	16s32f_C3R
8u32f_C4R	16u32f_C4R	16s32f_C4R

## Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>dstRoiSize</i>	Size of the destination ROI in pixels.

<i>maskSize</i>	Size of the vertical column mask in pixels.
<i>yAnchor</i>	Anchor cell specifying the column mask alignment with respect to the position of the input pixel.

## Description

The function `ippiSumWindowColumn` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function sets each pixel in the destination image ROI *pDst* as the sum of all the source image pixels in the vertical column mask of size *maskSize* with the anchor cell *yAnchor* at the corresponding pixel in the source image ROI *pSrc*. To ensure valid operation when image boundary pixels are processed, the application must correctly define additional border pixels (see [Borders](#)).

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pSrc</i> , <i>pDst</i> is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>dstRoiSize</i> has a field with a zero or negative value.
<code>ippStsMaskSizeErr</code>	Indicates an error condition if <i>maskSize</i> has a field with a zero or negative value.
<code>ippStsAnchorErr</code>	Indicates an error condition if <i>xAnchor</i> is outside the mask size.
<code>ippStsMemAllocErr</code>	Indicates a memory allocation error.

## FilterMin

*Applies the 'min' filter to an image.*

### Syntax

```
ippStatus ippiFilterMin_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pDst,
int dstStep, IppiSize dstRoiSize, IppiSize maskSize, IppiPoint anchor);
```

Supported values for mod:

8u_C1R	16u_C1R	16s_C1R	32f_C1R
8u_C3R	16u_C3R	16s_C3R	32f_C3R
8u_C4R	16u_C4R	16s_C4R	32f_C4R
8u_AC4R	16u_AC4R	16s_AC4R	32f_AC4R

## Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.

<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>dstRoiSize</i>	Size of the source and destination ROI in pixels.
<i>maskSize</i>	Size of the mask in pixels.
<i>anchor</i>	Anchor cell specifying the mask alignment with respect to the position of the input pixel.

## Description

The function `ippiFilterMin` is declared in the `ippi.h` file. It operates with ROI (see Regions [Regions of Interest in Intel IPP](#)).

This function sets each pixel in the destination image to the minimum value of all the source image pixel values in the neighborhood of size *maskSize* with the anchor cell at that pixel. This has the effect of decreasing the contrast in the image.

The anchor cell is specified by its coordinates *anchor.x* and *anchor.y* in the coordinate system associated with the top left corner of the kernel. To ensure valid operation when image boundary pixels are processed, the application should correctly define additional border pixels (see [Borders](#)).

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pSrc</i> or <i>pDst</i> is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>dstRoiSize</i> has a field with a zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> or <i>dstStep</i> has a zero or negative value.
<code>ippStsMaskSizeErr</code>	Indicates an error condition if <i>maskSize</i> has a field with a zero or negative value.
<code>ippStsAnchorErr</code>	Indicates an error condition if <i>anchor</i> is outside the mask size.

## Example

The code example below illustrates the use of the `ippiFilterMin` function. Note that this function operates on the assumption that all neighborhood pixels needed from outside the ROI are available.

```

IppStatus filterMin( void) {
    Ipp8u x[5*4], y[5*4]={0};
    IppiSize img={5,4}, roi={3,2}, mask={3,3};
    IppiPoint anchor = {1,1};
    ippiSet_8u_C1R( 3, x, 5, img );
    /// set a hole (1) at strow 1 and column 1.
    /// The value will be extended on filter
    /// mask area, depending on anchor
    x[5+1] = 1;
    /// ROI is inside the image.
    /// Offset pointers to jump at the ROI start
    return ippiFilterMin_8u_C1R( x+6, 5, y+6, 5, roi, mask, anchor );
}

```



The destination image *y* contains:

```
00 00 00 00 00
00 01 01 03 00
00 01 01 03 00
00 00 00 00 00
```

## FilterMax

*Applies the 'max' filter to an image.*

### Syntax

```
IppStatus ippiFilterMax_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pDst,
int dstStep, IppiSize dstRoiSize, IppiSize maskSize, IppiPoint anchor);
```

Supported values for *mod*:

8u_C1R	16u_C1R	16s_C1R	32f_C1R
8u_C3R	16u_C3R	16s_C3R	32f_C3R
8u_C4R	16u_C4R	16s_C4R	32f_C4R
8u_AC4R	16u_AC4R	16s_AC4R	32f_AC4R

### Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>dstRoiSize</i>	Size of the source and destination ROI in pixels.
<i>maskSize</i>	Size of the mask in pixels.
<i>anchor</i>	Anchor cell specifying the mask alignment with respect to the position of the input pixel.

### Description

The function `ippiFilterMax` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function sets each pixel in the destination image to the maximum value of all the source image pixel values in the neighborhood of size *maskSize* with the anchor cell at that pixel. This has the effect of increasing the contrast in the image.

The anchor cell is specified by its coordinates *anchor.x* and *anchor.y* in the coordinate system associated with the top left corner of the kernel. To ensure valid operation when image boundary pixels are processed, the application should correctly define additional border pixels (see [Borders](#)).

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
--------------------------	--

<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pSrc</i> or <i>pDst</i> is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>dstRoiSize</i> has a field with a zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> or <i>dstStep</i> has a zero or negative value.
<code>ippStsMaskSizeErr</code>	Indicates an error condition if <i>maskSize</i> has a field with a zero or negative value.
<code>ippStsAnchorErr</code>	Indicates an error condition if <i>anchor</i> is outside the mask size.

## FilterMinGetBufferSize

Computes the size of the working buffer for the minimum filter.

---

### Syntax

```
IppStatus ippFilterMinGetBufferSize_<mod>(int roiWidth, IppiSize maskSize, int* pBufferSize);
```

Supported values for *mod*:

<code>8u_C1R</code>	<code>32f_C1R</code>
<code>8u_C3R</code>	<code>32f_C3R</code>
<code>8u_C4R</code>	<code>32f_C4R</code>

### Parameters

<i>roiWidth</i>	Image width (in pixels).
<i>maskSize</i>	Size of the mask in pixels.
<i>pBufferSize</i>	Pointer to the computed size of the buffer.

### Description

The function `ippFilterMinGetBufferSize` is declared in the `ippcv.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function computes the size of the working buffer required for the `ippFilterMinBorderReplicate` function. The buffer with the length `pBufferSize[0]` can be used to filter images with width equal to or less than *roiWidth*.

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if the pointer <i>pBufferSize</i> is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>maskSize</i> has a field with a zero or negative value, or if <i>roiWidth</i> is less than 1.

## FilterMaxGetBufferSize

*Computes the size of the working buffer for the maximum filter.*

---

### Syntax

```
IppStatus ippiFilterMaxGetBufferSize_<mod>(int roiWidth, IppiSize maskSize, int* pBufferSize);
```

Supported values for mod:

8u_C1R	32f_C1R
8u_C3R	32f_C3R
8u_C4R	32f_C4R

### Parameters

<i>roiWidth</i>	Image width (in pixels).
<i>maskSize</i>	Size of the mask in pixels.
<i>pBufferSize</i>	Pointer to the computed size of the buffer.

### Description

The function `ippiFilterMaxGetBufferSize` is declared in the `ippcv.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function computes the size of the working buffer required for the `ippiFilterMaxBorderReplicate` function. The buffer with the length `pBufferSize[0]` can be used to filter images with width equal to or less than *roiWidth*.

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if the pointer <i>pBufferSize</i> is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>maskSize</i> has a field with a zero or negative value, or if <i>roiWidth</i> is less than 1.

## FilterMinBorderReplicate

*Applies the 'min' filter with border replication to an image.*

---

### Syntax

```
IppStatus ippiFilterMinBorderReplicate_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, IppiSize maskSize, IppiPoint anchor, Ipp8u* pBuffer);
```

Supported values for mod:

8u_C1R	32f_C1R
8u_C3R	32f_C3R

8u\_C4R

32f\_C4R

## Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>roiSize</i>	Size of the source and destination ROI in pixels.
<i>maskSize</i>	Size of the mask in pixels.
<i>anchor</i>	Anchor cell specifying the mask alignment with respect to the position of the input pixel.
<i>pBuffer</i>	Pointer to the working buffer.

## Description

The function `ippiFilterMinBorderReplicate` is declared in the `ippcv.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function sets each pixel in the destination image to the minimum value of all the source image pixel values in the neighborhood of size *maskSize* with the anchor cell at that pixel. This has the effect of decreasing the contrast in the image.

The anchor cell is specified by its coordinates *anchor.x* and *anchor.y* in the coordinate system associated with the top left corner of the kernel. Border pixels are chosen according to [Figure "Creating a Replicated Border"](#). The function requires the external buffer *pBuffer*, its size should be computed by the function `ippiFilterMinGetBufferSize` beforehand.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> or <i>maskSize</i> has a field with a zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> or <i>dstStep</i> is less than <i>roiSize.width</i> * <i>&lt;pixelSize&gt;</i> .
<code>ippStsNotEvenStepErr</code>	Indicates an error condition if one of the step values is not divisible by 4 for floating-point images, or by 2 for short-integer images.
<code>ippStsAnchorErr</code>	Indicates an error if <i>anchor</i> is outside the mask.

## FilterMaxBorderReplicate

Applies the 'max' filter with border replication to an image.

### Syntax

```
IppStatus ippiFilterMaxBorderReplicate_<mod>(const Ipp<datatype>* pSrc, int srcStep,
Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, IppiSize maskSize, IppiPoint anchor,
Ipp8u* pBuffer);
```

Supported values for mod:

8u_C1R	32f_C1R
8u_C3R	32f_C3R
8u_C4R	32f_C4R

### Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>roiSize</i>	Size of the source and destination ROI in pixels.
<i>maskSize</i>	Size of the mask in pixels.
<i>anchor</i>	Anchor cell specifying the mask alignment with respect to the position of the input pixel.
<i>pBuffer</i>	Pointer to the working buffer.

### Description

The function `ippiFilterMaxBorderReplicate` is declared in the `ippcv.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function sets each pixel in the destination image to the maximum value of all the source image pixel values in the neighborhood of size *maskSize* with the anchor cell at that pixel. This operation has the effect of increasing the contrast in the image.

The anchor cell is specified by its coordinates *anchor.x* and *anchor.y* in the coordinate system associated with the top left corner of the kernel. Border pixels are chosen according to [Figure "Creating a Replicated Border"](#). The function requires the external buffer *pBuffer*, its size should be computed by the function `ippiFilterMaxGetBufferSize` beforehand.

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointers is <code>NULL</code> .

<code>ippStsSizeErr</code>	Indicates an error condition if <code>roiSize</code> or <code>maskSize</code> has a field with a zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <code>srcStep</code> or <code>dstStep</code> is less than <code>roiSize.width * &lt;pixelSize&gt;</code>
<code>ippStsNotEvenStepErr</code>	Indicates an error condition if one of the step values is not divisible by 4 for floating-point images, or by 2 for short-integer images.
<code>ippStsAnchorErr</code>	Indicates an error if <code>anchor</code> is outside the mask.

## FilterBilateralGetBufSize

Computes the size of the buffer for the bilateral filter.

### Syntax

```
IppStatus ippFilterBilateralGetBufSize_8u_C1R(IppiFilterBilateralType filter, IppiSize maxDstRoiSize, IppiSize maxKernelSize, int* pBufferSize);
```

### Parameters

<code>filter</code>	Type of the bilateral filter; possible value <code>ippFilterBilateralGauss</code> - Gaussian bilateral filter.
<code>maxDstRoiSize</code>	Maximum size of the destination image ROI.
<code>maxKernelSize</code>	Maximum size of the filter kernel.
<code>pBufferSize</code>	Pointer to the computed size of the buffer.

### Description

The function `ippFilterBilateralGetBufSize` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)). This function computes the size of the working buffer for the bilateral filter of the `filter` type that is used by the function `ippFilterBilateralInit`.

The both dimensions of the filter kernel should be odd; if some of them size is even, the function changes its value to the nearest less odd number.

[Example](#) shows how to use the function `ippFilterBilateralGetBufSize`.

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if the pointer <code>pBufferSize</code> is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <code>maxDstRoiSize</code> has a field with a zero or negative value.
<code>ippStsMaskSizeErr</code>	Indicates an error condition if <code>maxKernelSize</code> has a field with a zero or negative value.
<code>ippStsNotSupportedModeErr</code>	Indicates an error condition if the type of the specified bilateral filter is not supported.

## FilterBilateralInit

*Initializes the bilateral filter structure.*

### Syntax

```
IppStatus ippiFilterBilateralInit_8u_C1R(IppiFilterBilateralType filter, IppiSize
maxKernelSize, Ipp32f valSquareSigma, Ipp32f posSquareSigma, int stepInKernel,
IppiFilterBilateralSpec* pSpec);
```

### Parameters

<i>filter</i>	Type of the bilateral filter; possible value <code>ippiFilterBilateralGauss</code> - Gaussian bilateral filter.
<i>maxKernelSize</i>	Maximum size of the filter kernel.
<i>valSquareSigma</i>	Square of the sigma for differences of pixel values.
<i>posSquareSigma</i>	Square of the sigma for pixel positions.
<i>stepInKernel</i>	Processing step in the filter kernel.
<i>pSpec</i>	Pointer to the bilateral filter structure.

### Description

The function `ippiFilterBilateralInit` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)). This function initializes the state structure for the bilateral filter of the *filter* type in the external buffer. Size of the buffer must be computed by the function `ippiFilterBilateralGetBufSize` beforehand.

*maxKernelSize* specifies the maximum sizes of the filter kernel that should be the same as specified in the function `ippiFilterBilateralGetBufSize`.

The anchor cell is always the central cell of the kernel. Therefore, both dimensions of the filter kernel should be odd; if any of the sizes is even, the function changes its value to the nearest less odd number.

Coefficients of the bilateral filter kernel depend on their position in the kernel and on the value of some pixels of the source image laying in the kernel. Only pixels with both coordinates divisible by *stepInKernel* are used in calculations.

The value of the output pixel *d* is

$$d = \frac{\sum_{i,j} w1_{i,j} \cdot w2_{i,j} \cdot v_{i,j}}{\sum_{i,j} w1_{i,j} \cdot w2_{i,j}}$$

Here  $v_{i,j}$  - value of a pixel in the kernel with coordinates *i* and *j* (coordinates of the central pixel are 0, 0),  $i, j = -(kernelSize-1)/2, -((kernelSize-1)/2)+1, \dots, -1, 0, 1, ((kernelSize-1)/2)-1, (kernelSize-1)/2$ .

$w1_{i,j} = \text{Fun}(\text{valSquareSigma}, (v_{i,j} - v_{0,0}))$ ,

$W2_{i,j} = \text{Fun}(\text{posSquareSigma}, \text{MaxByAbs}(i, j))$

where

- difference of pixel values,  $v_{i,j} - v_{0,0}$

$$\text{MaxByAbs}(i, j) = \begin{cases} i, & \text{if } \text{Abs}(i) \geq \text{Abs}(j) \\ j, & \text{if } \text{Abs}(i) < \text{Abs}(j) \end{cases}$$

- pixel position,

$\text{valSquareSigma}$  and  $\text{posSquareSigma}$  - squares of values of sigma for pixel differences and their positions respectively.

The Intel IPP functions implement only the Gaussian filtering, which reduces noise and smooths the image, but preserves edges. For the Gaussian filtering

$$\text{Fun}(S, I) = \exp\left(-\frac{I^2}{2S^2}\right),$$

$S^2$  -  $\text{valSquareSigma}$  or  $\text{posSquareSigma}$ ,  $I$  - difference of pixel values or position.

[Example](#) shows how to use the function `ippiFilterBilateralInit`.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if the pointer <code>pBufferSize</code> is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <code>maxDstRoiSize</code> has a field with a zero or negative value.
<code>ippStsMaskSizeErr</code>	Indicates an error condition if <code>maxKernelSize</code> has a field with a zero or negative value.
<code>ippStsNotSupportedModeErr</code>	Indicates an error condition if the type of the specified bilateral filter is not supported.

## FilterBilateral

*Performs bilateral filtering of the image.*

---

### Syntax

```
ippStatus ippiFilterBilateral_8u_C1R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst, int dstStep, IppiSize dstRoiSize, IppiSize kernelSize, IppiFilterBilateralSpec* pSpec);
```

### Parameters

`pSrc`                                      Pointer to the source image ROI.



<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>dstRoiSize</i>	Size of the source and destination ROI in pixels.
<i>kernelSize</i>	Size of the filter kernel.
<i>pSpec</i>	Pointer to the bilateral filter structure.

## Description

The function `ippiFilterBilateral` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)). This function applies the bilateral filter with the rectangular kernel of the size *kernelSize* to the pixels of ROI of the source image *pSrc*. The anchor cell is always central cell of the kernel. Therefore, both dimensions of the filter kernel should be odd; if some of size is even, the function changes its value to the nearest less odd number.

The bilateral filter structure *pSpec* contains the parameters of filtering and must be initialized by the function `ippiFilterBilateralInit` beforehand. The size of the filter kernel should not be greater than the value *maxKernelSize* specified in the filter structure.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <i>dstRoiSize</i> has a field with a zero or negative value.
<code>ippStsMaskSizeErr</code>	Indicates an error condition if <i>kernelSize</i> has a field with a zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> or <i>dstStep</i> is less than or equal to 0.
<code>ippStsContextMatchErr</code>	Indicates an error condition if a pointer to an invalid context structure is passed.

## Example

The code example below shows how to use the function `ippiFilterBilateral`.

```
#include <stdlib.h>
#include <stdio.h>
#include "ippcore.h"
#include "ipps.h"
#include "ippi.h"

#define D_WIDTH      8
#define D_HEIGHT     8
#define D_MAX_W_KNL  5
#define D_MAX_H_KNL  5

int main( void )
{
    Ipp8u      *src, *src_start;
```

```

Ipp8u      *dst1, *dst2, *dst3;
Ipp8u      *ptmp;
int        width_src, height_src;
int        width_dst, height_dst;
int        step_src;
int        step_dst1, step_dst2, step_dst3;
int        bufsize;
int        stepInKernel;
IppiSize   roi;
IppiSize   kernel;
int        i, j;
Ipp32f     valsigma;
Ipp32f     possigma;
IppiFilterBilateralSpec *pSpec;

width_src = D_WIDTH + D_MAX_W_KNL - 1;
height_src = D_HEIGHT + D_MAX_H_KNL - 1;
src = ippiMalloc_8u_C1( width_src, height_src, &step_src );
for ( i = 0, ptmp = src; i < height_src; i++ ) {
    for ( j = 0; j < width_src; j++ ) {
        ptmp[j] = (Ipp8u)rand();
    }
    ptmp = (Ipp8u *) ( (char *)ptmp + step_src );
}
src_start = (Ipp8u *) ( (char *)src + (D_MAX_H_KNL>>1) * step_src );
src_start += (D_MAX_W_KNL>>1);

width_dst = D_WIDTH;
height_dst = D_HEIGHT;
dst1 = ippiMalloc_8u_C1( width_dst, height_dst, &step_dst1 );
dst2 = ippiMalloc_8u_C1( width_dst, height_dst, &step_dst2 );
dst3 = ippiMalloc_8u_C1( width_dst, height_dst, &step_dst3 );

roi.width = D_WIDTH;
roi.height = D_HEIGHT;
kernel.width = D_MAX_W_KNL;
kernel.height = D_MAX_H_KNL;
ippiFilterBilateralGetBufSize_8u_C1R( ippiFilterBilateralGauss, roi,
                                     kernel, &bufsize);
pSpec = (IppiFilterBilateralSpec *)ippsMalloc_8u( bufsize );

valsigma = 16.f;
possigma = 16.f;
stepInKernel = 1;
ippiFilterBilateralInit_8u_C1R( ippiFilterBilateralGauss, kernel, valsigma,
                               possigma, stepInKernel, pSpec );
ippiFilterBilateral_8u_C1R( src_start, step_src, dst1, step_dst1, roi,
                           kernel, pSpec );

valsigma = 1.f;
possigma = 1.f;
stepInKernel = 1;
ippiFilterBilateralInit_8u_C1R( ippiFilterBilateralGauss, kernel, valsigma,
                               possigma, stepInKernel, pSpec );
ippiFilterBilateral_8u_C1R( src_start, step_src, dst2, step_dst2, roi,
                           kernel, pSpec );

valsigma = 16.f;
possigma = 16.f;

```

```

stepInKernel = 2;
ippiFilterBilateralInit_8u_C1R( ippiFilterBilateralGauss, kernel, valsigma,
                                possigma, stepInKernel, pSpec );
ippiFilterBilateral_8u_C1R( src_start, step_src, dst3, step_dst3, roi,
                            kernel, pSpec );

ippsFree( pSpec );

printf("\n Src =");
for ( i = 0, ptmp = src; i < height_src; i++ ) {
    printf("\n");
    for ( j = 0; j < width_src; j++ ) {
        printf("%3d ",ptmp[j]);
    }
    ptmp = (Ipp8u *) ( (char *)ptmp + step_src );
}

printf("\n Dst1 =");
for ( i = 0, ptmp = dst1; i < height_dst; i++ ) {
    printf("\n");
    for ( j = 0; j < width_dst; j++ ) {
        printf("%3d ",ptmp[j]);
    }
    ptmp = (Ipp8u *) ( (char *)ptmp + step_dst1 );
}
printf("\n Dst2 =");
for ( i = 0, ptmp = dst2; i < height_dst; i++ ) {
    printf("\n");
    for ( j = 0; j < width_dst; j++ ) {
        printf("%3d ",ptmp[j]);
    }
    ptmp = (Ipp8u *) ( (char *)ptmp + step_dst2 );
}
printf("\n Dst3 =");
for ( i = 0, ptmp = dst3; i < height_dst; i++ ) {
    printf("\n");
    for ( j = 0; j < width_dst; j++ ) {
        printf("%3d ",ptmp[j]);
    }
    ptmp = (Ipp8u *) ( (char *)ptmp + step_dst3 );
}

ippiFree(dst3);
ippiFree(dst2);
ippiFree(dst1);
ippsFree(src);
return 0;
}

```

## DecimateFilterRow, DecimateFilterColumn

*Decimates an image by the rows or by the columns.*

### Syntax

```

IppStatus ippiDecimateFilterRow_8u_C1R(const Ipp8u* pSrc, int srcStep, IppiSize srcRoiSize,
Ipp8u* pDst, int dstStep, IppiFraction fraction);

```

```
IppStatus ippDecimateFilterColumn_8u_C1R(const Ipp8u* pSrc, int srcStep, IppiSize
srcRoiSize, Ipp8u* pDst, int dstStep, IppiFraction fraction);
```

## Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>srcRoiSize</i>	Size of the source image ROI in pixels.
<i>pDst</i>	Pointer to the destination image.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>fraction</i>	Specifies how the decimating is performed. Possible values: ippPolyphase_1_2, ippPolyphase_3_5, ippPolyphase_2_3, ippPolyphase_7_10, ippPolyphase_3_4.

## Description

The functions `ippDecimateFilterRow` and `ippDecimateFilterColumn` are declared in the `ippi.h` file. They operate with ROI (see [Regions of Interest in Intel IPP](#)).

These functions perform decimating of the source image by the rows or by the columns respectively. The functions used the set of special internal polyphase filters. The parameter *fraction* specifies how the decimating is performed, for example, if the parameter is set to `ippPolyphase_3_5`, then each 5 pixels in the row (or column) of the source image give 3 pixels in the destination image, if the parameter is set to `ippPolyphase_1_2`, then each two pixels in the row (or column) of the source image give 1 pixel in the destination image, and so on.

To ensure valid operation, the application must correctly define additional border pixels (see [Borders](#)). For all *fraction* values the width of the border is four column/rows all around the source image ROI.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with a zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> or <i>dstStep</i> is less than or equal to 0.
<code>ippStsDecimateFractionErr</code>	Indicates an error condition if <i>fraction</i> has an illegal value.

## Median Filters

The median filter functions perform non-linear filtering of a source image data.

These functions use either an arbitrary rectangular mask, or the following predefined masks of the `IppiMaskSize` type to filter an image:

<code>ippMskSize3x1</code>	Horizontal mask of length 3
<code>ippMskSize5x1</code>	Horizontal mask of length 5
<code>ippMskSize1x3</code>	Vertical mask of length 3
<code>ippMskSize3x3</code>	Square mask of size 3
<code>ippMskSize1x5</code>	Vertical mask of length 5
<code>ippMskSize5x5</code>	Square mask of size 5

The size of the neighborhood and coordinates of the anchor cell in the neighborhood depend on the *mask* mean. Table “Median Filter Mask, Neighborhood, and Anchor Cell” lists the mask types with the corresponding neighborhood sizes and anchor cell coordinates. Note that in mask names the mask size is indicated in (XY) order. The anchor cell is specified by its coordinates *anchor.x* and *anchor.y* in the coordinate system associated with the upper left corner of the mask.

#### Median Filter Mask, Neighborhood, and Anchor Cell

Mask	Neighborhood Size		Anchor Cell
	Columns	Rows	
<code>ippMskSize3x1</code>	3	1	[1, 0]
<code>ippMskSize5x1</code>	5	1	[2, 0]
<code>ippMskSize1x3</code>	1	3	[0, 1]
<code>ippMskSize3x3</code>	3	3	[1, 1]
<code>ippMskSize1x5</code>	1	5	[0, 2]
<code>ippMskSize5x5</code>	5	5	[2, 2]

Median filters have the effect of removing the isolated intensity spikes and can be used to achieve noise reduction in an image.

For details on algorithms used in Intel IPP for median filtering, see [APMF].

## FilterMedianGetBufferSize

*Computes the size of the external buffer for `ippiFilterMedian` function.*

### Syntax

```
ippStatus ippiFilterMedianGetBufferSize_32f(IppiSize dstRoiSize, IppiSize maskSize, Ipp32u nChannels, Ipp32u* pBufferSize);
```

```
ippStatus ippiFilterMedianGetBufferSize_64f(IppiSize dstRoiSize, IppiSize maskSize, Ipp32u nChannels, Ipp32u* pBufferSize);
```

## Parameters

<i>dstRoiSize</i>	Size of the destination ROI in pixels.
<i>maskSize</i>	Size of the mask in pixels.
<i>nChannels</i>	Number of channels in the image.
<i>pBufferSize</i>	Pointer to the computed value of the external buffer size.

## Description

The function `ippiFilterMedianGetBufferSize` is declared in the `ippi.h` file. This function computes the size in bytes of an external memory buffer that is required for the function `ippiFilterMedian`, and stores the result in the *pBufferSize*.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if the <i>pBufferSize</i> pointer is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if one of the fields of <i>dstRoiSize</i> has a zero or negative value.
<code>ippStsMaskSizeErr</code>	Indicates an error condition if one of the fields of <i>maskSize</i> has a value less than or equal to 1.
<code>ippStsNumChannelsErr</code>	Indicates an error condition if <i>nChannels</i> is not 1, 3 or 4.

## FilterMedian

*Filters an image using a median filter.*

---

## Syntax

```
IppStatus ippiFilterMedian_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize dstRoiSize, IppiSize maskSize, IppiPoint anchor);
```

Supported values for `mod`:

8u_C1R	16u_C1R	16s_C1R	32f_C1R
8u_C3R	16u_C3R	16s_C3R	
8u_C4R	16u_C4R	16s_C4R	
8u_AC4R	16u_AC4R	16s_AC4R	

```
IppStatus ippiFilterMedian_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize dstRoiSize, IppiSize maskSize, IppiPoint anchor, Ippi8u pBuffer);
```

Supported values for `mod`:

```
32f_C3R
32f_C4R
64f_C1R
```

## Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>dstRoiSize</i>	Size of the source and destination ROI in pixels.
<i>maskSize</i>	Size of the mask in pixels.
<i>anchor</i>	Anchor cell specifying the mask alignment with respect to the position of the input pixel.
<i>pBuffer</i>	Pointer to the external work buffer.

## Description

The function `ippiFilterMedian` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function sets each pixel in the output buffer as the median value of all the input pixel values taken in the neighborhood of the processed pixel. To ensure valid operation when image boundary pixels are processed, the application should correctly define additional border pixels (see [Borders](#)). The anchor cell is specified by its coordinates *anchor.x* and *anchor.y* in the coordinate system associated with the top left corner of the kernel. The size of the source image ROI is equal to *dstRoiSize*, the size of the destination image ROI.

Some flavors of the function require the external buffer *pBuffer*. Its size must be computed by the function [ippiFilterMedianGetBufferSize](#) beforehand.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pSrc</i> or <i>pDst</i> is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>dstRoiSize</i> has a field with a zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> or <i>dstStep</i> has a zero or negative value.
<code>ippStsMaskSizeErr</code>	Indicates an error condition if <i>maskSize</i> has a field with zero, negative, or even value.
<code>ippStsAnchorErr</code>	Indicates an error condition if <i>anchor</i> is outside the mask size.

## Example

The code example below illustrates median filtering. Note that this filter removes noise and does not cut out signal brightness drops, as an averaging filter does.

```
IppStatus filterMedian(void ) {
    IppiPoint anchor = { 1,1 };
    Ipp8u x[5*4], y[5*4]={0};
    IppiSize img={3,4}, roi={3,2}, mask={3,3};
    ippiSet_8u_C1R( 0x10, x, 5, img );
```

```

    /// raise the level of the signal,
    /// The edge will not be destroyed by the filter
    img.width = 5-3;
    ippiSet_8u_C1R( 0x40, x+3, 5, img );
    /// a spike, will be filtered
    x[5+1] = 0;
    /// roi is inside the image.
    /// Offset pointers to jump at the roi's beginning
    return ippiFilterMedian_8u_C1R( x+6, 5, y+6, 5, roi, mask, anchor );
}

```

The destination image *y* contains:

```

00 00 00 00 00
00 10 10 40 00
00 10 10 40 00
00 00 00 00 00

```

## FilterMedianHoriz

*Filters an image using a horizontal median filter.*

---

### Syntax

```

IppStatus ippiFilterMedianHoriz_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>*
pDst, int dstStep, IppiSize dstRoiSize, IppiMaskSize mask);

```

Supported values for *mod*:

8u_C1R	16u_C1R	16s_C1R
8u_C3R	16u_C3R	16s_C3R
8u_C4R	16u_C4R	16s_C4R
8u_AC4R	16u_AC4R	16s_AC4R

### Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>dstRoiSize</i>	Size of the source and destination ROI in pixels.
<i>mask</i>	Predefined mask of <code>IppiMaskSize</code> type.

### Description

The function `ippiFilterMedianHoriz` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function sets each pixel in the output buffer as the median value of all the input pixel values taken in the neighborhood of the processed pixel. The horizontal size of the neighborhood and the anchor cell coordinates depend on the *mask* mean, which can be either `ippMskSize3x1` or `ippMskSize5x1` (see [Table "Median Filter Mask, Neighborhood, and Anchor Cell"](#)). The function is used on the assumption that the pixels outside of the



source image ROI exist along the distance of half the mask size. It means that the application program should provide appropriate values for the *pSrc* and *dstRoiSize* arguments, or define additional border pixels (see [Borders](#)). The size of the source image ROI is equal to *dstRoiSize*, the size of the destination image ROI.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pSrc</i> or <i>pDst</i> is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>dstRoiSize</i> has a field with a zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> or <i>dstStep</i> has a zero or negative value.
<code>ippStsMaskSizeErr</code>	Indicates an error condition if <i>mask</i> has an illegal value.

## FilterMedianVert

*Filters an image using a vertical median filter.*

### Syntax

```
IppStatus ippiFilterMedianVert_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize dstRoiSize, IppiMaskSize mask);
```

Supported values for *mod*:

8u_C1R	16u_C1R	16s_C1R
8u_C3R	16u_C3R	16s_C3R
8u_C4R	16u_C4R	16s_C4R
8u_AC4R	16u_AC4R	16s_AC4R

### Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>dstRoiSize</i>	Size of the source and destination ROI in pixels.
<i>mask</i>	Predefined mask of <code>IppiMaskSize</code> type.

### Description

The function `ippiFilterMedianVert` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function sets each pixel in the output buffer as the median value of all the input pixel values taken in the neighborhood of the processed pixel. The vertical size of the neighborhood and the anchor cell coordinates depend on the *mask* mean, which can be either `ippMskSize1x3` or `ippMskSize1x5` (see [Table "Median Filter Mask"](#),

[Neighborhood, and Anchor Cell](#)). The function is used on the assumption that the pixels outside of the source image ROI exist along the distance of half the mask size. It means that the application program should provide appropriate values for the *pSrc* and *dstRoiSize* arguments, or define additional border pixels (see [Borders](#)). The size of the source image ROI is equal to *dstRoiSize*, the size of the destination image ROI.

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pSrc</i> or <i>pDst</i> is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>dstRoiSize</i> has a field with a zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> or <i>dstStep</i> has a zero or negative value.
<code>ippStsMaskSizeErr</code>	Indicates an error condition if <i>mask</i> has an illegal value.

## FilterMedianCross

*Filters an image using a cross median filter.*

---

### Syntax

```
ippiFilterMedianCross_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize dstRoiSize, IppiMaskSize mask);
```

Supported values for *mod*:

<code>8u_C1R</code>	<code>16u_C1R</code>	<code>16s_C1R</code>
<code>8u_C3R</code>	<code>16u_C3R</code>	<code>16s_C3R</code>
<code>8u_AC4R</code>	<code>16u_AC4R</code>	<code>16s_AC4R</code>

### Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>dstRoiSize</i>	Size of the source and destination ROI in pixels.
<i>mask</i>	Predefined mask of <code>IppiMaskSize</code> type.

### Description

The function `ippiFilterMedianCross` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function sets each pixel in the output buffer as the median value of all the input pixel values taken in the neighborhood of the processed pixel. The neighborhood is determined by the square mask of the predefined size, which can be either `ippMskSize3x3` or `ippMskSize5x5` (see [Table "Median Filter Mask, Neighborhood,](#)

and Anchor Cell”). The function is used on the assumption that the pixels outside of the source image ROI exist along the distance of half the mask size. It means that the application program should provide appropriate values for the *pSrc* and *dstRoiSize* arguments, or define additional border pixels (see [Borders](#)). The size of the source image ROI is equal to *dstRoiSize*, the size of the destination image ROI.

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pSrc</i> or <i>pDst</i> is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>dstRoiSize</i> has a field with a zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> or <i>dstStep</i> has a zero or negative value.
<code>ippStsMaskSizeErr</code>	Indicates an error condition if <i>mask</i> has an illegal value.

## FilterMedianWeightedCenter3x3

*Filters an image using a median filter with weighted center pixel.*

### Syntax

```
ippStatus ippiFilterMedianWeightedCenter3x3_8u_C1R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst, int dstStep, IppiSize dstRoiSize, int weight);
```

### Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>dstRoiSize</i>	Size of the source and destination ROI in pixels.
<i>weight</i>	Weight of the pixel, must be an odd number.

### Description

The function `ippiFilterMedianWeightedCenter3x3` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function sets each pixel in the destination image as the median value of all the input pixel values taken in the neighborhood of the processed pixel. The neighborhood is determined by the fixed square mask of the size 3x3 with the anchor cell is the center cell of the mask. The parameter *weight* specifies the weight of the processed pixel, that is how many times its value will be included in the calculations. The value of this parameter should be odd. If it is even the function changes its value to the nearest less odd number and returns the warning message.

The function is used on the assumption that the pixels outside of the source image ROI exist along the distance of half the mask size. It means that the application program should provide appropriate values for the *pSrc* and *dstRoiSize* arguments, or define additional border pixels (see [Borders](#)). The size of the source image ROI is equal to *dstRoiSize*, the size of the destination image ROI.

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pSrc</i> or <i>pDst</i> is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>dstRoiSize</i> has a field with a zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> or <i>dstStep</i> has a zero or negative value.
<code>ippStsWeightErr</code>	Indicates an error condition if <i>weight</i> is less than or equal to 0.
<code>ippStsEvenMedianWeight</code>	Indicates a warning if <i>weight</i> has an even value.

## FilterMedianColor

*Filters an image using a color median filter.*

---

### Syntax

```
ippStatus ippiFilterMedianColor_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize dstRoiSize, IppiMaskSize mask);
```

Supported values for mod:

8u_C3R	16s_C3R	32f_C3R
8u_AC4R	16s_AC4R	32f_AC4R

### Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>dstRoiSize</i>	Size of the source and destination ROI in pixels.
<i>mask</i>	Predefined mask of <code>IppiMaskSize</code> type.

### Description

The function `ippiFilterMedianColor` is declared in the `ippi.h` file.

When applied to a color image, the previously described median filtering functions process color planes of an image separately, and as a result any correlation between color components is lost. If you want to preserve this information, use the `ippiFilterMedianColor` function instead. For each input pixel, this function computes

differences between red (R), green (G), and blue (B) color components of pixels in the *mask* neighborhood and the input pixel. The distance between the input pixel *i* and the neighborhood pixel *j* is formed as the sum of absolute values

$$\text{abs}(R(i)-R(j)) + \text{abs}(G(i)-G(j)) + \text{abs}(B(i)-B(j))$$

After scanning the entire neighborhood, the function sets the output value for pixel *i* as the value of the neighborhood pixel with the smallest distance to *i*.

The function `ippiFilterMedianColor` supports square masks of size either `ippMskSize3x3` or `ippMskSize5x5` and processes color images only. To ensure valid operation when image boundary pixels are processed, the application should correctly define additional border pixels (see [Borders](#)).

This function operates with ROI (see [Regions of Interest in Intel IPP](#)).

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pSrc</i> or <i>pDst</i> is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>dstRoiSize</i> has a field with a zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> or <i>dstStep</i> has a zero or negative value.
<code>ippStsMaskSizeErr</code>	Indicates an error condition if <i>mask</i> has an illegal value.

## General Linear Filters

These functions use a general rectangular kernel to filter an image. The kernel is a matrix of signed integers or single-precision real values. For each input pixel, the kernel is placed on the image in such a way that the fixed anchor cell within the kernel coincides with the input pixel. The anchor cell is usually a geometric center of the kernel, but can be skewed with respect to the geometric center.

A pointer to an array of kernel values is passed to filtering functions. These values are read in row-major order starting from the top left corner. There should be exactly *kernelSize.width* \* *kernelSize.height* entries in this array. The anchor cell is specified by its coordinates *anchor.x* and *anchor.y* in the coordinate system associated with the lower right corner of the kernel.

The output value is computed as a sum of neighbor pixels' values, with kernel matrix elements used as weight factors. Note that summation formulas implement a convolution operation, which means that kernel coefficients are used in inverse order. Optionally, the output pixel values may be scaled. To ensure valid operation when image boundary pixels are processed, the application should correctly define additional border pixels (see [Borders](#)).

## Filter

*Filters an image using a general rectangular kernel.*

---

### Syntax

#### Case 1: Operation on integer data

```
IppStatus ippiFilter_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pDst,
int dstStep, IppiSize dstRoiSize, const Ipp32s* pKernel, IppiSize kernelSize, IppiPoint
anchor, int divisor);
```

Supported values for mod:

8u_C1R	16u_C1R	16s_C1R
8u_C3R	16u_C3R	16s_C3R
8u_C4R	16u_C4R	16s_C4R
8u_AC4R	16u_AC4R	16s_AC4R

#### Case 2: Operation on 32f data

```
IppStatus ippiFilter_<mod>(const Ipp32f* pSrc, int srcStep, Ipp32f* pDst, int dstStep,
IppiSize dstRoiSize, const Ipp32f* pKernel, IppiSize kernelSize, IppiPoint anchor);
```

Supported values for mod:

```
32f_C1R
32f_C3R
32f_C4R
32f_AC4R
```

#### Case 3: Operation on 64f data

```
IppStatus ippiFilter_64f_C1R(const Ipp64f* pSrc, int srcStep, Ipp64f* pDst, int dstStep,
IppiSize dstRoiSize, const Ipp64f* pKernel, IppiSize kernelSize, IppiPoint anchor, Ipp8u*
pBuffer);
```

### Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>dstRoiSize</i>	Size of the source and destination ROI in pixels.
<i>pKernel</i>	Pointer to the kernel values.
<i>kernelSize</i>	Size of the rectangular kernel in pixels.
<i>anchor</i>	Anchor cell specifying the rectangular kernel alignment with respect to the position of the input pixel.

<i>divisor</i>	The integer value by which the computed result is divided (for operations on integer data only).
<i>pBuffer</i>	Pointer to the working buffer.

## Description

The function `ippiFilter` is declared in the `ippi.h` file. It operates with ROI (see Regions [Regions of Interest in Intel IPP](#)). This function uses the general rectangular kernel of size *kernelSize* to filter an image ROI. This function sums the products between the kernel coefficients *pKernel* and pixel values taken over the source pixel neighborhood defined by *kernelSize* and *anchor*. The anchor cell is specified by its coordinates *anchor.x* and *anchor.y* in the coordinate system associated with the bottom right corner of the kernel.

Note that kernel coefficients are used in inverse order. The sum is written to the destination pixel. In case of integer data, the result is divided by the fixed scaling factor *divisor*.

To ensure valid operation when image boundary pixels are processed, the application should correctly define additional border pixels (see [Borders](#)).

For function flavors that operate on integer data, the result value  $Y_{i,j}$  for pixel  $X_{i,j}$  inside image's ROI is computed according to the following formula:

$$Y_{i,j} = \frac{1}{divisor} \sum_{m=0}^{H-1} \sum_{n=0}^{W-1} X_{i+n-Q, j+m-P} \times K_{W-n-1, H-m-1}$$

where

$K_{n,m}$  are the kernel values

$H = kernelSize.height$  is the vertical size of the kernel

$W = kernelSize.width$  is the horizontal size of the kernel

$P = H - anchor.y - 1$

$Q = W - anchor.x - 1$

Function flavors that accept floating-point data of `Ipp32f` and `Ipp64f` type use the same summation formula, but no scaling of the result is done.

Function flavor that operates on data of the `Ipp64f` type requires the temporary working buffer. Its size must be computed beforehand using the function `ippiFilterGetBufSize`.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pSrc</i> , <i>pDst</i> , or <i>pKernel</i> is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>dstRoiSize</i> or <i>kernelSize</i> has a field with a zero or negative value.
<code>ippStsDivisorErr</code>	Indicates an error condition if the <i>divisor</i> value is zero.

`ippStsStepErr` Indicates an error condition if the `srcStep` is less than  $(roiSize.width + kernelSize.width) * sizeof(Ipp64f)$ ; or `dstStep` is less than  $roiSize.width * sizeof(Ipp64f)$ .

## FilterGetBufSize

*Computes the size of the working buffer.*

---

### Syntax

```
IppStatus ippFilterGetBufSize_64f_C1R(IppiSize kernelSize, int roiWidth, int* pBufferSize);
```

### Parameters

<code>kernelSize</code>	Size of the rectangular kernel in pixels.
<code>roiWidth</code>	Width of the image ROI in pixels.
<code>pBufferSize</code>	Pointer to the size of the working buffer.

### Description

The function `ippFilterGetBufSize` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function computes the size of the working buffer `pBufferSize` that is required for the function `ippFilter` (flavor which operates on data of the `Ipp64f` type).

In some cases the function may return zero size of the buffer.

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <code>pBufferSize</code> is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <code>kernelSize</code> has a field with a zero or negative value, or <code>roiWidth</code> is less than or equal to 0.

## Filter32f

*Filters an image with integer data using a floating-point rectangular kernel.*

---

### Syntax

```
IppStatus ippFilter32f_<mod>(const Ipp<srcDatatype>* pSrc, int srcStep, Ipp<dstDatatype>* pDst, int dstStep, IppiSize dstRoiSize, const Ipp32f* pKernel, IppiSize kernelSize, IppiPoint anchor);
```

Supported values for `mod`:

8u_C1R	8s_C1R	16u_C1R	16s_C1R	32s_C1R
8u_C3R	8s_C3R	16u_C3R	16s_C3R	32s_C3R
8u_C4R	8s_C4R	16u_C4R	16s_C4R	32s_C4R



8u_AC4R	16u_AC4R 16s_AC4R
8u16s_C1R	8s16s_C1R
8u16s_C3R	8s16s_C3R
8u16s_C4R	8s16s_C4R

## Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>dstRoiSize</i>	Size of the source and destination ROI in pixels.
<i>pKernel</i>	Pointer to the kernel values.
<i>kernelSize</i>	Size of the rectangular kernel in pixels.
<i>anchor</i>	Anchor cell specifying the rectangular kernel alignment with respect to the position of the input pixel.

## Description

The function `ippiFilter32f` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)). This function uses the rectangular kernel of floating-point values to filter an image that consists of integer data. This function sums the products between the kernel coefficients *pKernel* and pixel values taken over the source pixel neighborhood defined by *kernelSize* and *anchor*. The anchor cell is specified by its coordinates *anchor.x* and *anchor.y* in the coordinate system associated with the bottom right corner of the kernel. Note that kernel coefficients are used in inverse order. The sum is written to the destination pixel. The summation formula for computing result values is similar to that used by `ippiFilter` function, but no scaling of the result is done. To ensure valid operation when image boundary pixels are processed, the application must correctly define additional border pixels (see [Borders](#)).

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pSrc</i> , <i>pDst</i> , or <i>pKernel</i> is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>dstRoiSize</i> or <i>kernelSize</i> has a field with a zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if one the step value is less than <i>roiSize.width*sizeof(pSrc)*numberOfChannels</i> .

## Filter\_Round

*Perform filtering an image and rounding the result.*

---

### Syntax

#### Case 1: Filters with integer 16s kernel

```
IppStatus ippiFilter_Round16s_<mod>(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst, int
dstStep, IppiSize dstRoiSize, const Ipp16s* pKernel, IppiSize kernelSize, IppiPoint anchor,
int divisor, IppRoundMode roundMode, Ipp8u* pBuffer);
```

Supported values for mod:

8u\_C1R  
8u\_C3R  
8u\_C4R  
8u\_AC4R

#### Case 2: Filters with integer 32s kernel

```
IppStatus ippiFilter_Round32s_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>*
pDst, int dstStep, IppiSize dstRoiSize, const Ipp32s* pKernel, IppiSize kernelSize,
IppiPoint anchor, int divisor, IppRoundMode roundMode, Ipp8u* pBuffer);
```

Supported values for mod:

16u_C1R	16s_C1R
16u_C3R	16s_C3R
16u_C4R	16s_C4R
16u_AC4R	16s_AC4R

#### Case 3: Filters with floating-point 32f kernel

```
IppStatus ippiFilter_Round32f_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>*
pDst, int dstStep, IppiSize dstRoiSize, const Ipp32f* pKernel, IppiSize kernelSize,
IppiPoint anchor, IppRoundMode roundMode, Ipp8u* pBuffer);
```

Supported values for mod:

8u_C1R	16u_C1R	16s_C1R
8u_C3R	16u_C3R	16s_C3R
8u_C4R	16u_C4R	16s_C4R
8u_AC4R	16u_AC4R	16s_AC4R

### Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.

<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.						
<i>dstRoiSize</i>	Size of the source and destination ROI in pixels.						
<i>pKernel</i>	Pointer to the kernel values.						
<i>kernelSize</i>	Size of the rectangular kernel in pixels.						
<i>anchor</i>	Anchor cell specifying the rectangular kernel alignment with respect to the position of the input pixel.						
<i>divisor</i>	The integer value by which the computed result is divided (for operations on integer data only).						
<i>roundMode</i>	Rounding mode, the following values are possible: <table> <tr> <td><code>ippRndZero</code></td><td>specifies that floating-point values are truncated toward zero,</td></tr> <tr> <td><code>ippRndNear</code></td><td>specifies that floating-point values are rounded to the nearest even integer when the fractional part equals 0.5; otherwise they are rounded to the nearest integer,</td></tr> <tr> <td><code>ippRndFinancial</code></td><td>specifies that floating-point values are rounded down to the nearest integer when the fractional part is less than 0.5, or rounded up to the nearest integer if the fractional part is equal or greater than 0.5.</td></tr> </table>	<code>ippRndZero</code>	specifies that floating-point values are truncated toward zero,	<code>ippRndNear</code>	specifies that floating-point values are rounded to the nearest even integer when the fractional part equals 0.5; otherwise they are rounded to the nearest integer,	<code>ippRndFinancial</code>	specifies that floating-point values are rounded down to the nearest integer when the fractional part is less than 0.5, or rounded up to the nearest integer if the fractional part is equal or greater than 0.5.
<code>ippRndZero</code>	specifies that floating-point values are truncated toward zero,						
<code>ippRndNear</code>	specifies that floating-point values are rounded to the nearest even integer when the fractional part equals 0.5; otherwise they are rounded to the nearest integer,						
<code>ippRndFinancial</code>	specifies that floating-point values are rounded down to the nearest integer when the fractional part is less than 0.5, or rounded up to the nearest integer if the fractional part is equal or greater than 0.5.						
<i>pBuffer</i>	Pointer to the working buffer.						

## Description

The function `ippiFilter_Round` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)). This function uses the general rectangular kernel of size *kernelSize* to filter an image ROI. This function sums the products between the kernel coefficients *pKernel* and pixel values taken over the source pixel neighborhood defined by *kernelSize* and *anchor*. The anchor cell is specified by its coordinates *anchor.x* and *anchor.y* in the coordinate system associated with the bottom right corner of the kernel. Note that kernel coefficients are used in inverse order. The sum is written to the destination pixel.

In case of integer kernel, the result is divided by the fixed scaling factor *divisor* (see [formula](#) above) and then is rounded using the rounding method specified by the parameter *roundMode*.

In case of floating-point kernel the result is rounded using the rounding method specified by the parameter *roundMode*.

To ensure valid operation when image boundary pixels are processed, the application should correctly define additional border pixels (see [Borders](#)).

This function requires the temporary working buffer. Its size must be computed beforehand using the one of the functions `ippiFilterRoundGetBufSize16s`, `ippiFilterRoundGetBufSize32s`, `ippiFilterRoundGetBufSize32f`.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
--------------------------	---

<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <code>dstRoiSize</code> or <code>kernelSize</code> has a field with a zero or negative value.
<code>ippStsDivisorErr</code>	Indicates an error condition if the <code>divisor</code> value is zero, the execution is interrupted.
<code>ippStsStepErr</code>	Indicates an error condition if the <code>srcStep</code> is less than $(roiSize.width + kernelSize.width - 1) * sizeof(pSrc)$ ; or <code>dstStep</code> is less than $roiSize.width * sizeof(pSrc)$ .
<code>ippStsRoundModeNotSupportedErr</code>	Indicates an error condition if the <code>roundMode</code> has an illegal value.

## FilterRoundGetBufSize

*Computes the size of the working buffer for filters with rounding mode.*

---

### Syntax

#### Case 1: Filters with integer 16s kernel

```
IppStatus, ippFilterRoundGetBufSize16s_<mod>(IppiSize kernelSize, int roiWidth, int* pBufferSize);
```

Supported values for `mod`:

```
8u_C1R
8u_C3R
8u_C4R
8u_AC4R
```

#### Case 2: Filters with integer 32s kernel

```
IppStatus, ippFilterRoundGetBufSize32s_<mod>(IppiSize kernelSize, int roiWidth, int* pBufferSize);
```

Supported values for `mod`:

```
16u_C1R    16s_C1R
16u_C3R    16s_C3R
16u_C4R    16s_C4R
16u_AC4R   16s_AC4R
```

#### Case 3: Filters with floating-point 32f kernel

```
IppStatus, ippFilterRoundGetBufSize32f_<mod>(IppiSize kernelSize, int roiWidth, int* pBufferSize);
```

Supported values for `mod`:

```
8u_C1R    16u_C1R    16s_C1R
8u_C3R    16u_C3R    16s_C3R
8u_C4R    16u_C4R    16s_C4R
```

8u\_AC4R 16u\_AC4R 16s\_AC4R

## Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>dstRoiSize</i>	Size of the source and destination ROI in pixels.
<i>pKernel</i>	Pointer to the kernel values.
<i>kernelSize</i>	Size of the rectangular kernel in pixels.
<i>anchor</i>	Anchor cell specifying the rectangular kernel alignment with respect to the position of the input pixel.
<i>divisor</i>	The integer value by which the computed result is divided (for operations on integer data only).
<i>roundMode</i>	Rounding mode, the following values are possible:

## Description

The functions `ippiFilterRoundGetBufSize16s`, `ippiFilterRoundGetBufSize32s`, `ippiFilterRoundGetBufSize32f` are declared in the `ippi.h` file. They operate with ROI (see [Regions of Interest in Intel IPP](#)).

These functions compute the size of the working buffer *pBufferSize* that is required for the function `ippiFilter_Round16s`, `ippiFilter_Round32s`, `ippiFilter_Round32f` respectively.

In some cases the functions may return zero size of buffer.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pBufferSize</i> is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>kernelSize</i> has a field with a zero or negative value, or <i>roiWidth</i> is less than or equal to 0.

## Separable Filters

Separable filters use a spatial kernel consisting of a single column (as in the `FilterColumn` function) or a single row (as in the `FilterRow` function) to filter the source image.

## FilterColumn

*Filters an image using a spatial kernel that consists of a single column.*

---

### Syntax

#### Case 1: Operation on integer data

```
IppStatus ippiFilterColumn_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize dstRoiSize, const Ipp32s* pKernel, int kernelSize, int yAnchor, int divisor);
```

Supported values for mod:

8u_C1R	16u_C1R	16s_C1R
8u_C3R	16u_C3R	16s_C3R
8u_C4R	16u_C4R	16s_C4R
8u_AC4R	16u_AC4R	16s_AC4R

#### Case 2: Operation on floating-point data

```
IppStatus ippiFilterColumn_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize dstRoiSize, const Ipp32f* pKernel, int kernelSize, int yAnchor);
```

Supported values for mod:

32f_C1R	64f_C1R
32f_C3R	
32f_C4R	
32f_AC4R	

### Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>dstRoiSize</i>	Size of the source and destination ROI in pixels.
<i>pKernel</i>	Pointer to the kernel values.
<i>kernelSize</i>	Size of the kernel in pixels.
<i>yAnchor</i>	Anchor cell specifying the kernel vertical alignment with respect to the position of the input pixel.
<i>divisor</i>	The integer value by which the computed result is divided (for operations on integer data only).

## Description

The function `ippiFilterColumn` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function uses the vertical column kernel of size `kernelSize` to filter an image ROI. This function sums the products between the kernel coefficients `pKernel` and pixel values taken over the source pixel neighborhood defined by `kernelSize` and `yAnchor`. Note that kernel coefficients are used in inverse order. The sum is written to the destination pixel. In case of integer data, the result is divided by the fixed scaling factor `divisor`. To ensure valid operation when image boundary pixels are processed, the application must correctly define additional border pixels (see [Borders](#)).

For function flavors that operate on integer data, the result value  $Y_{i,j}$  for pixel  $X_{i,j}$  inside image's ROI is computed according to the following formula:

$$Y_{i,j} = \frac{1}{divisor} \sum_{m=0}^{H-1} X_{i,j+m-P} \times K_{H-m-1}$$

where:

$K_m$  are the kernel values;

$H = kernelSize$  is the size of the vertical column kernel;

$P = H - yAnchor - 1$ .

Function flavors that accept input data of the `Ipp32f` type use the same summation formula, but no scaling of the result is done.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <code>dstRoiSize</code> has a field with a zero or negative value.
<code>ippStsDivisorErr</code>	Indicates an error condition if the <code>divisor</code> value is zero.

## Example

The code example below show how to use the function `ippiFilterColumn_8u_C1R`.

```

Ipp8u src[4*5] = {
    1, 2, 3, 4,
    1, 2, 3, 4,
    1, 2, 3, 4,
    1, 2, 3, 4,
    1, 2, 3, 4
};
Ipp8u dst[4*3];
IppiSize srcRoi = { 4, 3 };
Ipp32s kern[] = { 1, -2, 1 }; // using "Sobel" kernel
int kernelSize = 3;

```

```
int yAnchor = 2;
int divisor = 2;
```

```
ippiFilterColumn_8u_C1R ( src, 4, dst, 4, srcRoi, kern, kernelSize, yAnchor, divisor);
```

**Result:**

```
1 2 3 4
1 2 3 4      src
1 2 3 4

1 -2 1      kern

0  0  0  0
0  0  0  0      dst
0  0  0  0
```

## FilterColumn32f

*Filters an image with integer data using a floating-point column kernel.*

---

### Syntax

```
IppStatus ippiFilterColumn32f_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>*
pDst, int dstStep, IppiSize dstRoiSize, const Ipp32f* pKernel, int kernelSize, int yAnchor);
```

Supported values for mod:

8u_C1R	16u_C1R	16s_C1R
8u_C3R	16u_C3R	16s_C3R
8u_C4R	16u_C4R	16s_C4R
8u_AC4R	16u_AC4R	16s_AC4R

### Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>dstRoiSize</i>	Size of the source and destination ROI in pixels.
<i>pKernel</i>	Pointer to the column kernel values.
<i>kernelSize</i>	Size of the kernel in pixels.
<i>yAnchor</i>	Anchor cell specifying the kernel vertical alignment with respect to the position of the input pixel.

### Description

The function `ippiFilterColumn32f` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).



This function uses the vertical column kernel of floating-point values to filter an image that consists of integer data. This function sums the products between the kernel coefficients *pKernel* and pixel values taken over the source pixel neighborhood defined by *kernelSize* and *yAnchor*. Note that kernel coefficients are used in inverse order. The sum is written to the destination pixel. The summation formula for computing result values is similar to that used by the `ippiFilterColumn` function, but no scaling of the result is done. To ensure valid operation when image boundary pixels are processed, the application should correctly define additional border pixels (see [Borders](#)).

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <i>dstRoiSize</i> has a field with a zero or negative value.

## FilterRow

*Filters an image using a spatial kernel that consists of a single row.*

## Syntax

### Case 1: Operation on integer data

```
ippStatus ippiFilterRow_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pDst,
int dstStep, IppiSize dstRoiSize, const Ipp32s* pKernel, int kernelSize, int xAnchor, int
divisor);
```

Supported values for `mod`:

8u_C1R	16u_C1R	16s_C1R
8u_C3R	16u_C3R	16s_C3R
8u_C4R	16u_C4R	16s_C4R
8u_AC4R	16u_AC4R	16s_AC4R

### Case 2: Operation on floating-point data

```
ippStatus ippiFilterRow_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pDst,
int dstStep, IppiSize dstRoiSize, const Ipp32f* pKernel, int kernelSize, int xAnchor);
```

Supported values for `mod`:

32f_C1R	64f_C1R
32f_C3R	
32f_C4R	
32f_AC4R	

## Parameters

*pSrc*                      Pointer to the source image ROI.

<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>dstRoiSize</i>	Size of the source and destination ROI in pixels.
<i>pKernel</i>	Pointer to the kernel values.
<i>kernelSize</i>	Size of the kernel in pixels.
<i>xAnchor</i>	Anchor cell specifying the kernel horizontal alignment with respect to the position of the input pixel.
<i>divisor</i>	The integer value by which the computed result is divided (for operations on integer data only).

## Description

The function `ippiFilterRow` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function uses the horizontal row kernel of size *kernelSize* to filter an image. This function sums the products between the kernel coefficients *pKernel* and pixel values taken over the source pixel neighborhood defined by *kernelSize* and *xAnchor*. Note that kernel coefficients are used in inverse order.

The sum is written to the destination pixel. In case of integer data, the result is divided by the fixed scaling factor *divisor*. To ensure valid operation when image boundary pixels are processed, the application should correctly define additional border pixels (see [Borders](#)).

For function flavors that operate on integer data, the result value for pixel  $x_{i,j}$  inside image's ROI is computed according to the following formula:

$$Y_{i,j} = \frac{1}{divisor} \sum_{n=0}^{W-1} X_{i+n-Q,j} \times K_{W-n-1}$$

where

$K_n$  are the kernel values;

$W = kernelSize$  is the size of the horizontal row kernel;

$Q = W - xAnchor - 1$ .

Function flavors that accept input data of the `Ipp32f` type use the same summation formula, but no scaling of the result is done.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pSrc</i> , <i>pDst</i> , or <i>pKernel</i> is NULL.

<code>ippStsSizeErr</code>	Indicates an error condition if <code>dstRoiSize</code> has a field with a zero or negative value.
<code>ippStsDivisorErr</code>	Indicates an error condition if the <code>divisor</code> value is zero.

## FilterRow32f

*Filters an image with integer data using a floating-point strow kernel.*

### Syntax

```
ippStatus ippFilterRow32f_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize dstRoiSize, const Ipp32f* pKernel, int kernelSize, int xAnchor);
```

Supported values for `mod`:

<code>8u_C1R</code>	<code>16u_C1R</code>	<code>16s_C1R</code>
<code>8u_C3R</code>	<code>16u_C3R</code>	<code>16s_C3R</code>
<code>8u_C4R</code>	<code>16u_C4R</code>	<code>16s_C4R</code>
<code>8u_AC4R</code>	<code>16u_AC4R</code>	<code>16s_AC4R</code>

### Parameters

<code>pSrc</code>	Pointer to the source image ROI.
<code>srcStep</code>	Distance in bytes between starts of consecutive lines in the source image.
<code>pDst</code>	Pointer to the destination image ROI.
<code>dstStep</code>	Distance in bytes between starts of consecutive lines in the destination image.
<code>dstRoiSize</code>	Size of the source and destination ROI in pixels.
<code>pKernel</code>	Pointer to the strow kernel values.
<code>kernelSize</code>	Size of the kernel in pixels.
<code>xAnchor</code>	Anchor cell specifying the kernel horizontal alignment with respect to the position of the input pixel.

### Description

The function `ippFilterRow32f` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function uses the horizontal strow kernel of floating-point values to filter an image that consists of integer data. This function sums the products between the kernel coefficients `pKernel` and pixel values taken over the source pixel neighborhood defined by `kernelSize` and `xAnchor`. Note that kernel coefficients are used in inverse order. The sum is written to the destination pixel. The summation formula for computing result values is similar to that used by `ippFilterRow` function, but no scaling of the result is done. To ensure valid operation when image boundary pixels are processed, the application should correctly define additional border pixels (see [Borders](#)).

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <code>dstRoiSize</code> has a field with a zero or negative value.

## FilterRowBorderPipelineGetBufferSize, FilterRowBorderPipelineGetBufferSize\_Low

Compute the size of working buffer for the strow filter.

### Syntax

```
IppStatus ippFilterRowBorderPipelineGetBufferSize_<mod>(IppiSize roiSize, int kernelSize,
int* pBufferSize);
```

Supported values for mod:

<code>8u16s_C1R</code>	<code>16s_C1R</code>	<code>32f_C1R</code>
<code>8u16s_C3R</code>	<code>16s_C3R</code>	<code>32f_C3R</code>

```
IppStatus ippFilterRowBorderPipelineGetBufferSize_Low_<mod>(IppiSize roiSize, int
kernelSize, int* pBufferSize);
```

Supported values for mod:

<code>16s_C1R</code>
<code>16s_C3R</code>

### Parameters

<code>roiSize</code>	Maximum size of the source and destination image ROI.
<code>kernelSize</code>	Size of the kernel in pixels.
<code>pBufferSize</code>	Pointer to the computed size of the buffer.

### Description

The functions `ippFilterRowBorderPipelineGetBufferSize` and `ippFilterRowBorderPipelineGetBufferSize_Low` are declared in the `ippcv.h` file. They operate with ROI (see [Regions of Interest in Intel IPP](#)).

These functions compute the size of the working buffer required for the functions `ippFilterRowBorderPipeline` and `ippFilterRowBorderPipeline_Low` respectively. The buffer with the length `pBufferSize[0]` can be used to filter images with width equal to or less than `roiSize`.

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if the pointer <code>pBufferSize</code> is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <code>maskSize</code> has a field with a zero or negative value, or if <code>roiWidth</code> is less than 1.

## FilterColumnPipelineGetBufferSize, FilterColumnPipelineGetBufferSize\_Low

Compute the size of working buffer for the column filter.

### Syntax

```
IppStatus ippiFilterColumnPipelineGetBufferSize_<mod>(IppiSize roiSize, int kernelSize,
int* pBufferSize);
```

Supported values for mod:

16s_C1R	16s8u_C1R	16s8s_C1R	32f_C1R
16s_C3R	16s8u_C3R	16s8s_C3R	32f_C3R

```
IppStatus ippiFilterColumnPipelineGetBufferSize_Low_<mod>(IppiSize roiSize, int kernelSize,
int* pBufferSize);
```

Supported values for mod:

16s_C1R
16s_C3R

### Parameters

<i>roiSize</i>	Maximum size of the source and destination image ROI.
<i>kernelSize</i>	Size of the kernel in pixels.
<i>pBufferSize</i>	Pointer to the computed size of the buffer.

### Description

The functions `ippiFilterColumnPipelineGetBufferSize` and `ippiFilterColumnPipelineGetBufferSize_Low` are declared in the `ippcv.h` file. They operate with ROI (see [Regions of Interest in Intel IPP](#)).

These functions compute the size of the working buffer required for the functions `ippiFilterColumnPipeline` and `ippiFilterColumnPipeline_Low` respectively. The buffer with the length `pBufferSize[0]` can be used to filter images with width equal to or less than *roiSize*.

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if the pointer <i>pBufferSize</i> is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>maskSize</i> has a field with a zero or negative value, or if <i>roiWidth</i> is less than 1.

## FilterRowBorderPipeline, FilterRowBorderPipeline\_Low

*Apply the filter with border to image rows.*

---

### Syntax

#### Case 1: Operation on one-channel integer data

```
IppStatus ippiFilterRowBorderPipeline_<mod>(const Ipp<srcDatatype>* pSrc, int srcStep,
Ipp<dstDatatype>** ppDst, IppiSize roiSize, const Ipp<dstDatatype>* pKernel, int kernelSize,
int xAnchor, IppiBorderType borderType, Ipp<srcDatatype> borderValue, int divisor, Ipp8u*
pBuffer);
```

Supported values for mod:

8u16s\_C1R    16s\_C1R

```
IppStatus ippiFilterRowBorderPipeline_Low_16s_C1R(const Ipp16s* pSrc, int srcStep, Ipp16s**
ppDst, IppiSize roiSize, const Ipp16s* pKernel, int kernelSize, int xAnchor, IppiBorderType
borderType, Ipp16s borderValue, int divisor, Ipp8u* pBuffer);
```

#### Case 2: Operation on one-channel floating point data

```
IppStatus ippiFilterRowBorderPipeline_32f_C1R(const Ipp32f* pSrc, int srcStep, Ipp32f**
ppDst, IppiSize roiSize, const Ipp32f* pKernel, int kernelSize, int xAnchor, IppiBorderType
borderType, Ipp32f borderValue, Ipp8u* pBuffer);
```

#### Case 3: Operation on three-channel integer data

```
IppStatus ippiFilterRowBorderPipeline_<mod>(const Ipp<srcDatatype>* pSrc, int srcStep,
Ipp<dstDatatype>** ppDst, IppiSize roiSize, const Ipp<dstDatatype>* pKernel, int kernelSize,
int xAnchor, IppiBorderType borderType, Ipp<srcDatatype> borderValue[3], int divisor,
Ipp8u* pBuffer);
```

Supported values for mod:

8u16s\_C3R    16s\_C3R

```
IppStatus ippiFilterRowBorderPipeline_Low_16s_C3R(const Ipp16s* pSrc, int srcStep, Ipp16s**
ppDst, IppiSize roiSize, const Ipp16s* pKernel, int kernelSize, int xAnchor, IppiBorderType
borderType, Ipp16s borderValue[3], int divisor, Ipp8u* pBuffer);
```

#### Case 4: Operation on three-channel floating point data

```
IppStatus ippiFilterRowBorderPipeline_32f_C3R(const Ipp32f* pSrc, int srcStep, Ipp32f**
ppDst, IppiSize roiSize, const Ipp32f* pKernel, int kernelSize, int xAnchor, IppiBorderType
borderType, Ipp32f borderValue[3], Ipp8u* pBuffer);
```

### Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>ppDst</i>	Double pointer to the destination image ROI.

<i>roiSize</i>	Size of the source and destination ROI in pixels.												
<i>pKernel</i>	Pointer to the row kernel values.												
<i>kernelSize</i>	Size of the kernel in pixels.												
<i>xAnchor</i>	Anchor value specifying the kernel row alignment with respect to the position of the input pixel.												
<i>borderType</i>	Type of border (see <a href="#">Borders</a> ); following values are possible: <table data-bbox="505 541 1343 779"> <tr> <td><code>ippBorderZero</code></td><td>Values of all border pixel are set to zero.</td></tr> <tr> <td><code>ippBorderConst</code></td><td>Values of all border pixels are set to constant.</td></tr> <tr> <td><code>ippBorderRepl</code></td><td>Replicated border is used.</td></tr> <tr> <td><code>ippBorderWrap</code></td><td>Wrapped border is used</td></tr> <tr> <td><code>ippBorderMirror</code></td><td>Mirrored border is used</td></tr> <tr> <td><code>ippBorderMirrorR</code></td><td>Mirrored border with replication is used</td></tr> </table>	<code>ippBorderZero</code>	Values of all border pixel are set to zero.	<code>ippBorderConst</code>	Values of all border pixels are set to constant.	<code>ippBorderRepl</code>	Replicated border is used.	<code>ippBorderWrap</code>	Wrapped border is used	<code>ippBorderMirror</code>	Mirrored border is used	<code>ippBorderMirrorR</code>	Mirrored border with replication is used
<code>ippBorderZero</code>	Values of all border pixel are set to zero.												
<code>ippBorderConst</code>	Values of all border pixels are set to constant.												
<code>ippBorderRepl</code>	Replicated border is used.												
<code>ippBorderWrap</code>	Wrapped border is used												
<code>ippBorderMirror</code>	Mirrored border is used												
<code>ippBorderMirrorR</code>	Mirrored border with replication is used												
<i>borderValue</i>	The constant value (constant vector in case of three-channel data) to assign to the pixels in the constant border (not applicable for other border's type).												
<i>divisor</i>	Value by which the computed result is divided (for operations on integer data only).												
<i>pBuffer</i>	Pointer to the working buffer.												

## Description

The functions `ippiFilterRowBorderPipeline` and `ippiFilterRowBorderPipeline_Low` are declared in the `ippcv.h` file. They operate with ROI (see Regions [Regions of Interest in Intel IPP](#) ).

The function `ippiFilterRowBorderPipeline_Low` performs calculation exclusively with the 16s-data, and the input data must be in the range ensuring that the overflow does not occur during calculation and the result can be represented by a 32-bit integer number.

These functions apply the horizontal row filter of the separable convolution kernel to the source image *pSrc*. The filter coefficients are placed in the reversed order. For integer data:

$$ppDst[i][j] = \frac{1}{divisor} \cdot \sum_{k=0}^{kernelSize-1} pSrc[i, j+k-xAnchor] \cdot pKernel[k]$$

and for floating point data:

$$ppDst[i][j] = \sum_{k=0}^{kernelSize-1} pSrc[i, j+k-xAnchor] \cdot pKernel[k]$$

Here  $j = 0, \dots, \text{roiSize.width} - 1, i = 0, \dots, \text{roiSize.height} - 1$ . The values of pixels of the source image that lies outside of the image ROI (that is, if for pixel  $pSrc[i, j]$   $i \notin [0, \text{roiSize.width}-1]$ ) are set in accordance with the specified parameters *borderType* and *borderValue*.

This function can be used to organize the separable convolution as a step of the image processing pipeline (see [Example](#) ).

The functions requires the external buffer *pBuffer*, its size should be previously computed by the functions [ippiFilterRowBorderPipelineGetBufferSize](#) and [ippiFilterRowBorderPipelineGetBufferSize\\_Low](#) respectively

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with a zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> or <i>dstStep</i> is less than $\text{roiSize.width} * \text{pixelSize}$
<code>ippStsNotEvenStepErr</code>	Indicates an error condition if one of the step values is not divisible by 4 for floating-point images, or by 2 for short-integer images.
<code>ippStsAnchorErr</code>	Indicates an error condition if <i>xAnchor</i> has a wrong value.
<code>ippStsBorderErr</code>	Indicates an error condition if <i>borderType</i> has a wrong value.
<code>ippStsBadArgErr</code>	Indicates an error condition if <i>divisor</i> is equal to 0.

## FilterColumnPipeline, FilterColumnPipeline\_Low

Apply the filter to image columns.

### Syntax

#### Case 1: Operation on integer data

```
IppStatus ippiFilterColumnPipeline_<mod>(const Ipp<srcDatatype>** ppSrc, Ipp<dstDatatype>* pDst, int dstStep, IppiSize dstRoiSize, const Ipp<srcDatatype>* pKernel, int kernelSize, int divisor, Ipp8u* pBuffer);
```

Supported values for mod:

16s_C1R	16s8u_C1R	16s8s_C1R
16s_C3R	16s8u_C3R	16s8s_C3R

```
IppStatus ippiFilterColumnPipeline_Low_16s_C1R(const Ipp16s** ppSrc, Ipp16s* pDst, int dstStep, IppiSize dstRoiSize, const Ipp16s* pKernel, int kernelSize, int divisor, Ipp8u* pBuffer);
```

```
IppStatus ippiFilterColumnPipeline_Low_16s_C3R(const Ipp16s** ppSrc, Ipp16s* pDst, int dstStep, IppiSize dstRoiSize, const Ipp16s* pKernel, int kernelSize, int divisor, Ipp8u* pBuffer);
```



## Case 2: Operation on floating-point data

```
IppStatus ippiFilterColumnPipeline_<mod>(const Ipp<datatype>** ppSrc, Ipp<datatype>* pDst,
int dstStep, IppiSize dstRoiSize, const Ipp<datatype>* pKernel, int kernelSize, Ipp8u*
pBuffer);
```

Supported values for `mod`:

32f\_C1R

32f\_C3R

## Parameters

<i>ppSrc</i>	Double pointer to the source image ROI.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>dstRoiSize</i>	Size of the destination ROI in pixels.
<i>pKernel</i>	Pointer to the strow kernel values.
<i>kernelSize</i>	Size of the kernel in pixels.
<i>divisor</i>	Value by which the computed result is divided (for operations on integer data only).
<i>pBuffer</i>	Pointer to the working buffer.

## Description

The functions `ippiFilterColumnPipeline` and `ippiFilterColumnPipeline_Low` are declared in the `ippcv.h` file. The operates with ROI (see [Regions of Interest in Intel IPP](#)).

The function `ippiFilterColumnPipeline_Low` performs calculation exclusively with the 16s-data, and the input data must be in the range ensuring that the overflow does not occur during calculation and the result can be represented by a 32-bit integer number.

These functions apply the column filter of the separable convolution kernel to the source image *pSrc*. The filter coefficients are placed in the reversed order. For integer data:

$$pDst[i, j] = \frac{1}{divisor} \cdot \sum_{k=0}^{kernelSize-1} pSrc[i+k, j] \cdot pKernel[k]$$

and for floating point data:

$$pDst[i, j] = \sum_{k=0}^{kernelSize-1} pSrc[i+k, j] \cdot pKernel[k]$$

Here  $j = 0, \dots, \text{dstRoiSize.width}-1, i=0, \dots, \text{dstRoiSize.height}-1$ .

The size of the source image is

$(\text{dstRoiSize.height} + \text{kernelSize} - 1) * \text{dstRoiSize.width}$ .

The functions requires the external buffer *pBuffer*, its size should be previously computed by the functions *ippiFilterColumnPipelineGetBufferSize* and *ippiFilterColumnPipelineGetBufferSize\_Low* respectively.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with a zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> or <i>dstStep</i> is less than $\text{roiSize.width} * \text{<pixelSize>}$
<code>ippStsNotEvenStepErr</code>	Indicates an error condition if one of the step values is not divisible by 4 for floating-point images, or by 2 for short-integer images.
<code>ippStsBadArgErr</code>	Indicates an error condition if <i>divisor</i> is equal to 0.

## Example

The code example shows how these functions can be used to organize the separable convolution as a step of image processing pipeline.

```
void Separable_3x3(const Ipp16s* pSrc, int srcStep, Ipp16s* pDst, int dstStep,
                  IppiSize roiSize, Ipp16s* pKerX, Ipp16s* pKerY) {
    Ipp16s **get, *dst=pDst;
    const Ipp16s *src=pSrc;
    IppiSize roi;
    int todo=roiSize.height,sizeRow,sizeCol,bufLen;
    int mStep=(roiSize.width+7)&(~7),sStep=srcStep>>1,dStep=dstStep>>1;
    Ipp8u *pBufRow, *pBufCol;
    ippiFilterRowBorderPipelineGetBufferSize_Low_16s_C1R(roiSize,3,&sizeRow);
    ippiFilterColumnPipelineGetBufferSize_Low_16s_C1R(roiSize,3,&sizeCol);
    bufLen = mStep*3*sizeof(Ipp16s)+4*sizeof(Ipp16s*);
    pBufRow = ippsMalloc_8u(sizeRow);
    pBufCol = ippsMalloc_8u(sizeCol);
    get = (Ipp16s**)ippsMalloc_8u(bufLen);
    get[0]=get[1]=(Ipp16s*)(get+4);
    get[2]=get[1]+mStep;
    get[3]=get[2]+mStep;
    roi.width = roiSize.width;
    roi.height = 1;
    ippiFilterRowBorderPipeline_Low_16s_C1R(src, srcStep, get, roi, pKerX,
        3, 1, ippBorderRepl, 0, 1, pBufRow);
    if (--todo) {
        get[2] = get[0];
    } else {
        get[2] = get[0] + mStep; get[3] = get[2] + mStep;
        for (; todo>0; src+=sStep, dst+=dStep, todo--) {
            ippiFilterRowBorderPipeline_Low_16s_C1R(src, srcStep, get+2, roi, pKerX,
                3, 1, ippBorderRepl, 0, 1, pBufRow);
            ippiFilterColumnPipeline_Low_16s_C1R(get, dst, dstStep, roi, pKerY,
                3, 1, pBufCol);
            get[0] = get[1]; get[1] = get[2]; get[2] = get[3]; get[3] = get[0];
        }
    }
}
```

```

    }
    ippiFilterColumnPipeline_Low_16s_C1R(get, dst, dstStep, roi, pKerY,
        3, 1, pBufCol);

    ippsFree(pBufRow);
    ippsFree(pBufCol);
    ippsFree(get);
}

```

## DotProdCol

*Calculates the dot product of taps vector and columns of the specified set of strows.*

---

### Syntax

```

IppStatus ippiDotProdCol_32f_L2(const Ipp32f** const ppSrcRow[], const Ipp32f* pTaps, int
tapsLen, Ipp32f* pDst, int width);

```

### Parameters

<i>ppSrcRow</i>	Pointer to the set of strows.
<i>pTaps</i>	Pointer to the taps vector.
<i>tapsLen</i>	Length of taps vector, is equal to the number of strows.
<i>pDst</i>	Pointer to the destination strow.
<i>width</i>	Width of the source and destination strows.

### Description

The function `ippiDotProdCol` is declared in the `ippi.h` file.

This function calculates the dot product of filter taps vector *pTaps* and columns of the specified set of the strows *ppSrcRow*. The computations are performed in accordance with the following formula:

$$pDst[i] = \sum_{j=0}^{tapsLen-1} ppSrcRow[j][i] \cdot pTaps[j]$$

It is useful for external vertical filtering pipeline implementation.

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <i>width</i> is less than or equal to 0.

## Wiener Filters

Intel IPP functions described in this section perform adaptive noise-removal filtering of an image using Wiener filter [Lim90]. The adaptive filter is more selective than a comparable linear filter in preserving edges and other high frequency parts of an image. Wiener filters are commonly used in image processing applications to remove additive noise from degraded images, to restore a blurry image, and in similar operations.

These functions use a pixel-wise adaptive Wiener method based on statistics estimated from a local neighborhood (mask) of arbitrary size for each pixel.

### FilterWienerGetBufferSize

*Computes the size of the external buffer for `ippiFilterWiener` function.*

---

#### Syntax

```
IppStatus ippiFilterWienerGetBufferSize(IppiSize dstRoiSize, IppiSize maskSize, int channels, int* pBufferSize);
```

#### Parameters

<code>dstRoiSize</code>	Size of the destination ROI in pixels.
<code>maskSize</code>	Size of the mask in pixels.
<code>channels</code>	Number of channels in the image.
<code>pBufferSize</code>	Pointer to the computed value of the external buffer size.

#### Description

The function `ippiFilterWienerGetBufferSize` is declared in the `ippi.h` file. This function computes the size in bytes of an external memory buffer that is required for the function `ippiFilterWiener`, and stores the result in the `pBufferSize`.

#### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if the <code>pBufferSize</code> pointer is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if one of the fields of <code>dstRoiSize</code> has a zero or negative value.
<code>ippStsMaskSizeErr</code>	Indicates an error condition if one of the fields of <code>maskSize</code> has a value less than or equal to 1.
<code>ippStsNumChannelsErr</code>	Indicates an error condition if <code>channels</code> is not 1, 3 or 4.

## FilterWiener

*Filters an image using the Wiener algorithm.*

---

### Syntax

#### Case 1: Operation on one-channel images

```
IppStatus ippiFilterWiener_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>*
pDst, int dstStep, IppiSize dstRoiSize, IppiSize maskSize, IppiPoint anchor, Ipp32f
noise[1], Ipp8u* pBuffer);
```

Supported values for mod:

8u\_C1R            16s\_C1R            32f\_C1R

#### Case 2: Operation on multi-channel images

```
IppStatus ippiFilterWiener_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>*
pDst, int dstStep, IppiSize dstRoiSize, IppiSize maskSize, IppiPoint anchor, Ipp32f
noise[3], Ipp8u* pBuffer);
```

Supported values for mod:

8u\_C3R            16s\_C3R            32f\_C3R  
8u\_AC4R           16s\_AC4R           32f\_AC4R

```
IppStatus ippiFilterWiener_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>*
pDst, int dstStep, IppiSize dstRoiSize, IppiSize maskSize, IppiPoint anchor, Ipp32f
noise[4], Ipp8u* pBuffer);
```

Supported values for mod:

8u\_C4R            16s\_C4R            32f\_C4R

### Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>dstRoiSize</i>	Size of the source and destination ROI in pixels.
<i>maskSize</i>	Size of the mask in pixels.
<i>anchor</i>	Anchor cell specifying the mask alignment with respect to the position of the input pixel.
<i>noise</i>	Noise level value or array of the noise level values in case of multi-channel image. This value must be in the range [0,1].
<i>pBuffer</i>	Pointer to the external work buffer.

## Description

The function `ippiFilterWiener` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)). This function performs adaptive filtering of the image degraded by constant power additive noise. For each pixel of the input image `pSrc`, the function estimates the local image mean  $\mu$  and variance  $\sigma$  in the rectangular neighborhood (mask) of size `maskSize` with the anchor cell `anchor` centered on the pixel. The anchor cell is specified by its coordinates `anchor.x` and `anchor.y` in the coordinate system associated with the bottom right corner of the mask.

The following formulas are used in computations:

$$\mu_{i,j} = \frac{1}{HW} \cdot \sum_{m=0}^{H-1} \sum_{n=0}^{W-1} x_{m,n}$$

$$\sigma_{i,j}^2 = \frac{1}{HW} \cdot \sum_{m=0}^{H-1} \sum_{n=0}^{W-1} x_{m,n}^2 - \mu_{i,j}^2$$

Here  $\mu_{i,j}$  and  $\sigma_{i,j}$  stand for local mean and variance for pixel  $x_{i,j}$ , respectively, and  $H$ ,  $W$  are the vertical and horizontal sizes of the mask, respectively.

The corresponding value for the output pixel  $y_{i,j}$  is computed as:

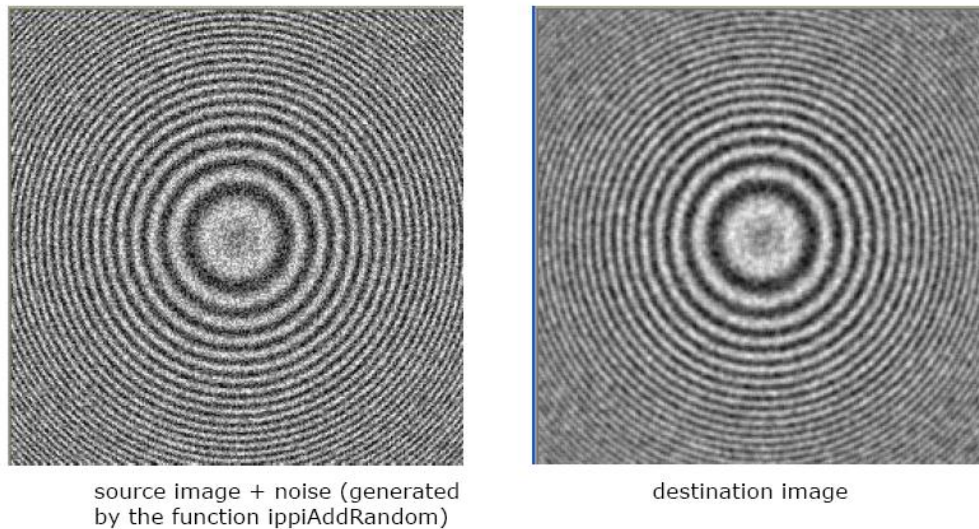
$$Y_{i,j} = \mu_{i,j} + \frac{\sigma_{i,j}^2 - v^2}{\sigma^2} \cdot [X_{i,j} - \mu_{i,j}]$$

and stored in the `pDst`. Here  $v^2$  is the noise variance, specified for each channel by the noise level parameter `noise`. If this parameter is not defined (`noise = 0`), then the function estimates the noise level by averaging through the image of all local variances  $\sigma_{i,j}$ , and stores the corresponding values in the `noise` for further use.

The function `ippiFilterWiener` uses the external work buffer `pBuffer`, which must be allocated before the function call. To determine the required buffer size, the function `ippiFilterWienerGetBufferSize` can be used.

Figure “Applying the function `ippiFilterWiener`” illustrates the result of using `ippiFilterWiener_32f_C1R` function.

### Applying the function `ippiFilterWiener`



### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if one of the fields of <code>dstRoiSize</code> has a zero or negative value.
<code>ippStsMaskSizeErr</code>	Indicates an error condition if one of the fields of <code>maskSize</code> has a value less than or equal to 1.
<code>ippStsNoiseRangeErr</code>	Indicates an error condition if one of the <code>noise</code> values is less than 0 or greater than 1.

### Example

The code example below shows how to use the function `ippiFilterWiener_32f_C1R`.

```
#include "stdio.h"
#include "ipp.h"

int main()
{
    IppStatus stat = ippStsNoErr;

    int length = 256*256;
    Ipp32f pSrc[256*256];
    Ipp32f *pSrcW;
    Ipp32f pDst[256*256];
    int srcStep = 256*sizeof(Ipp32f);
    int dstStep = 256*sizeof(Ipp32f);
```

```

IppiSize roiSize = {256, 256};
IppiSize maskSize = {3, 3};
IppiSize dstRoiSize = {256-maskSize.width, 256-maskSize.height};

int pBufferSize;
int channels = 1;
unsigned int pSeed = 3;

IppiPoint anchor = {1, 1};
Ipp32f noise[1] = {0.0};
Ipp8u* pBuffer;

stat = ippiImageJaehne_32f_C1R(pSrc, srcStep, roiSize);
if( stat != ippStsNoErr){
    printf (" ERROR !!! \n");
    return 1;
}

stat = ippiFilterWienerGetBufferSize( dstRoiSize, maskSize, channels,
&pBufferSize);
if( stat != ippStsNoErr ){
    printf (" ERROR !!! \n");
    return 1;
}

stat = ippsSet_32f( 1.0, pDst, length );

pBuffer = ippsMalloc_8u(pBufferSize);
pSrcW = (Ipp32f*)((Ipp8u*)pSrc + (maskSize.height - 1 - anchor.y ) *
srcStep + (maskSize.width - 1 - anchor.x ) * sizeof(Ipp32f) );

stat = ippiFilterWiener_32f_C1R( pSrcW, srcStep, pDst, dstStep,
dstRoiSize, maskSize, anchor, noise, pBuffer);

if( stat != ippStsNoErr ){
    printf (" ERROR !!! \n");
    return 1;
}

ippsFree(pBuffer);
return 0;
} // __main__

```

## Convolution

Intel IPP functions described in this section perform two-dimensional finite linear convolution operation between two source images and write the result into the destination image. Convolution is used to perform many common image processing operations including sharpening, blurring, noise reduction, embossing, and edge enhancement. For convenience, any digital image  $f$  is represented here as a matrix with  $M_f$  columns and  $N_f$  rows that contains pixel values  $f[i, j]$ ,  $0 \leq i < M_f$ ,  $0 \leq j < N_f$ .



## ConvFull

*Performs a full convolution of two images.*

---

### Syntax

#### Case 1: Operation on integer data

```
IppStatus ippiConvFull_<mod>(const Ipp<datatype>* pSrc1, int src1Step, IppiSize src1Size,
const Ipp<datatype>* pSrc2, int src2Step, IppiSize src2Size, Ipp<datatype>* pDst, int
dstStep, int divisor);
```

Supported values for mod

8u_C1R	16s_C1R
8u_C3R	16s_C3R
8u_AC4R	16s_AC4R

#### Case 2: Operation on floating-point data

```
IppStatus ippiConvFull_<mod>(const Ipp32f* pSrc1, int src1Step, IppiSize src1Size, const
Ipp32f* pSrc2, int src2Step, IppiSize src2Size, Ipp32f* pDst, int dstStep);
```

Supported values for mod

32f_C1R
32f_C3R
32f_AC4R

### Parameters

<i>pSrc1, pSrc2</i>	Pointers to the source images ROI.
<i>src1Step, src2Step</i>	Distances in bytes between starts of consecutive lines in the source images.
<i>src1Size, src2Size</i>	Sizes in pixels of the source images.
<i>pDst</i>	Pointer to the destination buffer ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>divisor</i>	The integer value by which the computed result is divided (for operations on integer data only).

### Description

The function `ippiConvFull` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function performs a full two-dimensional finite linear convolution operation between two source images pointed to by *pSrc1* and *pSrc2*. If we denote the first source image as a matrix  $f$  of size  $M_f$  by  $N_f$  and the second source image as a matrix  $g$  of size  $M_g$  by  $N_g$ , then the destination image  $h$  obtained as a result of the function operation will have the size  $M_h$  by  $N_h$ , where  $M_h = M_f + M_g - 1$  and  $N_h = N_f + N_g - 1$ .

The function `ippiConvFull` implements the following equation to compute values  $h[i, j]$  of the destination image:

$$h[i, j] = \frac{1}{divisor} \sum_{l=0}^{N_h-1} \sum_{k=0}^{M_h-1} f[k, l] \times g[i-k, j-l] \quad ,$$

where  $0 \leq i < M_h$ ,  $0 \leq j < N_h$  and

$$f[k, l] = \begin{cases} f[k, l], & 0 \leq k < M_f; \quad 0 \leq l < N_f \\ 0 & , otherwise \end{cases}$$

$$g[i-k, j-l] = \begin{cases} g[i-k, j-l], & 0 \leq i-k < M_g; \quad 0 \leq j-l < N_g \\ 0 & , otherwise \end{cases}$$

Function flavors that accept input data of the `Ipp32f` type use the same summation formula, but no scaling of the result is done ( $divisor = 1$  is assumed).

To illustrate the function operation, for the source images  $f, g$  of size  $3 \times 5$  represented as

$$f = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 0 & 0 \\ 1 & 1 & 1 \\ 0 & 0 & 1 \\ 1 & 1 & 1 \end{bmatrix} \quad , \quad g = f$$

with  $g = f$ ,

the resulting convolution image  $h$  is of size  $5 \times 9$  and contains the following data:

$$h = \begin{bmatrix} 1 & 2 & 3 & 2 & 1 \\ 2 & 2 & 2 & 0 & 0 \\ 3 & 4 & 6 & 4 & 2 \\ 2 & 2 & 4 & 2 & 2 \\ 3 & 6 & 11 & 6 & 3 \\ 2 & 2 & 4 & 2 & 2 \\ 2 & 4 & 6 & 4 & 3 \\ 0 & 0 & 2 & 2 & 2 \\ 1 & 2 & 3 & 2 & 1 \end{bmatrix}$$

**Example** shows how to use the function `ippiConvFull_16s_C1R`.

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pSrc1</i> , <i>pSrc2</i> , or <i>pDst</i> is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>src1Size</i> or <i>src2Size</i> has a zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <i>src1Step</i> , <i>src2Step</i> , or <i>dstStep</i> has a zero or negative value.
<code>ippStsDivisorErr</code>	Indicates an error condition if <i>divisor</i> has the zero value.
<code>ippStsMemAllocErr</code>	Indicates an error condition if memory allocation failed.

## ConvValid

*Performs a valid convolution of two images.*

### Syntax

#### Case 1: Operation on integer data

```
ippStatus ippiConvValid_<mod>(const Ipp<datatype>* pSrc1, int src1Step, IppiSize src1Size,
const Ipp<datatype>* pSrc2, int src2Step, IppiSize src2Size, Ipp<datatype>* pDst, int
dstStep, int divisor);
```

Supported values for *mod*:

<code>8u_C1R</code>	<code>16s_C1R</code>
<code>8u_C3R</code>	<code>16s_C3R</code>
<code>8u_AC4R</code>	<code>16s_AC4R</code>

## Case 2: Operation on floating-point data

```
IppStatus ippiConvValid_<mod>(const Ipp32f* pSrc1, int src1Step, IppiSize src1Size, const
Ipp32f* pSrc2, int src2Step, IppiSize src2Size, Ipp32f* pDst, int dstStep);
```

Supported values for `mod`:

```
32f_C1R
32f_C3R
32f_AC4R
```

## Parameters

<i>pSrc1, pSrc2</i>	Pointers to the source images ROI.
<i>src1Step, src2Step</i>	Distances in bytes between starts of consecutive lines in the source images.
<i>src1Size, src2Size</i>	Sizes in pixels of the source images.
<i>pDst</i>	Pointer to the destination buffer ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>divisor</i>	The integer value by which the computed result is divided (for operations on integer data only).

## Description

The function `ippiConvValid` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)). This function performs valid two-dimensional finite linear convolution operation between two source images pointed to by *pSrc1* and *pSrc2*.

If we denote the first source image as a matrix  $f$  of size  $M_f$  by  $N_f$  and the second source image as a matrix  $g$  of size  $M_g$  by  $N_g$ , then the destination image  $h$  obtained as a result of the function operation will have size  $M_h$  by  $N_h$ , where  $M_h = |M_f - M_g| + 1$  and  $N_h = |N_f - N_g| + 1$ .

The function `ippiConvFull` implements the following equation to compute values  $h[i, j]$  of the destination image:

$$h[i, j] = \frac{1}{divisor} \sum_{l=0}^{N_g-1} \sum_{k=0}^{M_g-1} f[i+k, j+l] \times g[M_g-k-1, N_g-l-1]$$

where  $0 \leq i < M_h$ ,  $0 \leq j < N_h$ .

Assume here that  $M_f \geq M_g$  and  $N_f \geq N_g$ . In case when  $M_f < M_g$  and  $N_f < N_g$ , the subscript index  $g$  in this equation must be replaced with the index  $f$ . For any other combination of source image sizes, the function `ippiConvValid` performs no operation.

Note that the above formula gives the same result as in the case of `ippiConvFull` function, but produces only that part of the convolution image which is computed without using zero-padded values.

Function flavors that accept input data of the `Ipp32f` type use the same summation formula, but no scaling of the result is done ( $divisor = 1$  is assumed).

To illustrate the function operation, for the source images  $f$ ,  $g$  of size  $3 \times 5$  represented as

$$f = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 0 & 0 \\ 1 & 1 & 1 \\ 0 & 0 & 1 \\ 1 & 1 & 1 \end{bmatrix}, \quad g = f,$$

with  $g = f$ ,

the resulting convolution image  $h$  is of size  $1 \times 1$  and contains the following data:

$h = [11]$ .

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <code>pSrc1</code> , <code>pSrc2</code> , or <code>pDst</code> is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <code>src1Size</code> or <code>src2Size</code> has a zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <code>src1Step</code> , <code>src2Step</code> , or <code>dstStep</code> has a zero or negative value.
<code>ippStsDivisorErr</code>	Indicates an error condition if <code>divisor</code> has a zero value.
<code>ippStsMemAllocErr</code>	Indicates an error condition if memory allocation failed.

## Example

The code example below shows how to use the function `ippiConvValid_16s_C1R`.

```
Ipp8u src1[8*4]={ 0, 1, 2, 3, 4, 5, 6, 7,
                  0, 1, 2, 3, 4, 5, 6, 7,
                  7, 6, 5, 4, 3, 2, 1, 0,
                  7, 6, 5, 4, 3, 2, 1, 0};

Ipp16s src2[3*3];
Ipp16s dst1[6*6];
Ipp16s dst2[2*2];
IppiSize src1Size = { 4, 4 };
IppiSize src2Size = { 3, 3 };
int divisor = 2;
int sign = 1;

ippiGenSobelKernel_16s ( src2, 9, 8, sign);    // using "Sobel" kernel

ippiConvFull_16s_C1R ( src1, 8*sizeof(Ipp16s), src1Size, src2,
```

```

        3*sizeof(Ipp16s), src2Size, dst1, 6*sizeof(Ipp16s), 2 );
ippiConvValid_16s_C1R ( src1, 8*sizeof(Ipp16s),  src1Size,  src2,
        3*sizeof(Ipp16s), src2Size, dst2, 2*sizeof(Ipp16s), 2 );

```

Result:

```

0 1 2 3
0 1 2 3
7 6 5 4      src1
7 6 5 4

```

```

 1 -8 28
-56 70 -56      src2
28 -8 1

```

```

 0  0 -3  8 16 42
 0 -28 -24 -34 65 -42
 4 -39 80 58 92 -26
-192 66 -26 -4 43 -54      Output of ippiConvFull  (dst1)
-98 133 -76 -66 -14 -110
 98 56 50 39 -14 2

      80 59      Output of ippiConvValid (dst2)
     -26 -5

```

## Deconvolution

Functions described in this section perform image deconvolution. They can be used for restoring the degraded image, in particular image that was obtained by applying the convolution operation with known kernel. The Intel IPP functions implement two methods: the Fourier deconvolution (noniterative method) [see for example, [\[Puetter05\]](#)], and the Richardson-Lucy method (iterative method) [\[Ric72\]](#). Border pixels of a source image are restored before deconvolution.

### DeconvFFTInitAlloc

*Allocates and initializes state structure for FFT deconvolution.*

---

#### Syntax

```

IppStatus ippiDeconvFFTInitAlloc_32f_C1R(IppiDeconvFFTState_32f_C1R** ppDeconvFFTState,
const Ipp32f* pKernel, int kernelSize, int FFTorder, Ipp32f threshold);

```

```

IppStatus ippiDeconvFFTInitAlloc_32f_C3R(IppiDeconvFFTState_32f_C3R** ppDeconvFFTState,
const Ipp32f* pKernel, int kernelSize, int FFTorder, Ipp32f threshold);

```

#### Parameters

<i>ppDeconvFFTState</i>	Double pointer to the FFT deconvolution state structure.
<i>pKernel</i>	Pointer to the kernel array.
<i>kernelSize</i>	Size of the kernel.
<i>FFTorder</i>	Order of the created FFT state structure.

*threshold* Value of the threshold level (to exclude dividing by zero).

## Description

The function `ippiDeconvFFTInitAlloc` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function allocates memory, initializes the deconvolution state structure and returns the pointer `ppDeconvFFTState` to it. This structure is used by the function `ippiDeconvFFT` that performs deconvolution of the source image using FFT.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <i>kernelSize</i> is less than or equal to 0, or if <i>kernelSize</i> is greater than $2^{FFTOrder}$ .
<code>ippStsBadArgErr</code>	Indicates an error condition if <i>threshold</i> is less than or equal to 0.

## DeconvFFTFree

*Frees memory allocated for the FFT deconvolution state structure.*

## Syntax

```
IPPStatus ippiDeconvFFTFree_32f_C1R(IppiDeconvFFTState_32f_C1R* pDeconvFFTState);
IPPStatus ippiDeconvFFTFree_32f_C3R(IppiDeconvFFTState_32f_C3R* pDeconvFFTState);
```

## Parameters

*pDeconvFFTState* Pointer to the FFT deconvolution state structure.

## Description

The function `ippiDeconvFFTFree` is declared in the `ippi.h` file. This function frees memory allocated by the function `ippiDeconvFFTInitAlloc` for the FFT deconvolution state structure `pDeconvFFTState`.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pDeconvFFTState</i> is <code>NULL</code> .

## DeconvFFT

*Performs FFT deconvolution of an image.*

---

### Syntax

```
IppStatus ippiDeconvFFT_32f_C1R(const Ipp32f* pSrc, int srcStep, Ipp32f* pDst, int dstStep,
IppiSize roiSize, IppiDeconvFFTState_32f_C1R* pDeconvFFTState);
```

```
IppStatus ippiDeconvFFT_32f_C3R(const Ipp32f* pSrc, int srcStep, Ipp32f* pDst, int dstStep,
IppiSize roiSize, IppiDeconvFFTState_32f_C3R* pDeconvFFTState);
```

### Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>roiSize</i>	Size of the source and destination image ROI.
<i>pDeconvFFTState</i>	Pointer to the FFT deconvolution state structure.

### Description

The function `ippiDeconvFFT` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function performs deconvolution of the source image *pSrc* using FFT with parameters specified in the FFT deconvolution state structure *pDeconvFFTState* and stores results to the destination image *pDst*. The FFT deconvolution state structure must be initialized by calling the function `ippiDeconvFFTInitAlloc` beforehand.

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointers is <code>NULL</code> .

## DeconvLRInitAlloc

*Allocates and initializes state structure for LR deconvolution.*

---

### Syntax

```
IppStatus ippiDeconvLRInitAlloc_32f_C1R(IppiDeconvLR_32f_C1R** ppDeconvLR, const Ipp32f*
pKernel, int kernelSize, IppiSize maxRoi, Ipp32f threshold);
```

```
IppStatus ippiDeconvLRInitAlloc_32f_C3R(IppiDeconvLR_32f_C3R** ppDeconvLR, const Ipp32f*
pKernel, int kernelSize, IppiSize maxRoi, Ipp32f threshold);
```



## Parameters

<i>ppDeconvLR</i>	Double pointer to the LR deconvolution state structure.
<i>pKernel</i>	Pointer to the kernel array.
<i>kernelSize</i>	Size of the kernel.
<i>maxRoi</i>	Maximum size of the image ROI in pixels.
<i>threshold</i>	Value of the threshold level (to except dividing by zero).

## Description

The function `ippiDeconvLRInitAlloc` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function allocates memory, initializes the deconvolution state structure and returns the pointer *ppDeconvLR* to it. This structure is used by the function [ippiDeconvLR](#) that performs deconvolution of the source image using the Lucy-Richardson algorithm.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <i>kernelSize</i> is less than or equal to 0, or if <i>kernelSize</i> is greater than <i>maxRoi.width</i> or <i>maxRoi.height</i> , or if any field of the <i>maxRoi</i> is less than or equal to 0.
<code>ippStsBadArgErr</code>	Indicates an error condition if threshold is less than or equal to 0.

## DeconvLRFree

*Frees memory allocated for the LR deconvolution state structure.*

## Syntax

```
IppStatus ippiDeconvLRFree_32f_C1R(IppiDeconvLR_32f_C1R* pDeconvLR);
IppStatus ippiDeconvLRFree_32f_C3R(IppiDeconvLR_32f_C3R* pDeconvLR);
```

## Parameters

<i>pDeconvLR</i>	Pointer to the LR deconvolution state structure.
------------------	--

## Description

The function `ippiDeconvLRFree` is declared in the `ippi.h` file. This function frees memory allocated by the function [ippiDeconvLRInitAlloc](#) for the LR deconvolution state structure *pDeconvLR*.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pDeconvLR</i> is <code>NULL</code> .

## DeconvLR

*Performs LR deconvolution of an image.*

---

### Syntax

```
IppStatus ippiDeconvLR_32f_C1R(const Ipp32f* pSrc, int srcStep, Ipp32f* pDst, int dstStep,
IppiSize roiSize, int numIter, IppiDeconvLR_32f_C1R* pDeconvLR);
```

```
IppStatus ippiDeconvLR_32f_C3R(const Ipp32f* pSrc, int srcStep, Ipp32f* pDst, int dstStep,
IppiSize roiSize, int numIter, IppiDeconvLR_32f_C3R* pDeconvLR);
```

### Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>roiSize</i>	Size of the source and destination image ROI.
<i>numIter</i>	Number of algorithm iterations.
<i>pDeconvLR</i>	Pointer to the LR deconvolution state structure.

### Description

The function `ippiDeconvLR` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function performs deconvolution of the source image *pSrc* using the Lucy-Richardson algorithm with parameters specified in the state structure *pDeconvLR* and stores results to the destination image *pDst*. The Lucy-Richardson deconvolution state structure must be initialized by calling the function `ippiDeconvLRInitAlloc` beforehand.

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roi.width</i> or <i>roi.height</i> is less than or equal to 0, or if <i>roi.width</i> is greater than ( <i>maxRoi.width</i> - <i>kernelSize</i> ), or <i>roi.height</i> is greater than ( <i>maxRoi.height</i> - <i>kernelSize</i> ).
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> or <i>dstStep</i> is less than <i>roiSize.width</i> * <i>&lt;pixelSize&gt;</i> .
<code>ippStsNotEvenStepErr</code>	Indicates an error condition if steps for floating-point images are not divisible by 4.
<code>ippStsBadArgErr</code>	Indicates an error condition if <i>numIter</i> is less than or equal to 0.

## Fixed Filters

The fixed filter functions perform linear filtering of a source image using one of the predefined convolution kernels. The supported fixed filters and their respective kernel sizes are listed in the following table:

### Types of the Fixed Filter Functions

Fixed Filter Type	Kernel Size
Horizontal Prewitt operator	3x3
Vertical Prewitt operator	3x3
Horizontal Scharr operator	3x3
Vertical Scharr operator	3x3
Horizontal Sobel operator	3x3 or 5x5
Vertical Sobel operator	3x3 or 5x5
Second derivative horizontal Sobel operator	3x3 or 5x5
Second derivative vertical Sobel operator	3x3 or 5x5
Second cross derivative Sobel operator	3x3 or 5x5
Horizontal Roberts operator	3x3
Vertical Roberts operator	3x3
Laplacian highpass filter	3x3 or 5x5
Gaussian lowpass filter	3x3 or 5x5
Highpass filter	3x3 or 5x5
Lowpass filter	3x3 or 5x5
Sharpening filter	3x3

Using fixed filter functions with predefined kernels is more efficient as it eliminates the need to create the convolution kernel in your application program.



**NOTE.** For all fixed filter functions, to ensure valid operation when image boundary pixels are processed, the application should correctly define additional border pixels (see [Borders](#)).



**NOTE.** The anchor cell is the center cell of the kernel for all fixed filters.

## FilterPrewittHoriz

*Filters an image using a horizontal Prewitt kernel.*

### Syntax

```
ippiStatus ippiFilterPrewittHoriz_<mod>(const Ipp<datatype>* pSrc, int srcStep,
ippiStatus ippiFilterPrewittHoriz_<mod>(const Ipp<datatype>* pSrc, int srcStep,
Ipp<datatype>* pDst, int dstStep, IppiSize dstRoiSize);
```

Supported values for mod:

8u_C1R	16s_C1R	32f_C1R
8u_C3R	16s_C3R	32f_C3R
8u_C4R	16s_C4R	32f_C4R

8u\_AC4R            16s\_AC4R            32f\_AC4R

## Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>dstRoiSize</i>	Size of the source and destination ROI in pixels.

## Description

The function `ippiFilterPrewittHoriz` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)). This function applies a horizontal Prewitt operator to an image ROI. The corresponding kernel is the matrix of 3x3 size with the following values:

```

  1   1   1
  0   0   0
 -1  -1  -1

```

The anchor cell is the center cell of the kernel (red). The size of the source image ROI is equal to *dstRoiSize*, the size of the destination image ROI.

This filter has the effect of enhancing horizontal edges of an image.

To ensure valid operation when image boundary pixels are processed, the application should correctly define additional border pixels (see [Borders](#)).

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pSrc</i> or <i>pDst</i> is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>dstRoiSize</i> has a field with a zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> or <i>dstStep</i> has a zero or negative value.

## FilterPrewittVert

*Filters an image using a vertical Prewitt kernel.*

---

### Syntax

```

IppStatus ippiFilterPrewittVert_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>*
pDst, int dstStep, IppiSize dstRoiSize);

```

Supported values for `mod`:

8u\_C1R            16s\_C1R            32f\_C1R  
8u\_C3R            16s\_C3R            32f\_C3R

8u_C4R	16s_C4R	32f_C4R
8u_AC4R	16s_AC4R	32f_AC4R

## Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>dstRoiSize</i>	Size of the source and destination ROI in pixels.

## Description

The function `ippiFilterPrewittVert` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)). This function applies a vertical Prewitt operator to an image ROI. The corresponding kernel is the matrix of 3x3 size with the following values:

```
-1  0  1
-1  0  1
-1  0  1
```

The anchor cell is the center cell (red) of the kernel. The size of the source image ROI is equal to *dstRoiSize*, the size of the destination image ROI.

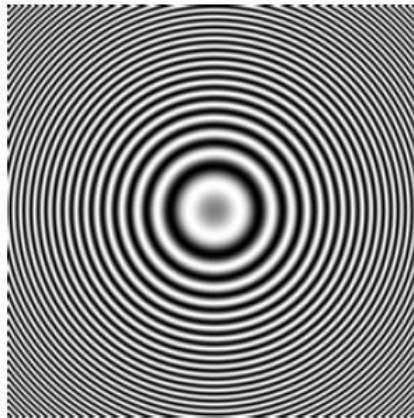
This filter has the effect of enhancing vertical edges of an image.

To ensure valid operation when image boundary pixels are processed, the application should correctly define additional border pixels (see [Borders](#)).

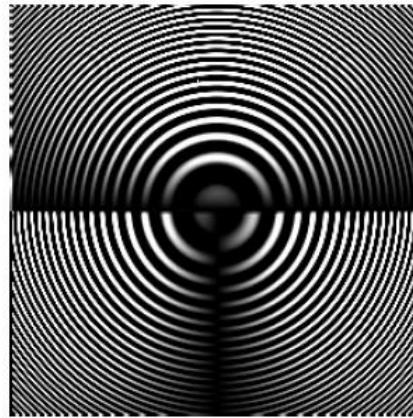
Figure "Using Prewitt Operator for Image Filtering" shows the result of filtering the sample image using the horizontal (upper part) and vertical (lower part) Prewitt operators. All pixels with negative values are zeroed because the function for `unsigned char` values was used.

### Using Prewitt Operator for Image Filtering

---



Sample Image



Filtered Image

---

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <code>pSrc</code> or <code>pDst</code> is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <code>dstRoiSize</code> has a field with a zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <code>srcStep</code> or <code>dstStep</code> has a zero or negative value.

## FilterScharrHoriz

*Filters an image using a horizontal Scharr kernel.*

---

### Syntax

```
IppStatus ippiFilterScharrHoriz_<mod>(const Ipp<srcDatatype>* pSrc, int srcStep,
Ipp<dstDatatype>* pDst, int dstStep, IppiSize dstRoiSize);
```

Supported values for `mod`:

`8u16s_C1R`      `8s16s_C1R`      `32f_C1R`

### Parameters

`pSrc`      Pointer to the source image ROI.

<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>dstRoiSize</i>	Size of the source and destination ROI in pixels.

## Description

The function `ippiFilterScharrHoriz` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)). This function applies a horizontal Scharr operator to an image ROI. The corresponding kernel is the matrix of 3x3 size with the following values:

```

3   10  3
0   0  0
-3  -10 -3
```

The anchor cell is the center cell (red) of the kernel. The size of the source image ROI is equal to *dstRoiSize*, the size of the destination image ROI.

This filter has the effect of both enhancing and smoothing horizontal edges of an image.

To ensure valid operation when image boundary pixels are processed, the application should correctly define additional border pixels (see [Borders](#)).

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pSrc</i> or <i>pDst</i> is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>dstRoiSize</i> has a field with a zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> or <i>dstStep</i> has a zero or negative value.

## FilterScharrVert

*Filters an image using a vertical Scharr kernel.*

### Syntax

```

IppStatus ippiFilterScharrVert_<mod>(const Ipp<srcDatatype>* pSrc, int srcStep,
Ipp<dstDatatype>* pDst, int dstStep, IppiSize dstRoiSize);
```

Supported values for `mod`:

```

8u16s_C1R      8s16s_C1R      32f_C1R
```

## Parameters

<i>pSrc</i>	Pointer to the source image ROI.
-------------	----------------------------------

<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>dstRoiSize</i>	Size of the source and destination ROI in pixels.

## Description

The function `ippiFilterScharVert` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)). This function applies a vertical Scharr operator to an image ROI. The corresponding kernel is the matrix of 3x3 size with the following values:

```

3   0   -3
10  0 -10
3   0   -3

```

The anchor cell is the center cell of the kernel (red). The size of the source image ROI is equal to *dstRoiSize*, the size of the destination image ROI.

This filter has the effect of both enhancing and smoothing vertical edges of an image.

To ensure valid operation when image boundary pixels are processed, the application should correctly define additional border pixels (see [Borders](#)).

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pSrc</i> or <i>pDst</i> is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>dstRoiSize</i> has a field with a zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> or <i>dstStep</i> has a zero or negative value.

## FilterSobelHoriz, FilterSobelHorizMask

*Filters an image using a horizontal Sobel kernel.*

---

### Syntax

```
IppStatus ippiFilterSobelHoriz_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize dstRoiSize);
```

Supported values for *mod*:

8u_C1R	16s_C1R	32f_C1R
8u_C3R	16s_C3R	32f_C3R
8u_C4R	16s_C4R	32f_C4R
8u_AC4R	16s_AC4R	32f_AC4R



```
IppStatus ippiFilterSobelHoriz_<mod>(const Ipp<srcDatatype>* pSrc, int srcStep,
Ipp<dstDatatype>* pDst, int dstStep, IppiSize dstRoiSize, IppiMaskSize mask);
```

Supported values for mod:

8u16s\_C1R 8s16s\_C1R

```
IppStatus ippiFilterSobelHorizMask_32f_C1R(const Ipp32f* pSrc, int srcStep, Ipp32f* pDst,
int dstStep, IppiSize dstRoiSize, IppiMaskSize mask);
```

## Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>dstRoiSize</i>	Size of the source and destination ROI in pixels.
<i>mask</i>	Predefined mask of IppiMaskSize type.

## Description

The functions `ippiFilterSobelHoriz` and `ippiFilterSobelHorizMask` are declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)). These functions apply a horizontal Sobel operator to an image ROI. The appropriate kernel is the matrix of 3x3 size, or either 3x3 or 5x5 size in accordance with the *mask* parameter of the corresponding function flavors. The kernels have the following values:

				1	4	6	4	1
1	2	1		2	8	12	8	2
0	0	0	or	0	0	0	0	0
-1	-2	-1		-2	-8	-12	-8	-2
				-1	-4	-6	-4	-1

The anchor cell is the center cell of the kernel (red). The size of the source image ROI is equal to *dstRoiSize*, the size of the destination image ROI.

This filter has the effect of both enhancing and smoothing horizontal edges of an image. To ensure valid operation when image boundary pixels are processed, the application should correctly define additional border pixels (see [Borders](#)).

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pSrc</i> or <i>pDst</i> is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>dstRoiSize</i> has a field with a zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> or <i>dstStep</i> has a zero or negative value.

`ippStsMaskSizeErr`Indicates an error condition if *mask* has an illegal value.

## FilterSobelVert, FilterSobelVertMask

*Filters an image using a vertical Sobel kernel.*

---

### Syntax

```
IppStatus ippifilterSobelVert_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>*
pDst, int dstStep, IppiSize dstRoiSize);
```

Supported values for *mod*:

8u_C1R	16s_C1R	32f_C1R
8u_C3R	16s_C3R	32f_C3R
8u_C4R	16s_C4R	32f_C4R
8u_AC4R	16s_AC4R	32f_AC4R

```
IppStatus ippifilterSobelVert_<mod>(const Ipp<srcDatatype>* pSrc, int srcStep,
Ipp<dstDatatype>* pDst, int dstStep, IppiSize dstRoiSize, IppiMaskSize mask);
```

Supported values for *mod*:

8u16s_C1R	8s16s_C1R
-----------	-----------

```
IppStatus ippifilterSobelVertMask_32f_C1R(const Ipp32f* pSrc, int srcStep, Ipp32f* pDst,
int dstStep, IppiSize dstRoiSize, IppiMaskSize mask);
```

### Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>dstRoiSize</i>	Size of the source and destination ROI in pixels.
<i>mask</i>	Predefined mask of <code>IppiMaskSize</code> type.

## Description

The functions `ippiFilterSobelVert` and `ippiFilterSobelVertMask` are declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)). These functions apply a vertical Sobel operator to an image ROI. The appropriate kernel is the matrix of 3x3 size, or either 3x3 or 5x5 size in accordance with the `mask` parameter of the corresponding function flavors. The kernels have the following values:

				-1	-2	0	2	1
-1	0	1		-4	-8	0	8	4
-2	0	2	or	-6	-12	0	12	6
-1	0	1		-4	-8	0	8	4
				-1	-2	0	2	1

The anchor cell is the center cell of the kernel (red). The size of the source image ROI is equal to `dstRoiSize`, the size of the destination image ROI.

This filter has the effect of both enhancing and smoothing vertical edges of an image. To ensure valid operation when image boundary pixels are processed, the application should correctly define additional border pixels (see [Borders](#)).

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <code>pSrc</code> or <code>pDst</code> is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <code>dstRoiSize</code> has a field with a zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <code>srcStep</code> or <code>dstStep</code> has a zero or negative value.
<code>ippStsMaskSizeErr</code>	Indicates an error condition if <code>mask</code> has an illegal value.

## FilterSobelHorizSecond

*Filters an image using a second derivative horizontal Sobel operator.*

### Syntax

```
ippStatus ippiFilterSobelHorizSecond_<mod>(const Ipp<srcDatatype>* pSrc, int srcStep,
Ipp<dstDatatype>* pDst, int dstStep, IppiSize dstRoiSize, IppiMaskSize mask);
```

Supported values for `mod`:

8u16s\_C1R      8s16s\_C1R      32f\_C1R

## Parameters

<code>pSrc</code>	Pointer to the source image ROI.
<code>srcStep</code>	Distance in bytes between starts of consecutive lines in the source image.

<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>dstRoiSize</i>	Size of the source and destination ROI in pixels.
<i>mask</i>	Predefined mask of <code>IppiMaskSize</code> type.

## Description

The function `ippiFilterSobelHorizSecond` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)). This function applies a second derivative horizontal Sobel operator to an image ROI. The corresponding kernel is the matrix of either 3x3 or 5x5 size with the following values:

				1	4	6	4	1
1	2	1		0	0	0	0	0
-2	-4	-2	or	-2	-8	-12	-8	-2
1	2	1		0	0	0	0	0
				1	4	6	4	1

The anchor cell is the center cell of the kernel (red). The size of the source image ROI is equal to *dstRoiSize*, the size of the destination image ROI.

This filter has the effect of both enhancing and smoothing horizontal edges of an image.

To ensure valid operation when image boundary pixels are processed, the application should correctly define additional border pixels (see [Borders](#)).

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pSrc</i> or <i>pDst</i> is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>dstRoiSize</i> has a field with a zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> or <i>dstStep</i> has a zero or negative value.
<code>ippStsMaskSizeErr</code>	Indicates an error condition if <i>mask</i> has an illegal value.

## FilterSobelVertSecond

*Filters an image using a second derivative vertical Sobel operator.*

---

## Syntax

```
ippiStatus ippiFilterSobelVertSecond_<mod>(const Ipp<srcDatatype>* pSrc, int srcStep,
Ipp<dstDatatype>* pDst, int dstStep, IppiSize dstRoiSize, IppiMaskSize mask);
```

Supported values for *mod*:

8u16s\_C1R      8s16s\_C1R      32f\_C1R

## Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>dstRoiSize</i>	Size of the source and destination ROI in pixels.
<i>mask</i>	Predefined mask of <code>IppiMaskSize</code> type.

## Description

The function `ippiFilterSobelVertSecond` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)). This function applies a second derivative vertical Sobel operator to an image ROI. The corresponding kernel is the matrix of either 3x3 or 5x5 size with the following values:

$$\begin{array}{ccccc}
 & & 1 & 0 & -2 & 0 & 1 \\
 1 & -2 & 1 & & & & \\
 2 & -4 & 2 & \text{or} & 6 & 0 & -12 & 0 & 6 \\
 1 & -2 & 1 & & 4 & 0 & -8 & 0 & 4 \\
 & & & & 1 & 0 & -2 & 0 & 1
 \end{array}$$

The anchor cell is the center cell of the kernel (red). The size of the source image ROI is equal to *dstRoiSize*, the size of the destination image ROI.

This filter has the effect of both enhancing and smoothing vertical edges of an image.

To ensure valid operation when image boundary pixels are processed, the application should correctly define additional border pixels (see [Borders](#)).

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pSrc</i> or <i>pDst</i> is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>dstRoiSize</i> has a field with a zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> or <i>dstStep</i> has a zero or negative value.
<code>ippStsMaskSizeErr</code>	Indicates an error condition if <i>mask</i> has an illegal value.

## Example

The code example below shows how to use the function `ippiFilterSobelVertSecond_8u16s_C1R`.

```

Ipp8u pSrc[8*8]={
    4, 4, 4, 4, 4, 4, 4, 4,
    4, 3, 3, 3, 3, 3, 3, 4,
    4, 3, 2, 2, 2, 2, 3, 4,
    4, 3, 2, 1, 1, 2, 3, 4,
    4, 3, 2, 1, 1, 2, 3, 4,

```

```

        4, 3, 2, 2, 2, 2, 3, 4,
        4, 3, 3, 3, 3, 3, 3, 4,
        4, 4, 4, 4, 4, 4, 4, 4
    };
    Ipp16s pDst[8*8];
    IppiSize Roi = { 8, 8 };

   ippiFilterSobelVertSecond_8u16s_C1R ( pSrc, 8, pDst, 8*sizeof( Ipp16s ),
    Roi, ippMskSize3x3 );

```

**Result:**

```

4 4 4 4 4 4 4 4
4 3 3 3 3 3 3 4
4 3 2 2 2 2 3 4
4 3 2 1 1 2 3 4
4 3 2 1 1 2 3 4      pSrc
4 3 2 2 2 2 3 4
4 3 3 3 3 3 3 4
4 4 4 4 4 4 4 4

```

```

399 1 0  0 0  0  1 -201
197 2 1  0 0  1  2  -3
 -4 1 2  1 1  2  1  -4
 -4 0 1  3 3  1  0  -4
 -4 0 1  3 3  1  0  -4
 -4 1 2  1 1  2  1  -4
 -3 2 1  0 0  1  2  197
-201 1 0 -12 75 -300 220 593

```

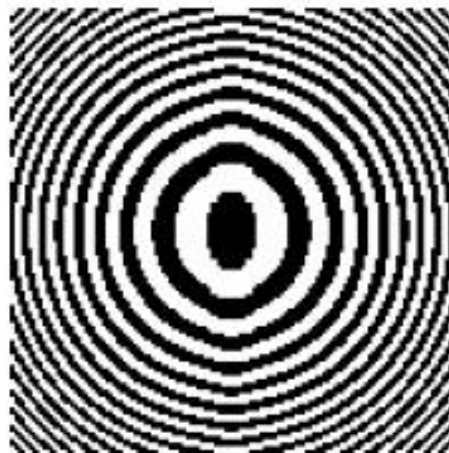
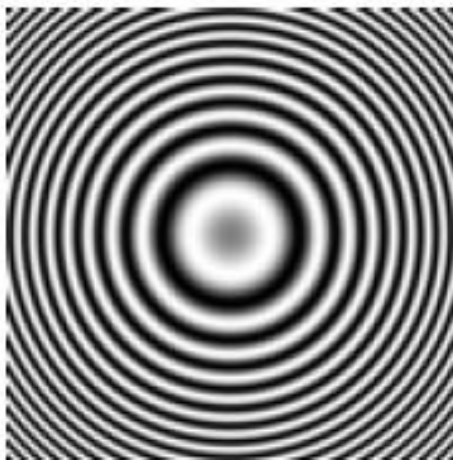
pDst

for example, on real picture it can looks like this

```

size 128x128      size 126x126
roi  126x126      roi  126x126
                      ippMskSize3x3

```



## FilterSobelCross

*Filters an image using a second cross derivative Sobel operator.*

### Syntax

```
IppStatus ippiFilterSobelCross_<mod>(const Ipp<srcDatatype>* pSrc, int srcStep,
Ipp<dstDatatype>* pDst, int dstStep, IppiSize dstRoiSize, IppiMaskSize mask);
```

Supported values for mod:

8u16s\_C1R      8s16s\_C1R      32f\_C1R

### Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>dstRoiSize</i>	Size of the source and destination ROI in pixels.
<i>mask</i>	Predefined mask of IppiMaskSize type.

### Description

The function `ippiFilterSobelCross` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)). This function applies a second cross derivative Sobel operator to an image ROI. The corresponding kernel is the matrix of either 3x3 or 5x5 size with the following values:

				-1	-2	0	2	1
-1	0	1		-2	-4	0	4	0
0	0	0	or	0	0	0	0	0
1	0	-1		2	4	0	-4	-2
				1	2	0	-2	-1

The anchor cell is the center cell of the kernel (red). The size of the source image ROI is equal to *dstRoiSize*, the size of the destination image ROI.

This filter has the effect of both enhancing and smoothing vertical edges of an image.

To ensure valid operation when image boundary pixels are processed, the application should correctly define additional border pixels (see [Borders](#)).

### Return Values

`ippStsNoErr`      Indicates no error. Any other value indicates an error or a warning.

<code>ippStsNullPtrErr</code>	Indicates an error condition if <code>pSrc</code> or <code>pDst</code> is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <code>dstRoiSize</code> has a field with a zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <code>srcStep</code> or <code>dstStep</code> has a zero or negative value.
<code>ippStsMaskSizeErr</code>	Indicates an error condition if <code>mask</code> has an illegal value.

## FilterRobertsDown

*Filters an image using a horizontal Roberts kernel.*

---

### Syntax

```
ippStatus ippiFilterRobertsDown_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize dstRoiSize);
```

Supported values for mod:

8u_C1R	16s_C1R	32f_C1R
8u_C3R	16s_C3R	32f_C3R
8u_AC4R	16s_AC4R	32f_AC4R

### Parameters

<code>pSrc</code>	Pointer to the source image ROI.
<code>srcStep</code>	Distance in bytes between starts of consecutive lines in the source image.
<code>pDst</code>	Pointer to the destination image ROI.
<code>dstStep</code>	Distance in bytes between starts of consecutive lines in the destination image.
<code>dstRoiSize</code>	Size of the source and destination ROI in pixels.

### Description

The function `ippiFilterRobertsDown` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)). This function applies a horizontal Roberts operator to an image ROI. The corresponding kernel is the matrix of 3x3 size with the following values:

```
0  0  0
0  1  0
0  0 -1
```

The anchor cell is the center cell of the kernel (red). The size of the source image ROI is equal to `dstRoiSize`, the size of the destination image ROI.

This filter gives the gross approximation of the pixel values' gradient in the horizontal direction.

To ensure valid operation when image boundary pixels are processed, the application should correctly define additional border pixels (see [Borders](#)).



## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <code>pSrc</code> or <code>pDst</code> is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <code>dstRoiSize</code> has a field with a zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <code>srcStep</code> or <code>dstStep</code> has a zero or negative value.

## FilterRobertsUp

*Filters an image using a vertical Roberts kernel.*

### Syntax

```
IppStatus ippiFilterRobertsUp_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize dstRoiSize);
```

Supported values for `mod`:

<code>8u_C1R</code>	<code>16s_C1R</code>	<code>32f_C1R</code>
<code>8u_C3R</code>	<code>16s_C3R</code>	<code>32f_C3R</code>
<code>8u_AC4R</code>	<code>16s_AC4R</code>	<code>32f_AC4R</code>

### Parameters

<code>pSrc</code>	Pointer to the source image ROI.
<code>srcStep</code>	Distance in bytes between starts of consecutive lines in the source image.
<code>pDst</code>	Pointer to the destination image ROI.
<code>dstStep</code>	Distance in bytes between starts of consecutive lines in the destination image.
<code>dstRoiSize</code>	Size of the source and destination ROI in pixels.

### Description

The function `ippiFilterRobertsUp` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)). This function applies a vertical Roberts operator to an image ROI. The corresponding kernel is the matrix of 3x3 size with the following values:

```
0  0  0
0  1  0
-1 0  0
```

The anchor cell is the center cell of the kernel (red). The size of the source image ROI is equal to `dstRoiSize`, the size of the destination image ROI. This filter gives the gross approximation of the pixel values' gradient in the vertical direction.

To ensure valid operation when image boundary pixels are processed, the application should correctly define additional border pixels (see [Borders](#)).

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <code>pSrc</code> or <code>pDst</code> is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <code>dstRoiSize</code> has a field with a zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <code>srcStep</code> or <code>dstStep</code> has a zero or negative value.

## FilterLaplace

*Filters an image using a Laplacian kernel.*

---

### Syntax

```
IppStatus ippFilterLaplace_<mod>(const Ipp<srcDatatype>* pSrc, int srcStep,
Ipp<dstDatatype>* pDst, int dstStep, IppiSize dstRoiSize, IppiMaskSize mask);
```

Supported values for `mod`:

<code>8u_C1R</code>	<code>16s_C1R</code>	<code>32f_C1R</code>	<code>8u16s_C1R</code> <code>8s16s_C1R</code>
<code>8u_C3R</code>	<code>16s_C3R</code>	<code>32f_C3R</code>	
<code>8u_C4R</code>	<code>16s_C4R</code>	<code>32f_C4R</code>	
<code>8u_AC4R</code>	<code>16s_AC4R</code>	<code>32f_AC4R</code>	

### Parameters

<code>pSrc</code>	Pointer to the source image ROI.
<code>srcStep</code>	Distance in bytes between starts of consecutive lines in the source image.
<code>pDst</code>	Pointer to the destination image ROI.
<code>dstStep</code>	Distance in bytes between starts of consecutive lines in the destination image.
<code>dstRoiSize</code>	Size of the source and destination ROI in pixels.
<code>mask</code>	Predefined mask of <code>IppiMaskSize</code> type.

## Description

The function `ippiFilterLaplace` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)). This function applies a highpass Laplacian filter to an image ROI. The corresponding kernel is the matrix of either 3x3 or 5x5 size with the following values:

			-1	-3	-4	-3	-1
-1	-1	-1		-3	0	6	0
-1	8	-1	or	-4	6	20	6
-1	-1	-1		-3	0	6	0
				-1	-3	-4	-3

The anchor cell is the center cell of the kernel (red). The size of the source image ROI is equal to `dstRoiSize`, the size of the destination image ROI.

This filter helps locate zero crossings in an image.

To ensure valid operation when image boundary pixels are processed, the application should correctly define additional border pixels (see [Borders](#)).

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <code>pSrc</code> or <code>pDst</code> is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <code>dstRoiSize</code> has a field with a zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <code>srcStep</code> or <code>dstStep</code> has a zero or negative value.
<code>ippStsMaskSizeErr</code>	Indicates an error condition if <code>mask</code> has an illegal value.

## FilterGauss

*Filters an image using a Gaussian kernel.*

## Syntax

```
ippStatus ippiFilterGauss_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize dstRoiSize, IppiMaskSize mask);
```

Supported values for `mod`:

8u_C1R	16u_C1R	16s_C1R	32f_C1R
8u_C3R	16u_C3R	16s_C3R	32f_C3R
8u_C4R	16u_C4R	16s_C4R	32f_C4R
8u_AC4R	16u_AC4R	16s_AC4R	32f_AC4R

## Parameters

`pSrc` Pointer to the source image ROI.

<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>dstRoiSize</i>	Size of the source and destination ROI in pixels.
<i>mask</i>	Predefined mask of <code>IppiMaskSize</code> type.

## Description

The function `ippiFilterGauss` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)). This function applies a lowpass Gaussian filter to an image ROI. The corresponding kernel is the matrix of either 3x3 or 5x5 size.

The 3x3 filter uses the kernel:

```
1/16  2/16  1/16
2/16  4/16  2/16
1/16  2/16  1/16
```

These filter coefficients correspond to a 2-dimensional Gaussian distribution with standard deviation 0.85.

The 5x5 filter uses the kernel:

```
2/571  7/571 12/571  7/571  2/571
7/571 31/571 52/571 31/571  7/571
12/571 52/571 127/571 52/571 12/571
7/571 31/571 52/571 31/571  7/571
2/571  7/571 12/571  7/571  2/571
```

These filter coefficients correspond to a 2-dimensional Gaussian distribution with standard deviation 1.0.

The anchor cell is the center cell of the kernels (red). The size of the source image ROI is equal to *dstRoiSize*, the size of the destination image ROI.

To ensure valid operation when image boundary pixels are processed, the application should correctly define additional border pixels (see [Borders](#)).

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pSrc</i> or <i>pDst</i> is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>dstRoiSize</i> has a field with a zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> or <i>dstStep</i> has a zero or negative value.
<code>ippStsMaskSizeErr</code>	Indicates an error condition if <i>mask</i> has an illegal value.

## Example

The code example below shows how to use the function `ippiFilterGauss_8u_C1R`.

```

Ipp8u src[9*9] = {
    0, 1, 2, 3, 4, 5, 6, 7, 0,
    1, 2, 3, 4, 5, 6, 7, 0, 1,
    2, 3, 4, 5, 6, 7, 0, 1, 2,
    3, 4, 5, 6, 7, 0, 1, 2, 3,
    4, 5, 6, 7, 0, 1, 2, 3, 4,
    5, 6, 7, 0, 1, 2, 3, 4, 5,
    6, 7, 0, 1, 2, 3, 4, 5, 6,
    7, 0, 1, 2, 3, 4, 5, 6, 7,
    0, 1, 2, 3, 4, 5, 6, 7, 0
};
Ipp8u dst[9*9];
IppiSize roiSize = { 9, 9 };

ippiFilterGauss_8u_C1R ( src, 9, dst, 9, roiSize, ippMskSize5x5 );

```

Result:

```

0 1 2 3 4 5 6 7 0
1 2 3 4 5 6 7 0 1
2 3 4 5 6 7 0 1 2
3 4 5 6 7 0 1 2 3      src
4 5 6 7 0 1 2 3 4
5 6 7 0 1 2 3 4 5
6 7 0 1 2 3 4 5 6
7 0 1 2 3 4 5 6 7
0 1 2 3 4 5 6 7 0

80 62 58 59 60 60 60 57 44
26 15 14 15 15 16 15 12 09
05 04 04 05 05 04 03 02 02
03 04 05 05 04 03 02 02 03
04 05 05 04 03 02 02 03 04      dst
05 05 04 03 02 02 03 04 05
05 04 03 02 02 03 04 05 08
12 13 13 13 13 14 15 18 26
39 48 53 54 52 50 50 53 63

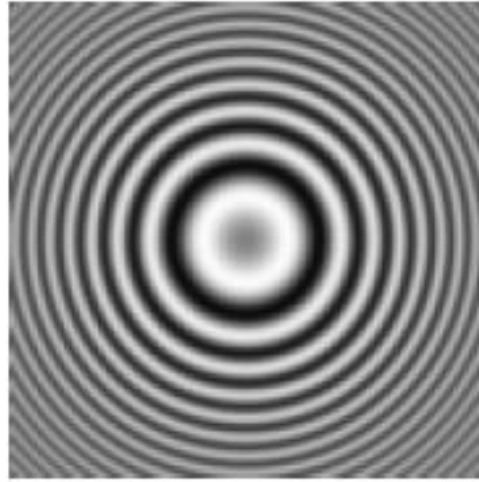
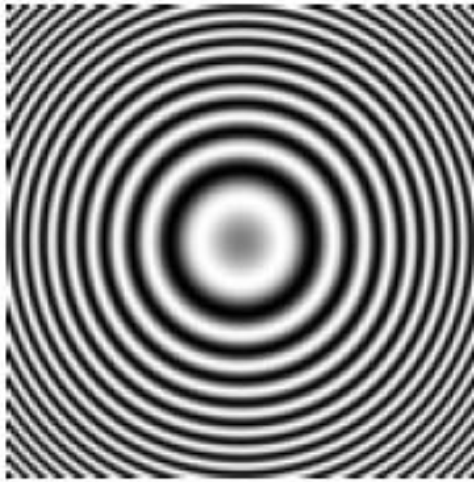
```

for example, on real picture it can look like this

```

size 128x128      size 124x124
roi 124x124      roi 124x124
                  ippMskSize5x5

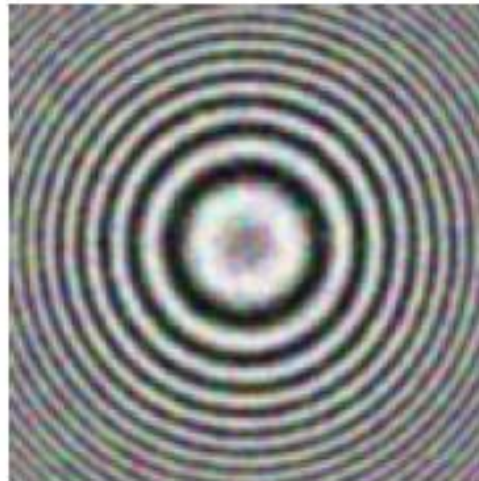
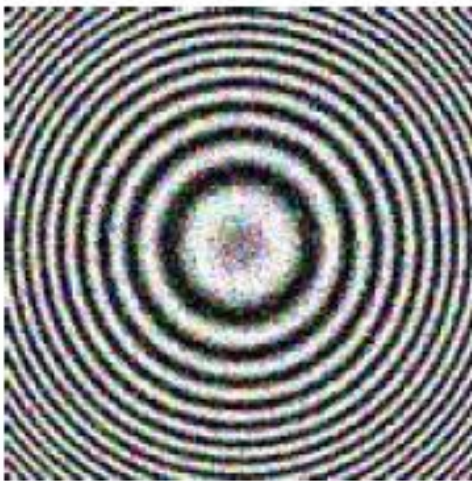
```



```
+ ippiAddRandGauss_Direct_8u_C3IR (.., 2, 50, 57328065)
```

```
size 128x128  
roi 124x124
```

```
size 124x124  
roi 124x124  
ippMskSize5x5
```



## FilterHipass

*Filters an image using a highpass filter.*

### Syntax

```
IppStatus ippiFilterHipass_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize dstRoiSize, IppiMaskSize mask);
```

Supported values for mod:

8u_C1R	16s_C1R	32f_C1R
8u_C3R	16s_C3R	32f_C3R
8u_C4R	16s_C4R	32f_C4R
8u_AC4R	16s_AC4R	32f_AC4R

### Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>dstRoiSize</i>	Size of the source and destination ROI in pixels.
<i>mask</i>	Predefined mask of IppiMaskSize type.

### Description

The function `ippiFilterHipass` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)). This function applies a highpass filter to an image ROI. The corresponding kernel is the matrix of either 3x3 or 5x5 size with the following values:

		-1	-1	-1	-1	-1
-1	-1	-1		-1	-1	-1
-1	8	-1	or	-1	-1	24
-1	-1	-1		-1	-1	-1
		-1		-1	-1	-1

The anchor cell is the center cell of the kernel (red). The size of the source image ROI is equal to *dstRoiSize*, the size of the destination image ROI.

This filter attenuates low-frequency components and thus sharpens an image.

To ensure valid operation when image boundary pixels are processed, the application should correctly define additional border pixels (see [Borders](#)).

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
--------------------------	--

<code>ippStsNullPtrErr</code>	Indicates an error condition if <code>pSrc</code> or <code>pDst</code> is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <code>dstRoiSize</code> has a field with a zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <code>srcStep</code> or <code>dstStep</code> has a zero or negative value.
<code>ippStsMaskSizeErr</code>	Indicates an error condition if <code>mask</code> has an illegal value.

## FilterLowpass

*Filters an image using a lowpass filter.*

---

### Syntax

```
IppStatus ippiFilterLowpass_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize dstRoiSize, IppiMaskSize mask);
```

Supported values for mod:

8u_C1R	16u_C1R	16s_C1R	32f_C1R
8u_C3R	16u_C3R	16s_C3R	32f_C3R
8u_AC4R	16u_AC4R	16s_AC4R	32f_AC4R

### Parameters

<code>pSrc</code>	Pointer to the source image ROI.
<code>srcStep</code>	Distance in bytes between starts of consecutive lines in the source image.
<code>pDst</code>	Pointer to the destination image ROI.
<code>dstStep</code>	Distance in bytes between starts of consecutive lines in the destination image.
<code>dstRoiSize</code>	Size of the source and destination ROI in pixels.
<code>mask</code>	Predefined mask of <code>IppiMaskSize</code> type.

### Description

The function `ippiFilterLowpass` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)). This function applies a lowpass filter to an image ROI. The corresponding kernel is the matrix of either 3x3 or 5x5 size.

The 3x3 filter uses the following kernel:

```
1/9  1/9  1/9
1/9  1/9  1/9
1/9  1/9  1/9
```



The 5x5 filter uses the following kernel:

```

1/25  1/25  1/25  1/25  1/25
1/25  1/25  1/25  1/25  1/25
1/25  1/25  1/25  1/25  1/25
1/25  1/25  1/25  1/25  1/25
1/25  1/25  1/25  1/25  1/25

```

The anchor cell is the center cell of the kernel (red). The size of the source image ROI is equal to *dstRoiSize*, the size of the destination image ROI.

This filter blurs an image by averaging the pixel values over some neighborhood.

To ensure valid operation when image boundary pixels are processed, the application should correctly define additional border pixels (see [Borders](#)).

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pSrc</i> or <i>pDst</i> is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>dstRoiSize</i> has a field with a zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> or <i>dstStep</i> has a zero or negative value.
<code>ippStsMaskSizeErr</code>	Indicates an error condition if <i>mask</i> has an illegal value.

## FilterSharpen

*Filters an image using a sharpening filter.*

### Syntax

```
IppStatus ippiFilterSharpen_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize dstRoiSize);
```

Supported values for *mod*:

8u_C1R	16s_C1R	32f_C1R
8u_C3R	16s_C3R	32f_C3R
8u_C4R	16s_C4R	32f_C4R
8u_AC4R	16s_AC4R	32f_AC4R

### Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.

<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>dstRoiSize</i>	Size of the source and destination ROI in pixels.

## Description

The function `ippiFilterSharpen` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)). This function applies a sharpening filter to an image ROI. The corresponding kernel is the matrix of 3x3 size with the following values:

```
-1/8  -1/8  -1/8
-1/8  16/8  -1/8
-1/8  -1/8  -1/8
```

The anchor cell is the center cell of the kernel (red). The size of the source image ROI is equal to *dstRoiSize*, the size of the destination image ROI. This filter enhances high-frequency components and thus sharpens an image.

To ensure valid operation when image boundary pixels are processed, the application should correctly define additional border pixels (see [Borders](#)).

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pSrc</i> or <i>pDst</i> is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>dstRoiSize</i> has a field with a zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> or <i>dstStep</i> has a zero or negative value.

## Fixed Filters with Border

This section describes the fixed filter functions that perform linear filtering of a source image using one of the predefined convolution kernels like the filter functions from the previous section do. Difference is that these functions automatically create a required border and define appropriate pixel values.

### FilterScharrHorizGetBufferSize

*Computes the size of the external buffer for the horizontal Scharr filter with border.*

---

## Syntax

```
IppStatus ippiFilterScharrHorizGetBufferSize_<mod>(IppiSize roiSize, int* pBufferSize);
```

Supported values for *mod*:

```
8u8s_C1R      8u16s_C1R      32f_C1R
```

## Parameters

<code>roiSize</code>	Maximum size of the source and destination image ROI.
<code>pBufferSize</code>	Pointer to the buffer size.

## Description

The function `ippiFilterScharrHorizGetBufferSize` is declared in the `ippcv.h` file. This function computes the size of the external buffer that is required for the function `ippiFilterScharrHorizBorder`. This buffer `pBufferSize[0]` can be used to filter an image whose width and height are equal to or less than corresponding fields of `roiSize`.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <code>pBufferSize</code> is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <code>roiSize</code> has a field with a zero or negative value.

## FilterScharrVertGetBufferSize

*Computes the size of the external buffer for the vertical Scharr filter with border.*

## Syntax

```
IppStatus ippiFilterScharrVertGetBufferSize__<mod>(IppiSize roiSize, int* pBufferSize);
```

Supported values for `mod`:

`8u8s_C1R`      `8u16s_C1R`      `32f_C1R`

## Parameters

<code>roiSize</code>	Maximum size of the source and destination image ROI.
<code>pBufferSize</code>	Pointer to the buffer size.

## Description

The function `ippiFilterScharrVertGetBufferSize` is declared in the `ippcv.h` file. This function computes the size of the external buffer that is required for the function `ippiFilterScharrVertBorder`. This buffer `pBufferSize[0]` can be used to filter an image whose width and height are equal to or less than corresponding fields of `roiSize`.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <code>pBufferSize</code> is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <code>roiSize</code> has a field with a zero or negative value.

## FilterSobelHorizGetBufferSize

*Computes the size of the external buffer for the horizontal Sobel filter with border.*

---

### Syntax

```
IppStatus ippiFilterSobelHorizGetBufferSize_<mod>(IppiSize roiSize, IppiMaskSize mask,
int* pBufferSize);
```

Supported values for mod:

8u8s\_C1R          8u16s\_C1R          32f\_C1R

### Parameters

<i>roiSize</i>	Maximum size of the source and destination image ROI.
<i>mask</i>	Predefined mask of <code>IppiMaskSize</code> type.
<i>pBufferSize</i>	Pointer to the buffer size.

### Description

The function `ippiFilterSobelHorizGetBufferSize` is declared in the `ippcv.h` file. This function computes the size of the external buffer that is required for the filter function `ippiFilterSobelHorizBorder`. The kernel of the filter is the matrix of either 3x3 or 5x5 size that is specified by the parameter *mask* (see [Table "Types of the Fixed Filter Functions"](#)). This buffer `pBufferSize[0]` can be used to filter an image whose width and height are equal to or less than corresponding fields of *roiSize*.

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pBufferSize</i> is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a zero or negative value.
<code>ippStsMaskSizeErr</code>	Indicates an error condition if <i>mask</i> has a wrong value.

## FilterSobelVertGetBufferSize, FilterSobelNegVertGetBufferSize

*Computes the size of the external buffer for the vertical Sobel filter with border.*

---

### Syntax

```
IppStatus ippiFilterSobelVertGetBufferSize_<mod>(IppiSize roiSize, IppiMaskSize mask, int*
pBufferSize);
```

```
IppStatus ippiFilterSobelNegVertGetBufferSize_<mod>(IppiSize roiSize, IppiMaskSize mask,
int* pBufferSize);
```

Supported values for mod:

8u8s\_C1R      8u16s\_C1R      32f\_C1R

## Parameters

<i>roiSize</i>	Maximum size of the source and destination image ROI.
<i>mask</i>	Predefined mask of <code>IppiMaskSize</code> type.
<i>pBufferSize</i>	Pointer to the buffer size.

## Description

The functions `ippiFilterSobelVertGetBufferSize` and `ippiFilterSobelNegVertGetBufferSize` are declared in the `ippcv.h` file. These functions compute the size of the external buffer that is required for the filter functions `ippiFilterSobelVertBorder`, and `ippiFilterSobelNegVertBorder` respectively. The kernel of the filter is the matrix of either 3x3 or 5x5 size that is specified by the parameter *mask* (see [Table “Types of the Fixed Filter Functions”](#)). This buffer *pBufferSize[0]* can be used to filter an image whose width and height are equal to or less than corresponding fields of *roiSize*.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pBufferSize</i> is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with a zero or negative value.
<code>ippStsMaskSizeErr</code>	Indicates an error condition if <i>mask</i> has a wrong value.

## FilterSobelHorizSecondGetBufferSize

*Computes the size of the external buffer for the second derivative horizontal Sobel filter with border.*

## Syntax

```
IppStatus ippiFilterSobelHorizSecondGetBufferSize_<mod>(IppiSize roiSize, IppiMaskSize
mask, int* pBufferSize);
```

Supported values for mod:

8u8s\_C1R      8u16s\_C1R      32f\_C1R

## Parameters

<i>roiSize</i>	Maximum size of the source and destination image ROI.
<i>mask</i>	Predefined mask of <code>IppiMaskSize</code> type.
<i>pBufferSize</i>	Pointer to the buffer size.

## Description

The function `ippiFilterSobelHorizSecondGetBufferSize` is declared in the `ippcv.h` file. This function computes the size of the external buffer that is required for the filter function `ippiFilterSobelHorizSecondBorder`. The kernel of the filter is the matrix of either 3x3 or 5x5 size that is specified by the parameter `mask` (see Table “Types of the Fixed Filter Functions”). This buffer `pBufferSize[0]` can be used to filter an image whose width and height are equal to or less than corresponding fields of `roiSize`.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if the <code>pBufferSize</code> pointer is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <code>roiSize</code> has a field with a zero or negative value.
<code>ippStsMaskSizeErr</code>	Indicates an error condition if <code>mask</code> has a wrong value.

## FilterSobelVertSecondGetBufferSize

*Computes the size of the external buffer for the second derivative vertical Sobel filter with border.*

---

## Syntax

```
IppStatus ippiFilterSobelVertSecondGetBufferSize_<mod>(IppiSize roiSize, IppiMaskSize mask, int* pBufferSize);
```

Supported values for `mod`:

`8u8s_C1R`      `8u16s_C1R`      `32f_C1R`

## Parameters

<code>roiSize</code>	Maximum size of the source and destination image ROI.
<code>mask</code>	Predefined mask of <code>IppiMaskSize</code> type.
<code>pBufferSize</code>	Pointer to the buffer size.

## Description

The function `ippiFilterSobelVertSecondGetBufferSize` is declared in the `ippcv.h` file. This function computes the size of the external buffer that is required for the filter function `ippiFilterSobelVertSecondBorder`. The kernel of the filter is the matrix of either 3x3 or 5x5 size that is specified by the parameter `mask` (see Table “Types of the Fixed Filter Functions”). This buffer `pBufferSize[0]` can be used to filter an image whose width and height are equal to or less than corresponding fields of `roiSize`.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if the <code>pBufferSize</code> pointer is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <code>roiSize</code> has a field with a zero or negative value.

`ippStsMaskSizeErr` Indicates an error condition if *mask* has a wrong value.

## FilterSobelCrossGetBufferSize

*Computes the size of the external buffer for the cross Sobel filter with border.*

---

### Syntax

```
IppStatus ippFilterSobelCrossGetBufferSize_<mod>(IppiSize roiSize, IppiMaskSize mask,
int* pBufferSize);
```

Supported values for mod:

8u8s\_C1R      8u16s\_C1R      32f\_C1R

### Parameters

<i>roiSize</i>	Maximum size of the source and destination image ROI.
<i>mask</i>	Predefined mask of <code>IppiMaskSize</code> type.
<i>pBufferSize</i>	Pointer to the buffer size.

### Description

The function `ippFilterSobelCrossGetBufferSize` is declared in the `ippcv.h` file. This function computes the size of the external buffer that is required for the filter function `ippFilterSobelCrossBorder`. The kernel of the filter is the matrix of either 3x3 or 5x5 size that is specified by the parameter *mask* (see [Table "Types of the Fixed Filter Functions"](#)). This buffer `pBufferSize[0]` can be used to filter an image whose width and height are equal to or less than corresponding fields of *roiSize*.

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if the <i>pBufferSize</i> pointer is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with a zero or negative value.
<code>ippStsMaskSizeErr</code>	Indicates an error condition if <i>mask</i> has a wrong value.

## FilterLaplacianGetBufferSize

*Computes the size of the external buffer for the Laplace filter with border.*

---

### Syntax

```
IppStatus ippFilterLaplacianGetBufferSize_<mod>(IppiSize roiSize, IppiMaskSize mask, int*
pBufferSize);
```

Supported values for mod:

8u8s\_C1R      8u16s\_C1R      32f\_C1R

## Parameters

<i>roiSize</i>	Maximum size of the source and destination image ROI.
<i>mask</i>	Predefined mask of <code>IppiMaskSize</code> type.
<i>pBufferSize</i>	Pointer to the buffer size.

## Description

The function `ippiFilterLaplacianGetBufferSize` is declared in the `ippcv.h` file. This function computes the size of the external buffer that is required for the filter function `ippiFilterLaplacianBorder`. The kernel of the filter is the matrix of either 3x3 or 5x5 size that is specified by the parameter *mask* (see [Table "Types of the Fixed Filter Functions"](#)). This buffer *pBufferSize[0]* can be used to filter an image whose width and height are equal to or less than corresponding fields of *roiSize*.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if the <i>pBufferSize</i> pointer is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with a zero or negative value.
<code>ippStsMaskSizeErr</code>	Indicates an error condition if <i>mask</i> has a wrong value.

## FilterGaussGetBufferSize

*Computes the size of the external buffer for the Gaussian filter with border.*

---

## Syntax

```
IppStatus ippiFilterGaussGetBufferSize_32f_C1R(IppiSize roiSize, int kernelSize, int* pBufferSize);
```

## Parameters

<i>roiSize</i>	Maximum size of the source and destination image ROI.
<i>kernelSize</i>	Size of the Gaussian kernel, odd, greater than or equal to 3.
<i>pBufferSize</i>	Pointer to the buffer size.

## Description

The function `ippiFilterGaussGetBufferSize` is declared in the `ippcv.h` file. This function computes the size of the external buffer that is required for the filter function `ippiFilterGaussBorder`. This buffer with the length *pBufferSize[0]* can be used to filter an image whose width and height are equal to or less than corresponding fields of *roiSize*, and/or the kernel size is equal to or less than *kernelSize*.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if the <i>pBufferSize</i> is <code>NULL</code> .



<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with a zero or negative value.
<code>ippStsBadArgErr</code>	Indicates an error condition if <i>kernelSize</i> is even or less than 3.

## FilterLowpassGetBufferSize

*Computes the size of the external buffer for the lowpass filter with border.*

---

### Syntax

```
IppStatus ippFilterLowpassGetBufferSize_8u_C1R(IppiSize roiSize, IppiMaskSize mask, int* pBufferSize);
```

```
IppStatus ippFilterLowpassGetBufferSize_32f_C1R(IppiSize roiSize, IppiMaskSize mask, int* pBufferSize);
```

### Parameters

<i>roiSize</i>	Maximum size of the source and destination image ROI.
<i>mask</i>	Predefined mask of <code>IppiMaskSize</code> type.
<i>pBufferSize</i>	Pointer to the buffer size.

### Description

The function `ippFilterLowpassGetBufferSize` is declared in the `ippcv.h` file. This function computes the size of the external buffer that is required for the filter function `ippFilterLowpassBorder`. The kernel of the filter is the matrix of either 3x3 or 5x5 size that is specified by the parameter *mask* (see [Table "Types of the Fixed Filter Functions"](#)). This buffer `pBufferSize[0]` can be used to filter an image whose width and height are equal to or less than corresponding fields of *roiSize*.

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if the <i>pBufferSize</i> pointer is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with a zero or negative value.
<code>ippStsMaskSizeErr</code>	Indicates an error condition if <i>mask</i> has a wrong value.

## GenSobelKernel

*Computes kernel for the Sobel filter.*

---

### Syntax

```
IppStatus ippGenSobelKernel_16s(Ipp16s* pDst, int kernelSize, int dx, int sign);
```

```
IppStatus ippGenSobelKernel_32f(Ipp132f* pDst, int kernelSize, int dx, int sign);
```

## Parameters

<i>pDst</i>	Pointer to the destination vector.
<i>kernelSize</i>	Size of the Sobel kernel.
<i>dx</i>	Order of derivative.
<i>sign</i>	Specifies signs of kernel elements.

## Description

The function `ippiGenSobelKernel` is declared in the `ippcv.h` file. This function computes the one-dimensional Sobel kernel. Kernel coefficients are equal to coefficients of the polynomial

$$(1 + x)^{kernelSize - dx - 1} \cdot (x - 1)^{dx}$$

If the *sign* parameter is negative, then signs of kernel coefficients are changed. Kernel calculated by this function can be used to filter images by a high order Sobel filter.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if the <i>pDst</i> pointer is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <i>kernelSize</i> is less than 3 or is even.
<code>ippStsBadArgErr</code>	Indicates an error condition if <i>dx</i> is equal to or less than <i>kernelSize</i> , or <i>dx</i> is negative.

## FilterScharrHorizBorder

*Applies horizontal Scharr filter with border.*

---

### Syntax

```
ippStatus ippiFilterScharrHorizBorder_8u8s_C1R(const Ipp8u* pSrc, int srcStep, Ipp8s* pDst, int dstStep, IppiSize roiSize, IppiBorderType borderType, Ipp8u borderValue, int divisor, Ipp8u* pBuffer);
```

```
ippStatus ippiFilterScharrHorizBorder_8u16s_C1R(const Ipp8u* pSrc, int srcStep, Ipp16s* pDst, int dstStep, IppiSize roiSize, IppiBorderType borderType, Ipp8u borderValue, Ipp8u* pBuffer);
```

```
ippStatus ippiFilterScharrHorizBorder_32f_C1R(const Ipp32f* pSrc, int srcStep, Ipp32f* pDst, int dstStep, IppiSize roiSize, IppiBorderType borderType, Ipp32f borderValue, Ipp8u* pBuffer);
```

## Parameters

<i>pSrc</i>	Pointer to the source image ROI.
-------------	----------------------------------

<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.										
<i>pDst</i>	Pointer to the destination image ROI.										
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.										
<i>roiSize</i>	Size of the source and destination image ROI.										
<i>borderType</i>	Type of border (see <a href="#">Borders</a> ); the following values are possible: <table> <tr> <td><code>ippBorderConst</code></td><td>Values of all border pixels are set to constant.</td></tr> <tr> <td><code>ippBorderRepl</code></td><td>Replicated border is used.</td></tr> <tr> <td><code>ippBorderWrap</code></td><td>Wrapped border is used</td></tr> <tr> <td><code>ippBorderMirror</code></td><td>Mirrored border is used</td></tr> <tr> <td><code>ippBorderMirrorR</code></td><td>Mirrored border with replication is used</td></tr> </table>	<code>ippBorderConst</code>	Values of all border pixels are set to constant.	<code>ippBorderRepl</code>	Replicated border is used.	<code>ippBorderWrap</code>	Wrapped border is used	<code>ippBorderMirror</code>	Mirrored border is used	<code>ippBorderMirrorR</code>	Mirrored border with replication is used
<code>ippBorderConst</code>	Values of all border pixels are set to constant.										
<code>ippBorderRepl</code>	Replicated border is used.										
<code>ippBorderWrap</code>	Wrapped border is used										
<code>ippBorderMirror</code>	Mirrored border is used										
<code>ippBorderMirrorR</code>	Mirrored border with replication is used										
<i>borderValue</i>	The constant value to assign to the pixels in the constant border (not applicable for other border's type).										
<i>divisor</i>	The constant value that output pixel value is divided by.										
<i>pBuffer</i>	Pointer to the working buffer.										

## Description

The function `ippiFilterScharrHorizBorder` is declared in the `ippcv.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)). This function applies the horizontal Scharr filter ( $y$ -derivative) to the source image *pSrc* and stores results to the destination image of the same size *pDst*. For the function flavor `ippiFilterScharrHorizBorder_32f_C1R` the source image can be used as the destination image.

The values of border pixels are assigned in accordance with the *borderType* and *borderValue* parameters. The kernel of this filter is a matrix of 3x3 size with the anchor in the center cell (red) and the following values:

3	10	3
0	0	0
-3	-10	-3

The function requires the working buffer *pBuffer* whose size should be computed by the function `ippiFilterScharrHorizGetBufferSize` beforehand.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with a zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> or <i>dstStep</i> is less than $roiSize.width * <pixelSize>$
<code>ippStsNotEvenStepErr</code>	Indicates an error condition if one of the step values is not divisible by 4 for floating-point images, or by 2 for short-integer images.

`ippStsBadArgErr`Indicates an error if *borderType* or *divisor* has a wrong value.

## FilterScharrVertBorder

*Applies vertical Scharr filter with border.*

---

### Syntax

```
IppStatus ippFilterScharrVertBorder_8u8s_C1R(const Ipp8u* pSrc, int srcStep, Ipp8s* pDst,
int dstStep, IppiSize roiSize, IppiBorderType borderType, Ipp8u borderValue, int divisor,
Ipp8u* pBuffer);
```

```
IppStatus ippFilterScharrVertBorder_8u16s_C1R(const Ipp8u* pSrc, int srcStep, Ipp16s*
pDst, int dstStep, IppiSize roiSize, IppiBorderType borderType, Ipp8u borderValue, Ipp8u*
pBuffer);
```

```
IppStatus ippFilterScharrVertBorder_32f_C1R(const Ipp32f* pSrc, int srcStep, Ipp32f*
pDst, int dstStep, IppiSize roiSize, IppiBorderType borderType, Ipp32f borderValue, Ipp8u*
pBuffer);
```

### Parameters

<i>pSrc</i>	Pointer to the source image ROI.										
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.										
<i>pDst</i>	Pointer to the destination image ROI.										
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.										
<i>roiSize</i>	Size of the source and destination image ROI.										
<i>borderType</i>	Type of border (see <a href="#">Borders</a> ); following values are possible: <table data-bbox="503 1260 1331 1491"> <tr> <td><code>ippBorderConst</code></td><td>Values of all border pixels are set to constant.</td></tr> <tr> <td><code>ippBorderRepl</code></td><td>Replicated border is used.</td></tr> <tr> <td><code>ippBorderWrap</code></td><td>Wrapped border is used</td></tr> <tr> <td><code>ippBorderMirror</code></td><td>Mirrored border is used</td></tr> <tr> <td><code>ippBorderMirrorR</code></td><td>Mirrored border with replication is used</td></tr> </table>	<code>ippBorderConst</code>	Values of all border pixels are set to constant.	<code>ippBorderRepl</code>	Replicated border is used.	<code>ippBorderWrap</code>	Wrapped border is used	<code>ippBorderMirror</code>	Mirrored border is used	<code>ippBorderMirrorR</code>	Mirrored border with replication is used
<code>ippBorderConst</code>	Values of all border pixels are set to constant.										
<code>ippBorderRepl</code>	Replicated border is used.										
<code>ippBorderWrap</code>	Wrapped border is used										
<code>ippBorderMirror</code>	Mirrored border is used										
<code>ippBorderMirrorR</code>	Mirrored border with replication is used										
<i>borderValue</i>	The constant value to assign to the pixels in the constant border (not applicable for other border's type).										
<i>divisor</i>	The constant value that output pixel value is divided by.										
<i>pBuffer</i>	Pointer to the working buffer.										

### Description

The function `ippFilterScharrVertBorder` is declared in the `ippcv.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)). This function applies the vertical Scharr filter (x-derivative) to the source image *pSrc* and stores results to the destination image of the same size *pDst*. For the function flavor `ippFilterSchar-`

`rVertBorder_32f_C1R` the source image can be used as the destination image. The values of border pixels are assigned in accordance with the *borderType* and *borderValue* parameters. The kernel of this filter is a matrix of 3x3 size with the anchor in the center cell (red) and the following values:

```

 3    0   -3
10    0   -10
 3    0   -3

```

The function requires the working buffer *pBuffer* whose size should be computed by the function `ippiFilterScharrVertGetBufferSize` beforehand.

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with a zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> or <i>dstStep</i> is less than <i>roiSize.width</i> * <i>&lt;pixelSize&gt;</i>
<code>ippStsNotEvenStepErr</code>	Indicates an error condition if one of the step values is not divisible by 4 for floating-point images, or by 2 for short-integer images.
<code>ippStsBadArgErr</code>	Indicates an error if <i>borderType</i> or <i>divisor</i> has a wrong value.

## FilterSobelHorizBorder

*Applies horizontal Sobel filter with border.*

### Syntax

```

IppStatus ippiFilterSobelHorizBorder_8u8s_C1R(const Ipp8u* pSrc, int srcStep, Ipp8s* pDst,
int dstStep, IppiSize roiSize, IppiMaskSize mask, IppiBorderType borderType, Ipp8u
borderValue, int divisor, Ipp8u* pBuffer);

```

```

IppStatus ippiFilterSobelHorizBorder_8u16s_C1R(const Ipp8u* pSrc, int srcStep, Ipp16s*
pDst, int dstStep, IppiSize roiSize, IppiMaskSize mask, IppiBorderType borderType, Ipp8u
borderValue, Ipp8u* pBuffer);

```

```

IppStatus ippiFilterSobelHorizBorder_32f_C1R(const Ipp32f* pSrc, int srcStep, Ipp32f*
pDst, int dstStep, IppiSize roiSize, IppiMaskSize mask, IppiBorderType borderType, Ipp32f
borderValue, Ipp8u* pBuffer);

```

### Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.

<i>roiSize</i>	Size of the source and destination image ROI.										
<i>mask</i>	Type of the filter kernel.										
<i>borderType</i>	Type of border (see <a href="#">Borders</a> ); following values are possible: <table> <tr> <td><code>ippBorderConst</code></td><td>Values of all border pixels are set to constant.</td></tr> <tr> <td><code>ippBorderRepl</code></td><td>Replicated border is used.</td></tr> <tr> <td><code>ippBorderWrap</code></td><td>Wrapped border is used</td></tr> <tr> <td><code>ippBorderMirror</code></td><td>Mirrored border is used</td></tr> <tr> <td><code>ippBorderMirrorR</code></td><td>Mirrored border with replication is used</td></tr> </table>	<code>ippBorderConst</code>	Values of all border pixels are set to constant.	<code>ippBorderRepl</code>	Replicated border is used.	<code>ippBorderWrap</code>	Wrapped border is used	<code>ippBorderMirror</code>	Mirrored border is used	<code>ippBorderMirrorR</code>	Mirrored border with replication is used
<code>ippBorderConst</code>	Values of all border pixels are set to constant.										
<code>ippBorderRepl</code>	Replicated border is used.										
<code>ippBorderWrap</code>	Wrapped border is used										
<code>ippBorderMirror</code>	Mirrored border is used										
<code>ippBorderMirrorR</code>	Mirrored border with replication is used										
<i>borderValue</i>	The constant value to assign to the pixels in the constant border (not applicable for other border's type).										
<i>divisor</i>	The constant value that output pixel value is divided by.										
<i>pBuffer</i>	Pointer to the working buffer.										

## Description

The function `ippiFilterSobelHorizBorder` is declared in the `ippcv.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)). This function applies the horizontal Sobel filter (y-derivative) to the source image *pSrc* and stores results to the destination image of the same size *pDst*. For the function flavor `ippiFilterSobelHorizBorder_32f_C1R` the source image can be used as the destination image. The values of border pixels are assigned in accordance with the *borderType* and *borderValue* parameters. The kernel of this filter is the matrix of either 3x3 or 5x5 size that is specified by the parameter *mask*. The kernels have the following values with the anchor in the center cell (red):

			1	4	6	4	1	
1	2	1		2	8	12	8	2
0	0	0	or	0	0	0	0	0
-1	-2	-1		-2	-8	-12	-8	-4
				-1	-4	-6	-4	-1

The function requires the working buffer *pBuffer* whose size should be computed by the function `ippiFilterSobelHorizGetBufferSize` beforehand.

[Example](#) shows how the function `ippiFilterSobelHorizBorder_8u16s_C1R` can be used for edge detection.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with a zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> or <i>dstStep</i> is less than $roiSize.width * <pixelSize>$
<code>ippStsNotEvenStepErr</code>	Indicates an error condition if one of the step values is not divisible by 4 for floating-point images, or by 2 for short-integer images.

<code>ippStsBadArgErr</code>	Indicates an error if <i>borderType</i> or <i>divisor</i> has a wrong value.
<code>ippStsMaskErr</code>	Indicates an error condition if <i>mask</i> has a wrong value.

## FilterSobelVertBorder, FilterSobelNegVertBorder

*Applies vertical Sobel filter with border.*

### Syntax

```
ippStatus ippiFilterSobelVertBorder_8u8s_C1R(const Ipp8u* pSrc, int srcStep, Ipp8s* pDst,
int dstStep, IppiSize roiSize, IppiMaskSize mask, IppiBorderType borderType, Ipp8u
borderValue, int divisor, Ipp8u* pBuffer);
```

```
ippStatus ippiFilterSobelVertBorder_<mod>(const Ipp<srcDatatype>* pSrc, int srcStep,
Ipp<dstDatatype>* pDst, int dstStep, IppiSize roiSize, IppiMaskSize mask, IppiBorderType
borderType, Ipp<srcDatatype> borderValue, Ipp8u* pBuffer);
```

Supported values for *mod*:

8u16s\_C1R    32f\_C1R

```
ippStatus ippiFilterSobelNegVertBorder_8u8s_C1R(const Ipp8u* pSrc, int srcStep, Ipp8s*
pDst, int dstStep, IppiSize roiSize, IppiMaskSize mask, IppiBorderType borderType, Ipp8u
borderValue, int divisor, Ipp8u* pBuffer);
```

```
ippStatus ippiFilterSobelNegVertBorder_<mod>(const Ipp<srcDatatype>* pSrc, int srcStep,
Ipp<dstDatatype>* pDst, int dstStep, IppiSize roiSize, IppiMaskSize mask, IppiBorderType
borderType, Ipp<srcDatatype> borderValue, Ipp8u* pBuffer);
```

Supported values for *mod*:

8u16s\_C1R    32f\_C1R

### Parameters

<i>pSrc</i>	Pointer to the source image ROI.						
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.						
<i>pDst</i>	Pointer to the destination image ROI.						
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.						
<i>roiSize</i>	Size of the source and destination image ROI.						
<i>mask</i>	Type of the filter kernel.						
<i>borderType</i>	Type of border (see <a href="#">Borders</a> ); following values are possible: <table data-bbox="503 1680 1315 1831"> <tr> <td><code>ippBorderConst</code></td><td>Values of all border pixels are set to constant.</td></tr> <tr> <td><code>ippBorderRepl</code></td><td>Replicated border is used.</td></tr> <tr> <td><code>ippBorderWrap</code></td><td>Wrapped border is used</td></tr> </table>	<code>ippBorderConst</code>	Values of all border pixels are set to constant.	<code>ippBorderRepl</code>	Replicated border is used.	<code>ippBorderWrap</code>	Wrapped border is used
<code>ippBorderConst</code>	Values of all border pixels are set to constant.						
<code>ippBorderRepl</code>	Replicated border is used.						
<code>ippBorderWrap</code>	Wrapped border is used						

	<code>ippBorderMirror</code>	Mirrored border is used
	<code>ippBorderMirrorR</code>	Mirrored border with replication is used
<code>borderValue</code>		The constant value to assign to the pixels in the constant border (not applicable for other border's type).
<code>divisor</code>		The constant value that output pixel value is divided by.
<code>pBuffer</code>		Pointer to the working buffer.

## Description

The function `ippiFilterSobelVertBorder` and `ippiFilterSobelNegVertBoreder` are declared in the `ippcv.h` file. They operate with ROI (see [Regions of Interest in Intel IPP](#)). These functions apply the vertical Sobel filter (x-derivative) to the source image ROI `pSrc` and stores results to the destination image ROI of the same size `pDst`. Source image can be used as the destination image if they have the same data type. The values of border pixels are assigned in accordance with the `borderType` and `borderValue` parameters. The kernel of this filter is the matrix of either 3x3 or 5x5 size that is specified by the parameter `mask`. The anchor cell is the center cell of the kernel (red).

The function `ippiFilterSobelVertBorde` uses the kernels with the following coefficients:

			-1	-2	0	2	1	
-1	0	1		-4	-8	0	8	4
-2	0	2	or	-6	-12	0	12	6
-1	0	1		-4	-8	0	8	4
				-1	-2	0	2	1

The function `ippiFilterSobelNegVertBoreder` uses the kernels which coefficients are the same in magnitude but opposite in sign:

			1	2	0	-2	-1	
1	0	-1		4	8	0	-8	-4
2	0	-2	or	6	12	0	-12	-6
1	0	-1		4	8	0	-8	-4
				1	2	0	-2	-1

Both functions require the working buffer `pBuffer` whose size must be computed beforehand by the functions `ippiFilterSobelVertGetBufferSize` and `ippiFilterSobelNegVertGetBufferSize` respectively.

[Example](#) shows how the function `ippiFilterSobelNegVertBorder_8u16s_C1R` can be used for edge detection.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <code>roiSize</code> has a field with a zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <code>srcStep</code> or <code>dstStep</code> is less than <code>roiSize.width * &lt;pixelSize&gt;</code>



<code>ippStsNotEvenStepErr</code>	Indicates an error condition if one of the step values is not divisible by 4 for floating-point images, or by 2 for short-integer images.
<code>ippStsBadArgErr</code>	Indicates an error if <i>borderType</i> or <i>divisor</i> has a wrong value.
<code>ippStsMaskErr</code>	Indicates an error condition if <i>mask</i> has a wrong value.

## FilterSobelHorizSecondBorder

*Applies horizontal (second derivative) Sobel filter with border.*

### Syntax

```
IppStatus ippFilterSobelHorizSecondBorder_8u8s_C1R(const Ipp8u* pSrc, int srcStep, Ipp8s* pDst, int dstStep, IppiSize roiSize, IppiMaskSize mask, IppiBorderType borderType, Ipp8u borderValue, int divisor, Ipp8u* pBuffer);
```

```
IppStatus ippFilterSobelHorizSecondBorder_<mod>(const Ipp<srcDatatype>* pSrc, int srcStep, Ipp<dstDatatype>* pDst, int dstStep, IppiSize roiSize, IppiMaskSize mask, IppiBorderType borderType, Ipp<srcDatatype> borderValue, Ipp8u* pBuffer);
```

Supported values for *mod*:

8u16s\_C1R    32f\_C1R

### Parameters

<i>pSrc</i>	Pointer to the source image ROI.										
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.										
<i>pDst</i>	Pointer to the destination image ROI.										
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.										
<i>roiSize</i>	Size of the source and destination image ROI.										
<i>mask</i>	Type of the filter kernel.										
<i>borderType</i>	Type of border (see <a href="#">Borders</a> ); following values are possible: <table data-bbox="503 1444 1331 1675"> <tr> <td><code>ippBorderConst</code></td><td>Values of all border pixels are set to constant.</td></tr> <tr> <td><code>ippBorderRepl</code></td><td>Replicated border is used.</td></tr> <tr> <td><code>ippBorderWrap</code></td><td>Wrapped border is used</td></tr> <tr> <td><code>ippBorderMirror</code></td><td>Mirrored border is used</td></tr> <tr> <td><code>ippBorderMirrorR</code></td><td>Mirrored border with replication is used</td></tr> </table>	<code>ippBorderConst</code>	Values of all border pixels are set to constant.	<code>ippBorderRepl</code>	Replicated border is used.	<code>ippBorderWrap</code>	Wrapped border is used	<code>ippBorderMirror</code>	Mirrored border is used	<code>ippBorderMirrorR</code>	Mirrored border with replication is used
<code>ippBorderConst</code>	Values of all border pixels are set to constant.										
<code>ippBorderRepl</code>	Replicated border is used.										
<code>ippBorderWrap</code>	Wrapped border is used										
<code>ippBorderMirror</code>	Mirrored border is used										
<code>ippBorderMirrorR</code>	Mirrored border with replication is used										
<i>borderValue</i>	The constant value to assign to the pixels in the constant border (not applicable for other border's type).										
<i>divisor</i>	The constant value that output pixel value is divided by.										
<i>pBuffer</i>	Pointer to the working buffer.										

## Description

The function `ippiFilterSobelHorizSecondBordered` is declared in the `ippcv.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)). This function applies the second derivative horizontal Sobel filter ( $y$ -derivative) to the source image `pSrc` and stores results to the destination image of the same size `pDst`. Source image can be used as the destination image if they both have the same data type. The values of border pixels are assigned in accordance with the `borderType` and `borderValue` parameters. The kernel of this filter is the matrix of either 3x3 or 5x5 size that is specified by the parameter `mask`. The kernels have the following values with the anchor in the center cell (red):

				1	4	6	4	1					
				0	0	0	0	0					
				-2	-8	-12	-8	-2					
				0	0	0	0	0					
				1	4	6	4	1					

or

1	2	1						
-2	-4	-2						
1	2	1						

The function requires the working buffer `pBuffer` whose size should be computed by the function `ippiFilterSobelHorizSecondGetBufferSize` beforehand.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <code>roiSize</code> has a field with a zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <code>srcStep</code> or <code>dstStep</code> is less than <code>roiSize.width * &lt;pixelSize&gt;</code>
<code>ippStsNotEvenStepErr</code>	Indicates an error condition if one of the step values is not divisible by 4 for floating-point images, or by 2 for short-integer images.
<code>ippStsBadArgErr</code>	Indicates an error if <code>borderType</code> or <code>divisor</code> has a wrong value.
<code>ippStsMaskErr</code>	Indicates an error condition if <code>mask</code> has a wrong value.

## FilterSobelVertSecondBorder

*Applies vertical (second derivative) Sobel filter with border.*

---

## Syntax

```
IppStatus ippiFilterSobelVertSecondBorder_8u8s_C1R(const Ipp8u* pSrc, int srcStep, Ipp8s* pDst, int dstStep, IppiSize roiSize, IppiMaskSize mask, IppiBorderType borderType, Ipp8u borderValue, int divisor, Ipp8u* pBuffer);
```

```
ippStatus ippiFilterSobelVertSecondBorder_<mod>(const Ipp<srcDatatype>* pSrc, int srcStep,
Ipp<dstDatatype>* pDst, int dstStep, IppiSize roiSize, IppiMaskSize mask, IppiBorderType
borderType, Ipp<srcDatatype> borderValue, Ipp8u* pBuffer);
```

Supported values for mod:

8u16s\_C1R    32f\_C1R

## Parameters

<i>pSrc</i>	Pointer to the source image ROI.										
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.										
<i>pDst</i>	Pointer to the destination image ROI.										
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.										
<i>roiSize</i>	Size of the source and destination image ROI.										
<i>mask</i>	Type of the filter kernel.										
<i>borderType</i>	Type of border (see <a href="#">Borders</a> ); following values are possible: <table> <tr> <td><code>ippBorderConst</code></td><td>Values of all border pixels are set to constant.</td></tr> <tr> <td><code>ippBorderRepl</code></td><td>Replicated border is used.</td></tr> <tr> <td><code>ippBorderWrap</code></td><td>Wrapped border is used</td></tr> <tr> <td><code>ippBorderMirror</code></td><td>Mirrored border is used</td></tr> <tr> <td><code>ippBorderMirrorR</code></td><td>Mirrored border with replication is used</td></tr> </table>	<code>ippBorderConst</code>	Values of all border pixels are set to constant.	<code>ippBorderRepl</code>	Replicated border is used.	<code>ippBorderWrap</code>	Wrapped border is used	<code>ippBorderMirror</code>	Mirrored border is used	<code>ippBorderMirrorR</code>	Mirrored border with replication is used
<code>ippBorderConst</code>	Values of all border pixels are set to constant.										
<code>ippBorderRepl</code>	Replicated border is used.										
<code>ippBorderWrap</code>	Wrapped border is used										
<code>ippBorderMirror</code>	Mirrored border is used										
<code>ippBorderMirrorR</code>	Mirrored border with replication is used										
<i>borderValue</i>	The constant value to assign to the pixels in the constant border (not applicable for other border's type).										
<i>divisor</i>	The constant value that output pixel value is divided by.										
<i>pBuffer</i>	Pointer to the working buffer.										

## Description

The function `ippiFilterSobelVertSecondBorder` is declared in the `ippcv.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)). This function applies the second derivative vertical Sobel filter (x-derivative) to the source image *pSrc* and stores results to the destination image of the same size *pDst*. Source image can be used as the destination image if they both have the same data type. The values of border pixels are assigned in accordance with the *borderType* and *borderValue* parameters. The kernel of this filter is the matrix of either 3x3 or 5x5 size that is specified by the parameter *mask*. The kernels have the following values with the anchor in the center cell (red):

			1	0	-2	0	1	
1	-2	1		4	0	-8	0	4
2	-4	2	or	6	0	-12	0	6
1	-2	1		4	0	-8	0	4
				1	0	-2	0	1

The function requires the working buffer *pBuffer* whose size should be computed by the function [ippiFilterSobelVertSecondGetBufferSize](#) beforehand.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with a zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> or <i>dstStep</i> is less than <i>roiSize.width</i> * <i>&lt;pixelSize&gt;</i>
<code>ippStsNotEvenStepErr</code>	Indicates an error condition if one of the step values is not divisible by 4 for floating-point images, or by 2 for short-integer images.
<code>ippStsBadArgErr</code>	Indicates an error if <i>borderType</i> or <i>divisor</i> has a wrong value.
<code>ippStsMaskErr</code>	Indicates an error condition if <i>mask</i> has a wrong value.

## Example

The code example below shows how to use the `ippiFilterSobelVertSecondBorder_8u16s_C1R` function.

```

Ipp8u pSrc[8*8]={
    4, 4, 4, 4, 4, 4, 4, 4,
    4, 3, 3, 3, 3, 3, 3, 4,
    4, 3, 2, 2, 2, 2, 3, 4,
    4, 3, 2, 1, 1, 2, 3, 4,
    4, 3, 2, 1, 1, 2, 3, 4,
    4, 3, 2, 2, 2, 2, 3, 4,
    4, 3, 3, 3, 3, 3, 3, 4,
    4, 4, 4, 4, 4, 4, 4, 4
};
Ipp16s pDst[8*8];
IppiSize Roi = { 8, 8 };
Ipp8u borderValue = 3;
int pBufferSize;

ippiFilterSobelVertSecondGetBufferSize_8u16s_C1R(Roi, ippMskSize3x3,
    &pBufferSize);

Ipp8u* pBuffer = ippiMalloc_8u_C1 ( 8, 8, &pBufferSize );

ippiFilterSobelVertSecondBorder_8u16s_C1R ( pSrc, 8, pDst,
    8*sizeof(Ipp16s), Roi, ippMskSize3x3, ippBorderConst, borderValue,
    pBuffer );

```

Result:

```

4 4 4 4 4 4 4 4
4 3 3 3 3 3 3 4
4 3 2 2 2 2 3 4
4 3 2 1 1 2 3 4
4 3 2 1 1 2 3 4    pSrc
4 3 2 2 2 2 3 4
4 3 3 3 3 3 3 4
4 4 4 4 4 4 4 4

```

```

-4 1 0 0 0 0 1 -4

```

```

-7 2 1 0 0 1 2 -7
-8 1 2 1 1 2 1 -8
-8 0 1 3 3 1 0 -8
-8 0 1 3 3 1 0 -8   pDst
-8 1 2 1 1 2 1 -8
-7 2 1 0 0 1 2 -7
-4 1 0 0 0 0 1 -4

```

## FilterSobelCrossBorder

*Applies second derivative cross Sobel filter with border.*

### Syntax

```

IppStatus ippiFilterSobelCrossBorder_8u8s_C1R(const Ipp8u* pSrc, int srcStep, Ipp8s* pDst,
int dstStep, IppiSize roiSize, IppiMaskSize mask, IppiBorderType borderType, Ipp8u
borderValue, int divisor, Ipp8u* pBuffer);

```

```

IppStatus ippiFilterSobelCrossBorder_<mod>(const Ipp<srcDatatype>* pSrc, int srcStep,
Ipp<dstDatatype>* pDst, int dstStep, IppiSize roiSize, IppiMaskSize mask, IppiBorderType
borderType, Ipp<srcDatatype> borderValue, Ipp8u* pBuffer);

```

Supported values for mod:

8u16s\_C1R    32f\_C1R

### Parameters

<i>pSrc</i>	Pointer to the source image ROI.										
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.										
<i>pDst</i>	Pointer to the destination image ROI.										
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.										
<i>roiSize</i>	Size of the source and destination image ROI.										
<i>mask</i>	Type of the filter kernel.										
<i>borderType</i>	Type of border (see <a href="#">Borders</a> ); following values are possible: <table data-bbox="505 1465 1328 1690"> <tr> <td><code>ippBorderConst</code></td><td>Values of all border pixels are set to constant.</td></tr> <tr> <td><code>ippBorderRepl</code></td><td>Replicated border is used.</td></tr> <tr> <td><code>ippBorderWrap</code></td><td>Wrapped border is used</td></tr> <tr> <td><code>ippBorderMirror</code></td><td>Mirrored border is used</td></tr> <tr> <td><code>ippBorderMirrorR</code></td><td>Mirrored border with replication is used</td></tr> </table>	<code>ippBorderConst</code>	Values of all border pixels are set to constant.	<code>ippBorderRepl</code>	Replicated border is used.	<code>ippBorderWrap</code>	Wrapped border is used	<code>ippBorderMirror</code>	Mirrored border is used	<code>ippBorderMirrorR</code>	Mirrored border with replication is used
<code>ippBorderConst</code>	Values of all border pixels are set to constant.										
<code>ippBorderRepl</code>	Replicated border is used.										
<code>ippBorderWrap</code>	Wrapped border is used										
<code>ippBorderMirror</code>	Mirrored border is used										
<code>ippBorderMirrorR</code>	Mirrored border with replication is used										
<i>borderValue</i>	The constant value to assign to the pixels in the constant border (not applicable for other border's type).										
<i>divisor</i>	The constant value that output pixel value is divided by.										
<i>pBuffer</i>	Pointer to the working buffer.										

## Description

The function `ippiFilterSobelCrossBorder` is declared in the `ippcv.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function applies the second derivative cross Sobel filter ( $xy$ -derivative) to the source image `pSrc` and stores results to the destination image of the same size `pDst`. Source image can be used as the destination image if they both have the same data type. The values of border pixels are assigned in accordance with the `borderType` and `borderValue` parameters. The kernel of this filter is the matrix of either 3x3 or 5x5 size that is specified by the parameter `mask`. The kernels have the following values with the anchor in the center cell (red):

				-1	-2	0	2	1
-1	0	1		-2	-4	0	4	2
0	0	0	or	0	0	0	0	0
1	0	-1		2	4	0	-4	-2
				1	2	0	-2	-1

The function requires the working buffer `pBuffer` whose size should be computed by the function `ippiFilterSobelCrossGetBufferSize` beforehand.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <code>roiSize</code> has a field with a zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <code>srcStep</code> or <code>dstStep</code> is less than <code>roiSize.width * &lt;pixelSize&gt;</code>
<code>ippStsNotEvenStepErr</code>	Indicates an error condition if one of the step values is not divisible by 4 for floating-point images, or by 2 for short-integer images.
<code>ippStsBadArgErr</code>	Indicates an error if <code>borderType</code> or <code>divisor</code> has a wrong value.
<code>ippStsMaskErr</code>	Indicates an error condition if <code>mask</code> has a wrong value.

## FilterLaplacianBorder

*Applies Laplacian filter with border.*

---

### Syntax

```
IppStatus ippiFilterLaplacianBorder_8u8s_C1R(const Ipp8u* pSrc, int srcStep, Ipp8s* pDst,
int dstStep, IppiSize roiSize, IppiMaskSize mask, IppiBorderType borderType, Ipp8u
borderValue, int divisor, Ipp8u* pBuffer);
```

```
IppStatus ippiFilterLaplacianBorder_<mod>(const Ipp<srcDatatype>* pSrc, int srcStep,
Ipp<dstDatatype>* pDst, int dstStep, IppiSize roiSize, IppiMaskSize mask, IppiBorderType
borderType, Ipp<srcDatatype> borderValue, Ipp8u* pBuffer);
```

Supported values for mod:

8u16s\_C1R    32f\_C1R

## Parameters

<i>pSrc</i>	Pointer to the source image ROI.										
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.										
<i>pDst</i>	Pointer to the destination image ROI.										
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.										
<i>roiSize</i>	Size of the source and destination image ROI.										
<i>mask</i>	Type of the filter kernel.										
<i>borderType</i>	Type of border (see <a href="#">Borders</a> ); following values are possible: <table> <tr> <td><code>ippiBorderConst</code></td><td>Values of all border pixels are set to constant.</td></tr> <tr> <td><code>ippiBorderRepl</code></td><td>Replicated border is used.</td></tr> <tr> <td><code>ippiBorderWrap</code></td><td>Wrapped border is used</td></tr> <tr> <td><code>ippiBorderMirror</code></td><td>Mirrored border is used</td></tr> <tr> <td><code>ippiBorderMirrorR</code></td><td>Mirrored border with replication is used</td></tr> </table>	<code>ippiBorderConst</code>	Values of all border pixels are set to constant.	<code>ippiBorderRepl</code>	Replicated border is used.	<code>ippiBorderWrap</code>	Wrapped border is used	<code>ippiBorderMirror</code>	Mirrored border is used	<code>ippiBorderMirrorR</code>	Mirrored border with replication is used
<code>ippiBorderConst</code>	Values of all border pixels are set to constant.										
<code>ippiBorderRepl</code>	Replicated border is used.										
<code>ippiBorderWrap</code>	Wrapped border is used										
<code>ippiBorderMirror</code>	Mirrored border is used										
<code>ippiBorderMirrorR</code>	Mirrored border with replication is used										
<i>borderValue</i>	The constant value to assign to the pixels in the constant border (not applicable for other border's type).										
<i>divisor</i>	The constant value that output pixel value is divided by.										
<i>pBuffer</i>	Pointer to the working buffer.										

## Description

The function `ippiFilterLaplacianBorderi` is declared in the `ippcv.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function applies the laplacian filter to the source image *pSrc* and stores results to the destination image of the same size *pDst*. Source image can be used as the destination image if they both have the same data type. The values of border pixels are assigned in accordance with the *borderType* and *borderValue* parameters. The kernel of this filter is the matrix of either 3x3 or 5x5 size that is specified by the parameter *mask*. The kernels have the following values with the anchor in the center cell (red):

				2	4	4	4	2
2	0	2		4	0	-8	0	4
0	-8	0	or	4	-8	-12	-8	4
2	0	2		4	0	-8	0	4
				2	4	4	4	2

The function requires the working buffer *pBuffer* whose size should be computed by the function [ippiFilterLaplacianGetBufferSize](#) beforehand.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with a zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> or <i>dstStep</i> is less than <i>roiSize.width</i> * <i>&lt;pixelSize&gt;</i>
<code>ippStsNotEvenStepErr</code>	Indicates an error condition if one of the step values is not divisible by 4 for floating-point images, or by 2 for short-integer images.
<code>ippStsBadArgErr</code>	Indicates an error if <i>borderType</i> or <i>divisor</i> has a wrong value.
<code>ippStsMaskErr</code>	Indicates an error condition if <i>mask</i> has a wrong value.

## FilterGaussBorder

*Applies Gauss filter with border.*

---

### Syntax

```
ippStatus ippiFilterGaussBorder_32f_C1R(const Ipp32f* pSrc, int srcStep, Ipp32f* pDst,
int dstStep, IppiSize roiSize, int kernelSize, Ipp32f sigma, IppiBorderType borderType,
Ipp32f borderValue, Ipp8u* pBuffer);
```

### Parameters

<i>pSrc</i>	Pointer to the source image ROI.										
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.										
<i>pDst</i>	Pointer to the destination image ROI.										
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.										
<i>roiSize</i>	Size of the source and destination image ROI.										
<i>kernelSize</i>	Specifies the size of the Gaussian kernel, must be odd and greater than or equal to 3.										
<i>sigma</i>	Standard deviation of the Gaussian kernel.										
<i>borderType</i>	Type of border (see <a href="#">Borders</a> ); the following values are possible: <table> <tr> <td><code>ippBorderConst</code></td><td>Values of all border pixels are set to constant.</td></tr> <tr> <td><code>ippBorderRepl</code></td><td>Replicated border is used.</td></tr> <tr> <td><code>ippBorderWrap</code></td><td>Wrapped border is used</td></tr> <tr> <td><code>ippBorderMirror</code></td><td>Mirrored border is used</td></tr> <tr> <td><code>ippBorderMirrorR</code></td><td>Mirrored border with replication is used</td></tr> </table>	<code>ippBorderConst</code>	Values of all border pixels are set to constant.	<code>ippBorderRepl</code>	Replicated border is used.	<code>ippBorderWrap</code>	Wrapped border is used	<code>ippBorderMirror</code>	Mirrored border is used	<code>ippBorderMirrorR</code>	Mirrored border with replication is used
<code>ippBorderConst</code>	Values of all border pixels are set to constant.										
<code>ippBorderRepl</code>	Replicated border is used.										
<code>ippBorderWrap</code>	Wrapped border is used										
<code>ippBorderMirror</code>	Mirrored border is used										
<code>ippBorderMirrorR</code>	Mirrored border with replication is used										



<i>borderValue</i>	The constant value to assign to the pixels in the constant border (not applicable for other border's type).
<i>pBuffer</i>	Pointer to the working buffer.

## Description

The function `ippiFilterGaussBorder` is declared in the `ippcv.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function applies the Gaussian filter to the source image ROI *pSrc*. The kernel of this filter is the matrix of size *kernelSize\*kernelSize* with the standard deviation *sigma*. The values of the elements of the Gaussian kernel are calculated according to the formula:

$$G(i, j) = \exp \left\{ -\frac{(K/2 - i)^2 + (K/2 - j)^2}{2\sigma^2} \right\}, \quad i, j = 0, \dots, K$$

and then are normalized. The anchor cell is the center of the kernel.

The function requires the working buffer *pBuffer* whose size should be computed by the function `ippiFilterGaussGetBufferSize` beforehand.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with a zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> or <i>dstStep</i> is less than <i>roiSize.width * &lt;pixelSize&gt;</i> .
<code>ippStsNotEvenStepErr</code>	Indicates an error condition if one of the step values is not divisible by 4.
<code>ippStsBorderErr</code>	Indicates an error condition if <i>borderType</i> has a wrong value.
<code>ippStsBadArgErr</code>	Indicates an error condition if <i>kernelSize</i> is less than 3, or <i>sigma</i> is less than or equal to 0.

## FilterLowpassBorder

*Applies lowpass filter with border.*

---

### Syntax

```
ippStatus ippiFilterLowpassBorder_8u_C1R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst, int dstStep, IppiSize roiSize, IppiMaskSize mask, IppiBorderType borderType, Ipp8u borderValue, Ipp8u* pBuffer);
```

```
IppStatus ippiFilterLowpassBorder_32f_C1R(const Ipp32f* pSrc, int srcStep, Ipp32f* pDst,
int dstStep, IppiSize roiSize, IppiMaskSize mask, IppiBorderType borderType, Ipp32f
borderValue, Ipp8u* pBuffer);
```

## Parameters

<i>pSrc</i>	Pointer to the source image ROI.										
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.										
<i>pDst</i>	Pointer to the destination image ROI.										
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.										
<i>roiSize</i>	Size of the source and destination image ROI.										
<i>mask</i>	Type of the filter kernel.										
<i>borderType</i>	Type of border (see <a href="#">Borders</a> ); the following values are possible: <table> <tr> <td><code>ippBorderConst</code></td><td>Values of all border pixels are set to constant.</td></tr> <tr> <td><code>ippBorderRepl</code></td><td>Replicated border is used.</td></tr> <tr> <td><code>ippBorderWrap</code></td><td>Wrapped border is used</td></tr> <tr> <td><code>ippBorderMirror</code></td><td>Mirrored border is used</td></tr> <tr> <td><code>ippBorderMirrorR</code></td><td>Mirrored border with replication is used</td></tr> </table>	<code>ippBorderConst</code>	Values of all border pixels are set to constant.	<code>ippBorderRepl</code>	Replicated border is used.	<code>ippBorderWrap</code>	Wrapped border is used	<code>ippBorderMirror</code>	Mirrored border is used	<code>ippBorderMirrorR</code>	Mirrored border with replication is used
<code>ippBorderConst</code>	Values of all border pixels are set to constant.										
<code>ippBorderRepl</code>	Replicated border is used.										
<code>ippBorderWrap</code>	Wrapped border is used										
<code>ippBorderMirror</code>	Mirrored border is used										
<code>ippBorderMirrorR</code>	Mirrored border with replication is used										
<i>borderValue</i>	The constant value to assign to the pixels in the constant border (not applicable for other border's type).										
<i>pBuffer</i>	Pointer to the working buffer.										

## Description

The function `ippiFilterLowpassBorder` is declared in the `ippcv.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function applies the lowpass filter (blur operation) to the source image *pSrc* and stores results to the destination image of the same size *pDst*. Source image can be used as the destination image if they both have the same data type. The values of border pixels are assigned in accordance with the *borderType* and *borderValue* parameters. The kernel of this filter is the matrix of either 3x3 or 5x5 size that is specified by the parameter *mask*. The anchor cell is the center cell (red) of the kernel.

The 3x3 filter uses the kernel with the following values:

```
1/9  1/9  1/9
1/9  1/9  1/9
1/9  1/9  1/9
```

The 5x5 filter uses the kernel with the following values:

```
1/25  1/25  1/25  1/25  1/25
1/25  1/25  1/25  1/25  1/25
1/25  1/25  1/25  1/25  1/25
1/25  1/25  1/25  1/25  1/25
1/25  1/25  1/25  1/25  1/25
```

The function requires the working buffer *pBuffer* whose size should be computed by the function [ippiFilterLowpassGetBufferSize](#) beforehand.

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with a zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> or <i>dstStep</i> is less than <i>roiSize.width</i> * <i>&lt;pixelSize&gt;</i> .
<code>ippStsNotEvenStepErr</code>	Indicates an error condition if one of the step values is not divisible by 4 for floating-point images.
<code>ippStsBorderErr</code>	Indicates an error condition if <i>borderType</i> has a wrong value.
<code>ippStsMaskErr</code>	Indicates an error condition if <i>mask</i> has a wrong value.



# Image Linear Transforms

This chapter describes the Intel® IPP image processing functions that perform linear transform operations on an image buffer.

These operations include Fast Fourier Transform (FFT), Discrete Fourier Transform (DFT), and Discrete Cosine Transform (DCT).

The table below lists the Intel IPP linear transform functions:

## Image Linear Transform Functions

Function Base Name	Operation
<b>Fourier Transforms</b>	
<code>FFTInitAlloc</code>	Allocates memory and fills in context data needed for the image FFT functions to operate.
<code>FFTFree</code>	Deallocates memory used by the FFT context structure.
<code>FFTGetBufSize</code>	Determines the size of an external work buffer that can be used by the FFT functions.
<code>FFTForward</code>	Applies forward Fast Fourier Transform to an image.
<code>FFTInv</code>	Applies inverse Fast Fourier Transform to complex source data and stores results in a destination image.
<code>DFTInitAlloc</code>	Allocates memory and fills in context data needed for the image DFT functions to operate.
<code>DFTFree</code>	Deallocates memory used by the DFT context structure.
<code>DFTGetBufSize</code>	Determines the size of an external work buffer that can be used by the DFT functions.
<code>DFTForward</code>	Applies forward Discrete Fourier Transform to an image.
<code>DFTInv</code>	Applies inverse Discrete Fourier Transform to complex source data and stores results in a destination image.
<code>MulPack</code>	Multiplies two source images with data in packed format and stores the result a destination image in packed format.
<code>MulPackConj</code>	Multiplies two source images with data in packed format and stores the result in a destination image that is complex-conjugate to that obtained with the <code>ippiMulPack</code> function.
<code>Magnitude</code>	Computes the magnitude of elements of complex data images.
<code>MagnitudePack</code>	Computes the magnitude of elements of an image in packed format.
<code>Phase</code>	Computes the phase of elements of complex data images.
<code>PhasePack</code>	Computes the phase of elements of an image in packed format.
<code>PolarToCart</code>	Converts an image in the polar coordinate form to Cartesian coordinate form.
<code>PackToCplxExtend</code>	Converts an image in packed format to a complex data image.
<code>CplxExtendToPack</code>	Converts a complex data image to an image in packed format.
<b>Windowing Functions</b>	
<code>WinBartlett</code> , <code>WinBartlettSep</code>	Applies Bartlett window function to the image.
<code>WinHamming</code> , <code>WinHammingSep</code>	Applies Hamming window function to the image.

Function Base Name	Operation
<b>Discrete Cosine Transforms</b>	
<a href="#">DCTFwdInitAlloc</a>	Allocates memory and fills in context data needed for the forward DCT function to operate
<a href="#">DCTInvInitAlloc</a>	Allocates memory and fills in context data needed for the inverse DCT function to operate
<a href="#">DCTFwdFree</a>	Deallocates memory used by the forward DCT context structure.
<a href="#">DCTInvFree</a>	Deallocates memory used by the inverse DCT context structure.
<a href="#">DCTFwdGetBufSize</a>	Determines the size of an external work buffer that can be used by the forward DCT function
<a href="#">DCTInvGetBufSize</a>	Determines the size of an external work buffer that can be used by the inverse DCT function
<a href="#">DCTFwd</a>	Performs a forward DCT of an image
<a href="#">DCTInv</a>	Performs an inverse DCT of an image
<a href="#">DCT8x8Fwd</a>	Performs a forward DCT on a buffer of 8x8 size
<a href="#">DCT8x8Inv</a> , <a href="#">DCT8x8Inv_A10</a>	Performs an inverse DCT on a buffer of 8x8 size
<a href="#">DCT8x8FwdLS</a>	Performs a forward DCT on a 2D buffer of 8x8 size with level shift
<a href="#">DCT8x8InvLSClip</a>	Performs an inverse DCT on a 2D buffer of 8x8 size with level shift
<a href="#">DCT8x8To2x2Inv</a> , <a href="#">DCT8x8To4x4Inv</a>	Perform an inverse DCT on a 2D buffer of 8x8 size with further downsampling to 2x2 or 4x4 size
<a href="#">DCT8x8Inv_2x2</a> , <a href="#">DCT8x8Inv_4x4</a>	Perform an inverse DCT on a top left quadrant 2x2 or 4x4 of the buffer of 8x8 size

To speed up performance, linear transform functions use precomputed auxiliary data that is needed for computation of the transforms (that is, tables of twiddle factors for FFT functions). This data is calculated by the respective initialization functions and passed to the transform functions in context structures specific for each type of transform.

Most linear transform functions in Intel IPP have code branches that implement different algorithms to compute the results. You can choose the desired code variety to be used by the transform function by setting the *hint* argument to one of the values listed in [Table "Hint Arguments for Linear Transform Functions"](#):

#### Hint Arguments for Linear Transform Functions

Value	Description
<a href="#">ippAlgHintNone</a>	The computation algorithm will be chosen by the internal function logic.
<a href="#">ippAlgHintFast</a>	Fast algorithm must be used. The output results will be less accurate.
<a href="#">ippAlgHintAccurate</a>	High accuracy algorithm must be used. The function will need more time to execute.

Intel IPP linear transform functions can use external work buffers for storing data and intermediate results, which eliminates the need to allocate and free internal memory buffers and thus helps to further increase function performance. To determine the required work buffer size, use one of the respective support functions specific for each transform type. In case when no external buffer is specified, the transform functions handle memory allocation internally.

All Intel IPP linear transform functions except DCT of 8x8 size work on images with floating-point data only.

## Fourier Transforms

Intel IPP functions that compute FFT and DFT can process both real and complex images. Function flavors operating on real data are distinguished by R suffix present in function-specific modifier of their full name, whereas complex flavors' names include C suffix (see [Function Naming](#) in Chapter 2).

The results of computing the Fourier transform can be normalized by specifying the appropriate value of *flag* argument for context initialization. This parameter sets up a pair of matched normalization factors to be used in forward and inverse transforms as listed in the following table:

#### Normalization Factors for Fourier Transform Results

Value of <i>flag</i> Argument	Normalization Factors	
	Forward Transform	Inverse Transform
IPP_FFT_DIV_FWD_BY_N	1/MN	1
IPP_FFT_DIV_INV_BY_N	1	1/MN
IPP_FFT_DIV_BY_SQRTN	1/sqrt(MN)	1/sqrt(MN)
IPP_FFT_NODIV_BY_ANY	1	1

In this table, *N* and *M* denote the length of Fourier transform in the x- and y-directions, respectively (or, equivalently, the number of columns and rows in the 2D array being transformed).

For the FFT, these lengths must be integer powers of 2, that is  $N=2^{\text{orderX}}$ ,  $M=2^{\text{orderY}}$ , where power exponents are known as order of FFT.

For the DFT, *N* and *M* can take on arbitrary integer non-negative values.

#### Real - Complex Packed (RCPack2D) Format

The forward Fourier transform of a real two-dimensional image data yields a matrix of complex results which has conjugate-symmetric properties. Intel IPP functions use packed format RCPack2D for storing and retrieving data of this type. Accordingly, real flavors of the inverse Fourier transform functions convert packed complex conjugate-symmetric data back to its real origin. The RCPack2D format exploits the complex conjugate symmetry of the transformed data to store only a half of the resulting Fourier coefficients. For the *N* by *M* transform, the respective FFT and DFT functions actually store real and imaginary parts of the complex Fourier coefficients  $A(i, j)$  for  $i = 0, \dots, M-1$ ;  $j = 0, \dots, N/2$  in a single real array of dimensions (*N*, *M*). The RCPack2D storage format is slightly different for odd and even *M* and is arranged in accordance with the following tables:

##### RCPack2D Storage for Odd Number of Rows

Re $A(0,0)$	Re $A(0,1)$	Im $A(0,1)$	...	Re $A(0,(N-1)/2)$	Im $A(0,(N-1)/2)$	Re $A(0,N/2)$
Re $A(1,0)$	Re $A(1,1)$	Im $A(1,1)$	...	Re $A(1,(N-1)/2)$	Im $A(1,(N-1)/2)$	Re $A(1,N/2)$
Im $A(1,0)$	Re $A(2,1)$	Im $A(2,1)$	...	Re $A(2,(N-1)/2)$	Im $A(2,(N-1)/2)$	Im $A(1,N/2)$
...	...	...	...	...	...	...
Re $A(M/2,0)$	Re $A(M-2,1)$	Im $A(M-2,1)$	...	Re $A(M-2,(N-1)/2)$	Im $A(M-2,(N-1)/2)$	Re $A(M/2,N/2)$
Im $A(M/2,0)$	Re $A(M-1,1)$	Im $A(M-1,1)$	...	Re $A(M-1,(N-1)/2)$	Im $A(M-1,(N-1)/2)$	Im $A(M/2,N/2)$

##### RCPack2D Storage for Even Number of Rows

Re $A(0,0)$	Re $A(0,1)$	Im $A(0,1)$	...	Re $A(0,(N-1)/2)$	Im $A(0,(N-1)/2)$	Re $A(0,N/2)$
Re $A(1,0)$	Re $A(1,1)$	Im $A(1,1)$	...	Re $A(1,(N-1)/2)$	Im $A(1,(N-1)/2)$	Re $A(1,N/2)$
Im $A(1,0)$	Re $A(2,1)$	Im $A(2,1)$	...	Re $A(2,(N-1)/2)$	Im $A(2,(N-1)/2)$	Im $A(1,N/2)$
...	...	...	...	...	...	...

Re $A(0,0)$	Re $A(0,1)$	Im $A(0,1)$	...	Re $A(0,(N-1)/2)$	Im $A(0,(N-1)/2)$	Re $A(0,N/2)$
Re $A(M/2-1,0)$	Re $A(M-3,1)$	Im $A(M-3,1)$	...	Re $A(M-3,(N-1)/2)$	Im $A(M-3,(N-1)/2)$	Re $A(M/2-1,N/2)$
Im $A(M/2-1,0)$	Re $A(M-2,1)$	Im $A(M-2,1)$	...	Re $A(M-2,(N-1)/2)$	Im $A(M-2,(N-1)/2)$	Im $A(M/2-1,N/2)$
Re $A(M/2,0)$	Re $A(M-1,1)$	Im $A(M-1,1)$	...	Re $A(M-1,(N-1)/2)$	Im $A(M-1,(N-1)/2)$	Re $A(M/2,N/2)$

The shaded columns to the right side of the tables indicate values for even  $N$  only.

Note the above tables show the arrangement of coefficients for one channel. For multichannel images the channel coefficients are clustered and stored consecutively, for example, for 3-channel image they are stored in the following way: C1-Re  $A(0,0)$ ; C2-Re  $A(0,0)$ ; C3-Re  $A(0,0)$ ; C1-Re  $A(0,1)$ ; C2-Re  $A(0,1)$ ; C3-Re  $A(0,1)$ ; C1-Im  $A(0,1)$ ; C2-Im  $A(0,1)$ ; ...

The remaining Fourier coefficients are obtained using the following relationships based on conjugate-symmetric properties:

$$A(i,j) = \text{conj}(A(M-i,N-j)), \quad i = 1, \dots, M-1; \quad j = 1, \dots, N-1$$

$$A(0,j) = \text{conj}(A(0,N-j)), \quad j = 1, \dots, N-1$$

$$A(i,0) = \text{conj}(A(M-i,0)), \quad i = 1, \dots, M-1$$

## FFTInitAlloc

*Allocates memory and fills in context data needed for the image FFT functions to operate.*

### Syntax

```
IppStatus ippiFFTInitAlloc_R_32s (IppiFFTSpec_R_32s** ppFFTSpec, int orderX, int orderY,
int flag, IppHintAlgorithm hint);
```

```
IppStatus ippiFFTInitAlloc_R_32f (IppiFFTSpec_R_32f** ppFFTSpec, int orderX, int orderY,
int flag, IppHintAlgorithm hint);
```

```
IppStatus ippiFFTInitAlloc_C_32fc (IppiFFTSpec_C_32fc** ppFFTSpec, int orderX, int orderY,
int flag, IppHintAlgorithm hint);
```

### Parameters

<i>ppFFTSpec</i>	Pointer to the pointer to the FFT context structure being initialized.
<i>orderX, orderY</i>	Order of the FFT in x- and y- directions, respectively.
<i>flag</i>	Flag to choose the results normalization option.
<i>hint</i>	Option to select the algorithmic implementation of the transform function (see <a href="#">Table "Hint Arguments for Linear Transform Functions"</a> ).



## Description

The function `ippiFFTInitAlloc` is declared in the `ippi.h` file. This function allocates memory and initializes the context structure `pFFTSpec` needed to compute the forward and inverse FFT of a two-dimensional image data.

The `ippiFFTFwd` and `ippiFFTInv` functions called with the pointer to the initialized `pFFTSpec` structure as an argument will compute the fast Fourier transform with the following characteristics:

- length  $N=2^{orderX}$  in x-direction by  $M=2^{orderY}$  in y-direction
- results normalization mode as set by `flag` (see [Table "Normalization Factors for Fourier Transform Results"](#))
- computation algorithm indicated by `hint`.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <code>pFFTSpec</code> pointer is <code>NULL</code> .
<code>ippStsFftOrderErr</code>	Indicates an error condition if the FFT order value is illegal.
<code>ippStsFFTFlagErr</code>	Indicates an error condition if <code>flag</code> has an illegal value.
<code>ippStsMemAllocErr</code>	Indicates an error condition if memory allocation fails.

## FFTFree

Deallocates memory used by the FFT context structure.

## Syntax

```
IppStatus ippiFFTFree_R_32s (IppiFFTSpec_R_32s* pFFTSpec);
IppStatus ippiFFTFree_R_32f (IppiFFTSpec_R_32f* pFFTSpec);
IppStatus ippiFFTFree_C_32fc (IppiFFTSpec_C_32fc* pFFTSpec);
```

## Parameters

`pFFTSpec`                      Pointer to the FFT context structure that has to be released.

## Description

The function `ippiFFTFree` is declared in the `ippi.h` file. This function releases memory used by the previously initialized FFT context structure `pFFTSpec`.

Call this function after your application program has finished computing the fast Fourier transforms.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <code>pFFTSpec</code> pointer is <code>NULL</code> .
<code>ippStsContextMatchErr</code>	Indicates an error condition if a pointer to an invalid <code>pFFTSpec</code> structure is passed.

## FFTGetBufSize

*Determines the size of external work buffer that can be used by the FFT functions.*

---

### Syntax

```
IppStatus ippiFFTGetBufSize_R_32s (IppiFFTSpec_R_32s* pFFTSpec, int* pBufferSize);
IppStatus ippiFFTGetBufSize_R_32f (IppiFFTSpec_R_32f* pFFTSpec, int* pBufferSize);
IppStatus ippiFFTGetBufSize_C_32fc (IppiFFTSpec_C_32fc* pFFTSpec, int* pBufferSize);
```

### Parameters

<i>pFFTSpec</i>	Pointer to the previously initialized FFT context structure.
<i>pBufferSize</i>	Pointer to the size of the external work buffer to be used by the FFT functions.

### Description

The function `ippiFFTGetBufSize` is declared in the `ippi.h` file. This function determines the size in bytes of an external memory buffer that can be used by FFT functions with context *pFFTSpec* as a workspace during calculations. If you want to use external buffering, call this function after *pFFTSpec* initialization. Use the computed *pBufferSize* value to set the size in bytes of an external buffer to be allocated by means of the functions `ippMalloc` or `ippsMalloc_8u`. Note that the allocated memory can be freed only by the functions `ippFree` or `ippsFree` respectively. Refer to chapter 3 "Support Functions" of the *Intel IPP Reference Manual, vol.1* for more details about the memory allocation functions.

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pFFTSpec</i> or <i>pBufferSize</i> pointer is NULL.
<code>ippStsContextMatchErr</code>	Indicates an error condition if a pointer to an invalid <i>pFFTSpec</i> structure is passed.

## FFTfwd

*Applies forward Fast Fourier Transform to an image.*

---

### Syntax

#### Case 1: Not-in-place operation on integer data

```
IppStatus ippiFFTfwd_RToPack_<mod> (const Ipp8u* pSrc, int srcStep, Ipp32s* pDst, int dstStep, const IppiFFTSpec_R_32s* pFFTSpec, int scaleFactor, Ipp8u* pBuffer);
```

Supported values for `mod`:

`8u32s_C1RSfs`

8u32s\_C3RSfs

8u32s\_C4RSfs

8u32s\_AC4RSfs

### Case 2: Not-in-place operation on floating-point data

```
IppStatus ippiFFTFwd_RToPack_<mod> (const Ipp32f* pSrc, int srcStep, Ipp32f* pDst, int
dstStep, const IppiFFTSpec_R_32f* pFFTSpec, Ipp8u* pBuffer);
```

Supported values for mod:

32f\_C1R

32f\_C3R

32f\_C4R

32f\_AC4R

### Case 3: Not-in-place operation on complex data

```
IppStatus ippiFFTFwd_CToC_32fc_C1R(const Ipp32fc* pSrc, int srcStep, Ipp32fc* pDst, int
dstStep, const IppiFFTSpec_C_32fc* pFFTSpec, Ipp8u* pBuffer);
```

### Case 4: In-place operation on floating-point data

```
IppStatus ippiFFTFwd_RToPack_<mod>(Ipp32f* pSrcDst, int srcDstStep, const IppiFFTSpec_R_32f*
pFFTSpec, Ipp8u* pBuffer);
```

Supported values for mod:

32f\_C1IR

32f\_C3IR

32f\_C4IR

32f\_AC4IR

### Case 5: In-place operation on complex data

```
IppStatus ippiFFTFwd_CToC_32fc_C1IR(Ipp32fc* pSrcDst, int srcDstStep, const
IppiFFTSpec_C_32fc* pFFTSpec, Ipp8u* pBuffer);
```

## Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.

<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>pSrcDst</i>	Pointer to the source and destination image ROI for the in-place operation.
<i>srcDstStep</i>	Distance in bytes between starts of consecutive lines in the source and destination image for the in-place operation.
<i>scaleFactor</i>	Scale factor (see <a href="#">Integer Result Scaling</a> ).
<i>pBuffer</i>	Pointer to the external work buffer, can be NULL.

## Description

The function `ippiFFTFwd` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function performs a forward FFT on each channel of the source image ROI *pSrc* (*pSrcDst* for in-place flavors) and writes the Fourier coefficients into the corresponding channel of the destination buffer *pDst* (*pSrcDst* for in-place flavors). The size of ROI is  $N \times M$ , it is specified by the parameters *orderX*, *orderY* calling the function `ippiFFTInitAlloc`.

The function flavor `ippiFFTFwd_RToPack` that operates on images with real data takes advantage of the symmetry property and stores the output data in [RCPack2D format](#). It supports processing of the 1-, 3-, and 4-channel images. Note that the functions with *AC4* descriptor do not process alpha channel.

The function flavor `ippiFFTFwd_CToC` that operates on images with complex data does not perform any packing of the transform results as no symmetry with respect to frequency domain data is observed in this case. Memory layout of images with complex data follows the same conventions as for real images provided that each pixel value consists of two numbers: imaginary and real part.

The forward FFT functions use the previously initialized *pFFTSpec* context structure to set the mode of calculations and retrieve support data.

The function may be used with the external work buffer *pBuffer* to avoid memory allocation within the functions. Once the work buffer is allocated, it can be used for all following calls to the functions computing FFT. As internal allocation of memory is too expensive operation and depends on operating system and/or runtime libraries used - the use of an external buffer improves performance significantly, especially for the small size transforms.

The size of the external buffer must be previously computed by the function `ippiFFTGetBufSize`. If the external buffer is not specified (*pBuffer* is set to NULL), the function itself allocates the memory needed for operation.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pSrc</i> , <i>pDst</i> , or <i>pFFTSpec</i> pointer is NULL.
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> or <i>dstStep</i> value is zero or negative.
<code>ippStsContextMatchErr</code>	Indicates an error condition if a pointer to an invalid <i>pFFTSpec</i> structure is passed.
<code>ippStsMemAllocErr</code>	Indicates an error condition if memory allocation fails.

## Example

The code example below illustrates the use of the forward FFT function for processing real data.

```
IppStatus fft( void )
{
    Ipp32f src[64] = {0}, dst[64];
    IppiFFTSpec_R_32f *spec;
    IppStatus status;
    src[0] = -3; src[9] = 1;
    ippiFFTInitAlloc_R_32f(&spec, 3, 3, IPP_FFT_DIV_INV_BY_N, ippiAlgHintAccurate);
    status = ippiFFTForward_RToPack_32f_C1R( src, 8*sizeof(Ipp32f), dst,
                                              8*sizeof(Ipp32f), spec, 0 );

    ippiFFTFree_R_32f( spec );
    return status;
}
```

On exit from the ippiFFTForward\_RToPack function, the destination buffer contains the following data in RCPack2D format:

```
-2.00 -2.29 -0.71 -3.00 -1.00 -3.71 -0.71 -4.00
-2.29 -3.00 -1.00 -3.71 -0.71 -4.00 +0.00 -3.71
-0.71 -3.71 -0.71 -4.00 +0.00 -3.71 +0.71 +0.71
-3.00 -4.00 +0.00 -3.71 +0.71 -3.00 +1.00 -3.00
-1.00 -3.71 +0.71 -3.00 +1.00 -2.29 +0.71 +1.00
-3.71 -3.00 +1.00 -2.29 +0.71 -2.00 +0.00 -2.29
-0.71 -2.29 +0.71 -2.00 +0.00 -2.29 -0.71 +0.71
-4.00 -2.00 +0.00 -2.29 -0.71 -3.00 -1.00 -2.00
```

The code example below explains FFT processing of complex data. Note that input values are the same as in the previous example, but specified in complex domain.

```
IppStatus fft_cplx( void ) {
    Ipp32fc src[64] = {0}, dst[64], m3 = {-3,0}, one = {1,0};
    IppiFFTSpec_C_32fc *spec;
    IppStatus status;
    src[0] = m3; src[9] = one;
    ippiFFTInitAlloc_C_32fc( &spec, 3, 3, IPP_FFT_DIV_INV_BY_N, ippiAlgHintAccurate);
    status = ippiFFTForward_CToC_32fc_C1R( src, 8*sizeof(Ipp32fc), dst,
                                              8*sizeof(Ipp32fc), spec, 0 );

    ippiFFTFree_C_32fc( spec );
    return status;
}
```

On exit from the ippiFFTForward\_CToC function, the destination buffer contains the following data (real and imaginary part of complex numbers are given in comma-delimited format):

```
-2.0, 0.0 -2.3,-0.7 -3.0,-1.0 -3.7,-0.7 -4.0, 0.0 -3.7, 0.7 -3.0, 1.0 -2.3, 0.7
-2.3,-0.7 -3.0,-1.0 -3.7,-0.7 -4.0,-0.0 -3.7, 0.7 -3.0, 1.0 -2.3, 0.7 -2.0, 0.0
-3.0,-1.0 -3.7,-0.7 -4.0, 0.0 -3.7, 0.7 -3.0, 1.0 -2.3, 0.7 -2.0,-0.0 -2.3,-0.7
-3.7,-0.7 -4.0, 0.0 -3.7, 0.7 -3.0, 1.0 -2.3, 0.7 -2.0,-0.0 -2.3,-0.7 -3.0,-1.0
-4.0, 0.0 -3.7, 0.7 -3.0, 1.0 -2.3, 0.7 -2.0, 0.0 -2.3,-0.7 -3.0,-1.0 -3.7,-0.7
-3.7, 0.7 -3.0, 1.0 -2.3, 0.7 -2.0, 0.0 -2.3,-0.7 -3.0,-1.0 -3.7,-0.7 -4.0,-0.0
-3.0, 1.0 -2.3, 0.7 -2.0, 0.0 -2.3,-0.7 -3.0,-1.0 -3.7,-0.7 -4.0,-0.0 -3.7, 0.7
-2.3, 0.7 -2.0,-0.0 -2.3,-0.7 -3.0,-1.0 -3.7,-0.7 -4.0, 0.0 -3.7, 0.7 -3.0, 1.0
```

## FFTInv

*Applies an inverse FFT to complex source data and stores results in a destination image.*

---

### Syntax

#### Case 1: Not-in-place operation on integer data

```
IppStatus ippiFFTInv_PackToR_<mod>(const Ipp32s* pSrc, int srcStep, Ipp8u* pDst, int dstStep, const IppiFFTSpec_R_32s* pFFTSpec, int scaleFactor, Ipp8u* pBuffer);
```

Supported values for mod:

32s8u\_C1RSfs

32s8u\_C3RSfs

32s8u\_C4RSfs

32s8u\_AC4RSfs

#### Case 2: Not-in-place operation on floating-point data

```
IppStatus ippiFFTInv_PackToR_<mod>(const Ipp32f* pSrc, int srcStep, Ipp32f* pDst, int dstStep, const IppiFFTSpec_R_32f* pFFTSpec, Ipp8u* pBuffer);
```

Supported values for mod:

32f\_C1R

32f\_C3R

32f\_C4R

32f\_AC4R

#### Case 3: Not-in-place operation on complex data

```
IppStatus ippiFFTInv_CToC_32fc_C1R(const Ipp32fc* pSrc, int srcStep, Ipp32fc* pDst, int dstStep, const IppiFFTSpec_C_32fc* pFFTSpec, Ipp8u* pBuffer);
```

#### Case 4: In-place operation on floating-point data

```
IppStatus ippiFFTInv_PackToR_<mod>(Ipp32f* pSrcDst, int srcDstStep, const IppiFFTSpec_R_32f* pFFTSpec, Ipp8u* pBuffer);
```

Supported values for mod:

32f\_C1IR

32f\_C3IR

32f\_C4IR

32f\_AC4IR

### Case 5: In-place operation on complex data

```
IppStatus ippiFFTInv_CToC_32fc_C1IR(Ipp32fc* pSrcDst, int srcDstStep, const
IppiFFTSpec_C_32fc* pFFTSpec, Ipp8u* pBuffer);
```

### Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>pSrcDst</i>	Pointer to the source and destination image ROI for the in-place operation.
<i>srcDstStep</i>	Distance in bytes between starts of consecutive lines in the source and destination image for the in-place operation.
<i>pFFTSpec</i>	Pointer to the previously initialized FFT context structure.
<i>scaleFactor</i>	Scale factor (see <a href="#">Integer Result Scaling</a> ).
<i>pBuffer</i>	Pointer to the external work buffer, can be NULL.

### Description

The function `ippiFFTInv` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function performs an inverse FFT on each channel of the source image *pSrc* (*pSrcDst* for in-place flavors) and writes the restored image data into the corresponding channel of the destination image buffer *pDst* (*pSrcDst* for in-place flavors). The size of ROI is  $N \times M$ , it is specified by the parameters *orderX*, *orderY* calling the function `ippiFFTInitAlloc`.

For function flavor `ippiFFTInv_PackToR`, the input buffer must contain data in [RCPack2D format](#).

The inverse FFT functions use the previously initialized *pFFTSpec* context structure to set the mode of calculations and retrieve support data.

The function may be used with the external work buffer *pBuffer* to avoid memory allocation within the functions. Once the work buffer is allocated, it can be used for all following calls to the functions computing FFT. As internal allocation of memory is too expensive operation and depends on operating system and/or runtime libraries used - the use of an external buffer improves performance significantly, especially for the small size transforms.

The size of the external buffer must be previously computed by the function `ippiFFTGetBufSize`. If the external buffer is not specified (*pBuffer* is set to NULL), the function itself allocates the memory needed for operation.

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
--------------------------	--

<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pSrc</i> , <i>pDst</i> , or <i>pFFTSpec</i> pointer is <code>NULL</code> .
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> or <i>dstStep</i> value is zero or negative.
<code>ippStsContextMatchErr</code>	Indicates an error condition if a pointer to an invalid <i>pFFTSpec</i> structure is passed.
<code>ippStsMemAllocErr</code>	Indicates an error condition if memory allocation fails.

## DFTInitAlloc

*Allocates memory and fills in context data needed for the image DFT functions to operate.*

---

### Syntax

```

IppStatus ippIDFTInitAlloc_R_32s (IppiDFTSpec_R_32s** ppDFTSpec, IppiSize roiSize, int
flag, IppHintAlgorithm hint);

IppStatus ippIDFTInitAlloc_R_32f (IppiDFTSpec_R_32f** ppDFTSpec, IppiSize roiSize, int
flag, IppHintAlgorithm hint);

IppStatus ippIDFTInitAlloc_C_32fc (IppiDFTSpec_C_32fc** ppDFTSpec, IppiSize roiSize, int
flag, IppHintAlgorithm hint);

```

### Parameters

<i>ppDFTSpec</i>	Pointer to pointer to the DFT context structure being initialized.
<i>roiSize</i>	Size of the source and destination ROI in pixels.
<i>flag</i>	Flag to choose the results normalization option.
<i>hint</i>	Option to select the algorithmic implementation of the transform function (see <a href="#">Table "Hint Arguments for Linear Transform Functions"</a> ).

### Description

The function `ippIDFTInitAlloc` is declared in the `ippi.h` file. This function allocates memory and initializes the context structure *pDFTSpec* needed to compute the forward and inverse DFT of a two-dimensional image data.

The `ippIDFTFwd` and `ippIDFTInv` functions called with the pointer to the initialized *pDFTSpec* structure as an argument will compute the discrete Fourier transform for points in the ROI of size *roiSize*, with results normalization mode set according to *flag* (see [Table "Normalization Factors for Fourier Transform Results"](#)), and computation algorithm indicated by *hint*.

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pDFTSpec</i> pointer is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.



<code>ippStsFFTFflagErr</code>	Indicates an error condition if <i>flag</i> has an illegal value.
<code>ippStsMemAllocErr</code>	Indicates an error condition if memory allocation fails.

## DFTFree

Deallocates memory used by the DFT context structure.

### Syntax

```
IppStatus ippIDFTFree_R_32s (IppIDFTSpec_R_32s* pDFTSpec);
IppStatus ippIDFTFree_R_32f (IppIDFTSpec_R_32f* pDFTSpec);
IppStatus ippIDFTFree_C_32fc (IppIDFTSpec_C_32fc* pDFTSpec);
```

### Parameters

*pDFTSpec*                      Pointer to the DFT context structure that has to be released

### Description

The function `ippIDFTFree` is declared in the `ippi.h` file. This function releases memory used by the previously initialized DFT context structure *pDFTSpec*. Call this function after your application program has finished computing the discrete Fourier transforms.

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pDFTSpec</i> pointer is <code>NULL</code> .
<code>ippStsContextMatchErr</code>	Indicates an error condition if a pointer to an invalid <i>pDFTSpec</i> structure is passed.

## DFTGetBufSize

Determines the size of work buffer that can be used by the DFT functions.

### Syntax

```
IppStatus ippIDFTGetBufSize_R_32s(IppIDFTSpec_R_32s* pDFTSpec, int* pBufferSize);
IppStatus ippIDFTGetBufSize_R_32f(IppIDFTSpec_R_32f* pDFTSpec, int* pBufferSize);
IppStatus ippIDFTGetBufSize_C_32fc(IppIDFTSpec_C_32fc* pDFTSpec, int* pBufferSize);
```

### Parameters

<i>pDFTSpec</i>	Pointer to the previously initialized DFT context structure.
<i>pBufferSize</i>	Pointer to the size of the external work buffer to be used by the DFT functions.

## Description

The function `ippiDFTGetBufSize` is declared in the `ippi.h` file. This function determines the size in bytes of an external memory buffer that can be used by DFT functions with context `pDFTSpec` as a workspace during calculations.

If you want to use external buffering, call this function after `pDFTSpec` initialization. Use the computed `pBufferSize` value as the size in bytes of an external buffer to be allocated by means of the functions `ippMalloc` or `ippsMalloc_8u`. Note that the allocated memory can be freed only by the functions `ippFree` or `ippsFree` respectively. Refer to the chapter 3 “Support Functions” of the *Intel IPP Reference Manual*, vol.1 for more details about the memory allocation functions.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <code>pDFTSpec</code> pointer is <code>NULL</code> .
<code>ippStsContextMatchErr</code>	Indicates an error condition if a pointer to an invalid <code>pDFTSpec</code> structure is passed.

## DFTFwd

Applies forward discrete Fourier transform to an image.

### Syntax

#### Case 1: Not-in-place operation on integer data

```
ippStatus ippiDFTFwd_RToPack_<mod> (const Ipp8u* pSrc, int srcStep, Ipp32s* pDst, int dstStep, const IppiDFTSpec_R_32s* pDFTSpec, int scaleFactor, Ipp8u* pBuffer);
```

Supported values for `mod`:

`8u32s_C1RSfs`

`8u32s_C3RSfs`

`8u32s_C4RSfs`

`8u32s_AC4RSfs`

#### Case 2: Not-in-place operation on floating-point data

```
ippStatus ippiDFTFwd_RToPack_<mod>(const Ipp32f* pSrc, int srcStep, Ipp32f* pDst, int dstStep, const IppiDFTSpec_R_32f* pDFTSpec, Ipp8u* pBuffer);
```

Supported values for `mod`:

`32f_C1R`

`32f_C3R`

`32f_C4R`

32f\_AC4R

### Case 3: Not-in-place operation on complex data

```
IppStatus ippiDFTFwd_CToC_32fc_C1R(const Ipp32fc* pSrc, int srcStep, Ipp32fc* pDst, int dstStep, const IppiDFTSpec_C_32fc* pDFTSpec, Ipp8u* pBuffer);
```

### Case 4: In-place operation on floating-point data

```
IppStatus ippiDFTFwd_RToPack_<mod>(Ipp32f* pSrcDst, int srcDstStep, const IppiDFTSpec_R_32f* pDFTSpec, Ipp8u* pBuffer);
```

Supported values for `mod`:

32f\_C1IR

32f\_C3IR

32f\_C4IR

32f\_AC4IR

### Case 5: In-place operation on complex data

```
IppStatus ippiDFTFwd_CToC_32fc_C1IR(Ipp32fc* pSrcDst, int srcDstStep, const IppiDFTSpec_C_32fc* pDFTSpec, Ipp8u* pBuffer);
```

## Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>pSrcDst</i>	Pointer to the source and destination image ROI for the in-place operation.
<i>srcDstStep</i>	Distance in bytes between starts of consecutive lines in the source and destination image for the in-place operation.
<i>pDFTSpec</i>	Pointer to the previously initialized DFT context structure.
<i>scaleFactor</i>	Scale factor (see <a href="#">Integer Result Scaling</a> ).
<i>pBuffer</i>	Pointer to the external work buffer, can be <code>NULL</code> .

## Description

The function `ippiDFTFwd` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)) of the size specified by the function `ippiDFTInitAlloc`.

This function performs a forward DFT on each channel of the source image ROI *pSrc* (*pSrcDst* for in-place flavors) and writes the Fourier coefficients into the corresponding channel of the destination buffer *pDst* (*pSrcDst* for in-place flavors).

The function flavor `ippiDFTFwd_RToPack` that operates on images with real data takes advantage of the symmetry property and stores the output data in [RCPack2D format](#). It supports processing of the 1-, 3-, and 4-channel images. Note that the functions with `AC4` descriptor do not process alpha channel.

The function flavor `ippiDFTFwd_CToC` that operates on images with complex data performs no packing of the transform results as no symmetry with respect to frequency domain data is observed in this case. Memory layout of images with complex data follows the same conventions as for real images provided that each pixel value consists of two numbers: imaginary and real part.

The forward DFT functions use the previously initialized `pDFTSpec` context structure to set the mode of calculations and retrieve support data.

The function may be used with the external work buffer `pBuffer` to avoid memory allocation within the functions. Once the work buffer is allocated, it can be used for all following calls to the functions computing DFT. As internal allocation of memory is too expensive operation and depends on operating system and/or runtime libraries used - the use of an external buffer improves performance significantly, especially for the small size transforms.

The size of the external buffer must be previously computed by the function [ippiDFTGetBufSize](#). If the external buffer is not specified (`pBuffer` is set to `NULL`), the function itself allocates the memory needed for operation.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <code>pSrc</code> , <code>pDst</code> , or <code>pDFTSpec</code> pointer is <code>NULL</code> .
<code>ippStsStepErr</code>	Indicates an error condition if <code>srcStep</code> or <code>dstStep</code> value is zero or negative.
<code>ippStsContextMatchErr</code>	Indicates an error condition if a pointer to an invalid <code>pDFTSpec</code> structure is passed.
<code>ippStsMemAllocErr</code>	Indicates an error condition if memory allocation fails.

## DFTInv

*Applies an inverse DFT to complex source data and stores results in a destination image.*

---

## Syntax

### Case 1: Not-in-place operation on integer data

```
IppStatus ippiDFTInv_PackToR_<mod>(const Ipp32s* pSrc, int srcStep, Ipp8u* pDst, int dstStep, const IppiDFTSpec_R_32s* pDFTSpec, int scaleFactor, Ipp8u* pBuffer);
```

Supported values for `mod`:

`32s8u_C1RSfs`

`32s8u_C3RSfs`

`32s8u_C4RSfs`

`32s8u_AC4RSfs`

**Case 2: Not-in-place operation on floating-point data**

```
IppStatus ippiDFTInv_PackToR_<mod>(const Ipp32f* pSrc, int srcStep, Ipp32f* pDst, int
dstStep, const IppiDFTSpec_R_32f* pDFTSpec, Ipp8u* pBuffer);
```

Supported values for mod:

32f\_C1R

32f\_C3R

32f\_C4R

32f\_AC4R

**Case 3: Not-in-place operation on complex data**

```
IppStatus ippiDFTInv_CToC_32fc_C1R(const Ipp32fc* pSrc, int srcStep, Ipp32fc* pDst, int
dstStep, const IppiDFTSpec_C_32fc* pDFTSpec, Ipp8u* pBuffer);
```

**Case 4: In-place operation on floating-point data**

```
IppStatus ippiDFTInv_PackToR_<mod>(Ipp32f* pSrcDst, int srcDstStep, const IppiDFTSpec_R_32f*
pDFTSpec, Ipp8u* pBuffer);
```

Supported values for mod:

32f\_C1IR

32f\_C3IR

32f\_C4IR

32f\_AC4IR

**Case 5: In-place operation on complex data**

```
IppStatus ippiDFTInv_CToC_32fc_C1IR(Ipp32fc* pSrcDst, int srcDstStep, const
IppiDFTSpec_C_32fc* pDFTSpec, Ipp8u* pBuffer);
```

**Parameters**

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>pSrcDst</i>	Pointer to the source and destination image ROI for the in-place operation.
<i>srcDstStep</i>	Distance in bytes between starts of consecutive lines in the source and destination image for the in-place operation.

<i>pDFTSpec</i>	Pointer to the previously initialized DFT context structure.
<i>scaleFactor</i>	Scale factor (see <a href="#">Integer Result Scaling</a> ).
<i>pBuffer</i>	Pointer to the external work buffer, can be <code>NULL</code> .

## Description

The function `ippiDFTInv` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)) of the size specified by the function `ippiDFTInitAlloc`.

This function performs an inverse DFT on each channel of the input buffer *pSrc* (*pSrcDst* for in-place flavors) and writes the restored image data into the corresponding channel of the output image buffer *pDst* (*pSrcDst* for in-place flavors).

For function flavor `ippiDFTInv_PackToR`, the input buffer must contain data in [RCPack2D format](#).

The inverse DFT functions use the previously initialized *pDFTSpec* context structure to set the mode of calculations and retrieve support data.

The function may be used with the external work buffer *pBuffer* to avoid memory allocation within the functions. Once the work buffer is allocated, it can be used for all following calls to the functions computing DFT. As internal allocation of memory is too expensive operation and depends on operating system and/or runtime libraries used - the use of an external buffer improves performance significantly, especially for the small size transforms.

The size of the external buffer must be previously computed by the function `ippiDFTGetBufSize`. If the external buffer is not specified (*pBuffer* is set to `NULL`), the function itself allocates the memory needed for operation.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pSrc</i> , <i>pDst</i> , or <i>pDFTSpec</i> pointer is <code>NULL</code> .
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> or <i>dstStep</i> value is zero or negative.
<code>ippStsContextMatchErr</code>	Indicates an error condition if a pointer to an invalid <i>pDFTSpec</i> structure is passed.
<code>ippStsMemAllocErr</code>	Indicates an error condition if memory allocation fails.

## MulPack

*Multiplies two source images in packed format.*

---

## Syntax

### Case 1: Not-in-place operation on integer data

```
IppStatus ippiMulPack_<mod>(const Ipp<datatype>* pSrc1, int src1Step, const Ipp<datatype>* pSrc2, int src2Step, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, int scaleFactor);
```

Supported values for `mod`:

```
16s_C1RSfs 32s_C1RSfs
```

16s\_C3RSfs 32s\_C3RSfs

16s\_C4RSfs 32s\_C4RSfs

16s\_AC4RSfs 32s\_AC4RSfs

### Case 2: Not-in-place operation on floating-point data

```
IppStatus ippMulPack_<mod>(const Ipp32f* pSrc1, int src1Step, const Ipp32f* pSrc2, int  
src2Step, Ipp32f* pDst, int dstStep, IppiSize roiSize);
```

Supported values for mod:

32f\_C1R

32f\_C3R

32f\_C4R

32f\_AC4R

### Case 3: In-place operation on integer data

```
IppStatus ippMulPack_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pSrcDst,  
int dstSrcStep, IppiSize roiSize, int scaleFactor);
```

Supported values for mod:

16s\_C1IRSfs 32s\_C1IRSfs

16s\_C3IRSfs 32s\_C3IRSfs

16s\_C4IRSfs 32s\_C4IRSfs

16s\_AC4IRSfs 32s\_AC4IRSfs

### Case 4: In-place operation on floating-point data

```
IppStatus ippMulPack_<mod>(const Ipp32f* pSrc, int srcStep, Ipp32f* pSrcDst, int  
dstSrcStep, IppiSize roiSize);
```

Supported values for mod:

32f\_C1IR

32f\_C3IR

32f\_C4IR

32f\_AC4IR

## Parameters

<i>pSrc1, pSrc2</i>	Pointer to the ROI in the source images.
<i>src1Step, src2Step</i>	Distance in bytes between starts of consecutive lines in the source images.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>pSrc</i>	Pointer to the first source image ROI for the in-place operation.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the first source image for the in-place operation.
<i>pSrcDst</i>	Pointer to the second source and destination image ROI for the in-place operation.
<i>srcDstStep</i>	Distance in bytes between starts of consecutive lines in the source and destination image for the in-place operation.
<i>roiSize</i>	Size of the source and destination ROI in pixels.
<i>scaleFactor</i>	Scale factor (see <a href="#">Integer Result Scaling</a> ).

## Description

The function `ippiMulPack` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function multiplies corresponding pixel values of two source images, *A* and *B* represented in [RCPack2D format](#) and stores the result into the destination image *C* in packed format also. The multiplying is performed according to the following formulas:

$$\text{Re}C = \text{Re}A * \text{Re}B - \text{Im}A * \text{Im}B;$$

$$\text{Im}C = \text{Im}A * \text{Re}B + \text{Im}B * \text{Re}A.$$

Not-in-place flavors multiply pixel values of ROI in the source images *pSrc1* and *pSrc2*, and store result in the *pDst*.

In-place flavors multiply pixel values of ROI in the source images *pSrc* and *pSrcDst*, and store result in the *pSrcDst*.

In case of operations on integer data, the resulting values are scaled by *scaleFactor* (see [Integer Result Scaling](#)). Note that the functions with *AC4* descriptor do not process the alpha channel.

This function can be used in image filtering operations that include FFT transforms.



## Example

The code example below illustrates how the horizontal edges of an image can be detected. First, both the source image and the filter are transformed into the frequency domain by applying the forward FFT function which yields results in packed format. Then, actual filtering is performed through multiplying these packed data on a point by point basis using the function `ippiMulPack`. Finally, filtered data is transformed to the time domain by means of the inverse FFT function.

```
IppStatus mulpack( void ) {
    Ipp32f tsrc[64], tflt[64], tdst[64];
    Ipp32f fsrc[64], fflt[64], fdst[64];
    IppiFFTSpec_R_32f *spec;
    const IppiSize roiSize8x8 = { 8, 8 }, roiSize3x3 = { 3, 3 };
    const Ipp32f filter[3*3] = {-1,-1,-1, 0,0,0, 1,1,1};
    ippiSet_32f_C1R( 0, tsrc, 8*sizeof(Ipp32f), roiSize8x8 );
    ippiAddC_32f_C1R( 1, tsrc+8+1, 8*sizeof(Ipp32f), roiSize3x3 );
    ippiSet_32f_C1R( 0, tflt, 8*sizeof(Ipp32f), roiSize8x8 );
    ippiCopy_32f_C1R( filter, 3*sizeof(Ipp32f), tflt, 8*sizeof(Ipp32f), roiSize3x3 );
    ippiFFTInitAlloc_R_32f( &spec, 3, 3, IPP_FFT_DIV_INV_BY_N, ippiAlgHintAccurate );
    ippiFFTFwd_RToPack_32f_C1R( tsrc, 8*sizeof(Ipp32f), fsrc, 8*sizeof(Ipp32f), spec, 0 );
    ippiFFTFwd_RToPack_32f_C1R( tflt, 8*sizeof(Ipp32f), fflt, 8*sizeof(Ipp32f), spec, 0 );
    ippiMulPack_32f_C1R( fsrc, 8*sizeof(Ipp32f), fflt, 8*sizeof(Ipp32f), fdst, 8*4, roiSize8x8 );
    ippiFFTInv_PackToR_32f_C1R( fdst, 8*sizeof(Ipp32f), tdst, 8*sizeof(Ipp32f), spec, 0 );
    ippiFFTFree_R_32f( spec );
    return 0;
}
```

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if any of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <code>roiSize</code> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if any of the specified buffer step values is zero or negative.

## MulPackConj

*Multiplies a source image by the complex conjugate image with data in packed format and stores the result in the destination buffer in the packed format.*

---

### Syntax

#### Case 1: Not-in-place operation

```
IppStatus ippiMulPackConj_<mod>(const Ipp32f* pSrc1, int src1Step, const Ipp32f* pSrc2,
int src2Step, Ipp32f* pDst, int dstStep, IppiSize roiSize);
```

Supported values for *mod*:

32f\_C1R

32f\_C3R

32f\_C4R

32f\_AC4R

#### Case 2: In-place operation

```
IppStatus ippiMulPackConj_<mod>(const Ipp32f* pSrc, int srcStep, Ipp32f* pSrcDst, int
srcDstStep, IppiSize roiSize);
```

Supported values for *mod*:

32f\_C1IR

32f\_C3IR

32f\_C4IR

32f\_AC4IR

### Parameters

<i>pSrc1, pSrc2</i>	Pointer to the ROI in the source images.
<i>src1Step, src2Step</i>	Distance in bytes between starts of consecutive lines in the source images.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>pSrc</i>	Pointer to the first source image ROI for the in-place operation.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the first source image for the in-place operation.

<i>pSrcDst</i>	Pointer to the second source and destination image ROI for the in-place operation.
<i>srcDstStep</i>	Distance in bytes between starts of consecutive lines in the source and destination image for the in-place operation.
<i>roiSize</i>	Size of the source and destination ROI in pixels.
<i>scaleFactor</i>	Scale factor (see <a href="#">Integer Result Scaling</a> ).

## Description

The function `ippiMulPackConj` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function multiplies pixel values of the source image *A* by the corresponding pixel values of the complex conjugate image *A\**, represented in [RCPack2D format](#). The result of the operation is written into the destination buffer in packed format also.

Not-in-place flavors multiply pixel values of ROI in the source images *pSrc1* and *pSrc2*, and store result in the *pDst*.

In-place flavors multiply pixel values of ROI in the source images *pSrc* and *pSrcDst*, and store result in the *pSrcDst*.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if any of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if any of the specified buffer step values is zero or negative.

## Magnitude

*Computes magnitude of elements of a complex data image.*

## Syntax

```
ippStatus ippiMagnitude_<mod>(const Ipp<srcDatatype>* pSrc, int srcStep, Ipp<dstDatatype>* pDst, int dstStep, IppiSize roiSize);
```

Supported values for `mod`:

`32fc32f_C1R 32fc32f_C3R`

```
ippStatus ippiMagnitude_<mod>(const Ipp<srcDatatype>* pSrc, int srcStep, Ipp<dstDatatype>* pDst, int dstStep, IppiSize roiSize, int scaleFactor);
```

Supported values for `mod`:

`16uc16u_C1RSfs 16uc16u_C3RSfs`

16sc16s\_C1RSfs 16sc16s\_C3RSfs

32sc32s\_C1RSfs 32sc32s\_C3RSfs

## Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>roiSize</i>	Size of the source and destination ROI in pixels.
<i>scaleFactor</i>	Scale factor (see <a href="#">Integer Result Scaling</a> ).

## Description

The function `ippiMagnitude` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function computes magnitude of elements of the source image *pSrc* given in complex data format, and stores results in the destination image *pDst*.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pSrc</i> or <i>pDst</i> pointer is <code>NULL</code> .
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> or <i>dstStep</i> value is zero or negative.
<code>ippStsSizeErr</code>	Indicates an error condition if width or height of images is less than or equal to zero.

## MagnitudePack

*Computes magnitude of elements of an image in packed format.*

---

## Syntax

```
ippStatus ippiMagnitudePack_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize dstRoiSize);
```

Supported values for `mod`:

32f\_C1R 32f\_C3R

```
IppStatus ippMagnitudePack_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>*  
pDst, int dstStep, IppiSize dstRoiSize, int scaleFactor);
```

Supported values for `mod`:

```
16s_C1RSfs 16s_C3RSfs  
32s_C1RSfs 32s_C3RSfs
```

## Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>dstRoiSize</i>	Size of the destination ROI in pixels.
<i>scaleFactor</i>	Scale factor (see <a href="#">Integer Result Scaling</a> ).

## Description

The function `ippMagnitudePack` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function computes magnitude of elements of the source image *pSrc* given in [RCPack2D format](#) and stores results in the destination image *pDst*.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pSrc</i> or <i>pDst</i> pointer is <code>NULL</code> .
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> or <i>dstStep</i> value is zero or negative.
<code>ippStsSizeErr</code>	Indicates an error condition if width or height of images is less than or equal to zero.

## Phase

*Computes the phase of elements of a complex data image.*

---

### Syntax

```
IppStatus ippiPhase_<mod>(const Ipp<srcDatatype>* pSrc, int srcStep, Ipp<dstDatatype>* pDst, int dstStep, IppiSize roiSize);
```

Supported values for `mod`:

32fc32f\_C1R 32fc32f\_C3R

```
IppStatus ippiPhase_<mod>(const Ipp<srcDatatype>* pSrc, int srcStep, Ipp<dstDatatype>* pDst, int dstStep, IppiSize roiSize, int scaleFactor);
```

Supported values for `mod`:

16uc16u\_C1RSfs 16uc16u\_C3RSfs

16sc16s\_C1RSfs 16sc16s\_C3RSfs

32sc32s\_C1RSfs 32sc32s\_C3RSfs

### Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>roiSize</i>	Size of the source and destination ROI in pixels.
<i>scaleFactor</i>	Scale factor (see <a href="#">Integer Result Scaling</a> ).

### Description

The function `ippiPhase` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function computes the phase in radians of elements of a source image *pSrc* given in complex data format, and stores results in the destination image *pDst*.

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pSrc</i> or <i>pDst</i> pointer is <code>NULL</code> .
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> or <i>dstStep</i> value is zero or negative.

`ippStsSizeErr`

Indicates an error condition if width or height of images is less than or equal to zero.

## PhasePack

*Computes the phase of elements of an image in packed format.*

---

### Syntax

```
ippStatus ippPhasePack_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pDst,
int dstStep, IppiSize dstRoiSize);
```

Supported values for mod:

`32f_C1R`      `32f_C3R`

```
ippStatus ippPhasePack_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pDst,
int dstStep, IppiSize dstRoiSize, int scaleFactor);
```

Supported values for mod:

`16s_C1RSfs`   `32s_C1RSfs`

`16s_C3RSfs`   `32s_C3RSfs`

### Parameters

<code>pSrc</code>	Pointer to the source image ROI.
<code>srcStep</code>	Distance in bytes between starts of consecutive lines in the source image.
<code>pDst</code>	Pointer to the destination image ROI.
<code>dstStep</code>	Distance in bytes between starts of consecutive lines in the destination image.
<code>dstRoiSize</code>	Size of the destination ROI in pixels.
<code>scaleFactor</code>	The factor for integer result scaling (see <a href="#">Integer Result Scaling</a> ).

### Description

The function `ippPhasePack` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function computes the phase of elements of a source image `pSrc` given in [RCPack2D format](#) and stores results in the destination image `pDst`.

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <code>pSrc</code> or <code>pDst</code> pointer is NULL.

<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> or <i>dstStep</i> value is zero or negative.
<code>ippStsSizeErr</code>	Indicates an error condition if width or height of images is less than or equal to zero.

## PolarToCart

*Converts an image in the polar coordinate form to Cartesian coordinate form.*

---

### Syntax

```
IppStatus ippipolarToCart_<mod>(const Ipp32f* pSrcMagn, const Ipp32f* pSrcPhase, int
srcStep, IppiSize roiSize, Ipp32fc* pDst, int dstStep);
```

Supported values for *mod*:

`32fc_C1R`    `32fc_C3R`

```
IppStatus ippipolarToCart_<mod>(const Ipp<sourceDatatype>* pSrcMagn, const
Ipp<sourceDatatype>* pSrcPhase, int srcStep, int phaseFixedPoint, IppiSize roiSize,
Ipp<dstDatatype>* pDst, int dstStep);
```

Supported values for *mod*:

`16sc_C1R`    `16sc_C3R`

`32sc_C1R`    `32sc_C3R`

### Parameters

<i>pSrcMagn</i>	Pointer to the buffer containing magnitudes of the source image.
<i>pSrcPhase</i>	Pointer to the buffer containing phase values of the source image.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source buffers.
<i>phaseFixedPoint</i>	Specified the position of the decimal fixed point for the phase.
<i>roiSize</i>	Size of the source and destination image ROI.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image buffer.

### Description

The function `ippipolarToCart` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).



This function converts the polar coordinate of the source image stored in the arrays of magnitudes *pSrcMagn* and phase values *pSrcPhase* to the destination image *pDst* in complex-data format (in Cartesian coordinate form).

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointers is <code>NULL</code> .
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> or <i>dstStep</i> value is zero or negative.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>srcSize</i> has a field with value less than 1.

### Example

The code example below shows how to use the function `ippiPolarToCart_32fc_C1R`.

```
void func_polartocart()
{
    Ipp32f pSrcMagn[2*2] = {1.0, 0.0, 2.1, 3.2};

    Ipp32f pSrcPhase[2*2] = {0.0, 2.0, 1.6,-1.0};
    Ipp32fc pDst[2*2] = {0};
    int srcStep = 2*sizeof(Ipp32f);
    int dstStep = 2*sizeof(Ipp32fc);
    IppiSize roiSize = {2, 2};

    ippiPolarToCart_32fc_C1R(pSrcMagn, pSrcPhase, srcStep, roiSize, pDst, dstStep);
}
```

Result: *pDst* - > (1.0, 0.0) (0.0, 0.0) (-0.1, 2.1) (1.7, -2.7)

## PackToCplxExtend

*Converts an image in packed format to a complex data image.*

### Syntax

```
IppStatus ippiPackToCplxExtend_32s32sc_C1R(const Ipp32s* pSrc, IppiSize srcSize, int
srcStep, Ipp32sc* pDst, int dstStep);
```

```
IppStatus ippiPackToCplxExtend_32f32fc_C1R(const Ipp32f* pSrc, IppiSize srcSize, int
srcStep, Ipp32fc* pDst, int dstStep);
```

### Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcSize</i>	Size in pixels of the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source buffer.
<i>pDst</i>	Pointer to the destination image buffer.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image buffer.

### Description

The function `ippiPackToCplxExtend` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts the source image *pSrc* in [RCPack2D format](#) to complex data format and stores the results in *pDst*, which is a matrix with complete set of the Fourier coefficients. Note that if the *pSrc* in RCPack2D format is a real array of dimensions (NXM), then the *pDst* is a real array of dimensions (2xNXM). This should be taken into account when allocating memory for the function operation.

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pSrc</i> or <i>pDst</i> pointer is NULL.
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> or <i>dstStep</i> value is zero or negative.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>srcSize</i> has field with zero or negative value.

## CplxExtendToPack

*Converts a complex data image to an image in packed format.*

---

### Syntax

```
IppStatus ippiCplxExtendToPack_<mod>(const Ipp<srcDatatype>* pSrc, int srcStep, IppiSize
srcSize, Ipp<dstDatatype>* pDst, int dstStep);
```

Supported values for `mod`:

16sc16s_C1R	32sc32s_C1R	32fc32f_C1R
16sc16s_C3R	32sc32s_C3R	32fc32f_C3R

## Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcSize</i>	Size in pixels of the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source buffer.
<i>pDst</i>	Pointer to the destination image buffer.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image buffer.

## Description

The function `ippiCplxExtendToPack` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts the source image *pSrc* in complex data format to [RCPack2D format](#) and stores the results in *pDst*, which is a real array of dimensions (NXM). The *pSrc* is a matrix with complete set of the Fourier coefficients.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pSrc</i> or <i>pDst</i> pointer is <code>NULL</code> .
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> or <i>dstStep</i> value is zero or negative.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>srcSize</i> has field with zero or negative value.

## Windowing Functions

This section describes Intel IPP windowing functions used in image processing. A window is a mathematical function by which pixel values are multiplied to prepare an image for the subsequent analysis. This procedure is often called 'windowing'. In fact, a window function approaches zero towards the edges of the image avoiding strong distortions of spectral densities in the Fourier domain.

The Intel IPP provides two following types of window functions:

- Bartlett window function
- Hamming window function

These functions generate the window samples and applied them to the specified image. To obtain the window samples themselves, you should apply the desired function to the image with all pixel values set to 1.0. As the windowing operation is very time consuming, it may be useful if you want to apply the same window to the multiple images. In this case use one of the image multiplication functions (`ippiMul`) to multiply the pixel values of the image by the window samples.

## WinBartlett, WinBartlettSep

*Applies Bartlett window function to the image.*

---

### Syntax

#### Case 1: Not-in-place operation

```
IppStatus ippiWinBartlett_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);
```

Supported values for mod:

8u\_C1R            16u\_C1R            32f\_C1R

```
IppStatus ippiWinBartlettSep_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);
```

Supported values for mod:

8u\_C1R            16u\_C1R            32f\_C1R

#### Case 2: In-place operation

```
IppStatus ippiWinBartlett_<mod>(Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize);
```

Supported values for mod:

8u\_C1IR           16u\_C1IR           32f\_C1IR

```
IppStatus ippiWinBartlettSep_<mod>(Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize);
```

Supported values for mod:

8u\_C1IR           16u\_C1IR           32f\_C1IR

### Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>pSrcDst</i>	Pointer to the source and destination image ROI for the in-place operation.
<i>srcDstStep</i>	Distance in bytes between starts of consecutive lines in the source and destination image for the in-place operation.

*roiSize*

Size of the source and destination ROI in pixels.

## Description

The functions `ippiWinBartlett` and `ippiWinBartlettSep` are declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

These functions compute the Bartlett (triangle) window samples, multiply pixel values of the source image *pSrc* (*pSrcDst* for in-place flavors) with these samples, and store results in the destination image *pDst* (*pSrcDst* for in-place flavors).

The Bartlett window function for one-dimensional case with *M* elements is defined as follows:

$$w_{\text{bartlett}}(i) = \begin{cases} \frac{2i}{M-1}, & 0 \leq i \leq \frac{M-1}{2} \\ 2 - \frac{2i}{M-1}, & \frac{M-1}{2} < i \leq M-1 \end{cases}$$

The `ippiWinBartlettSep` flavor applies the window function successively to the rows and then to the columns of the image.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pSrc</i> or <i>pDst</i> pointer is <code>NULL</code> .
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> or <i>dstStep</i> value is zero or negative.
<code>ippStsSizeErr</code>	Indicates an error condition if width or height of images is less than or equal to zero.

## WinHamming, WinHammingSep

*Applies Hamming window function to the image.*

---

## Syntax

### Case 1: Not-in-place operation

```
IppStatus ippiWinHamming_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);
```

Supported values for *mod*:

8u\_C1R      16u\_C1R      32f\_C1R

```
IppStatus ippiWinHammingSep_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>*
pDst, int dstStep, IppiSize roiSize);
```

Supported values for *mod*:

8u\_C1R            16u\_C1R            32f\_C1R

### Case 2: In-place operation

```
IppStatus ippiWinHamming_<mod>(Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize);
```

Supported values for *mod*:

8u\_C1IR            16u\_C1IR            32f\_C1IR

```
IppStatus ippiWinHammingSep_<mod>(Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize);
```

Supported values for *mod*:

8u\_C1IR            16u\_C1IR            32f\_C1IR

## Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>pSrcDst</i>	Pointer to the source and destination image ROI for the in-place operation.
<i>srcDstStep</i>	Distance in bytes between starts of consecutive lines in the source and destination image for the in-place operation.
<i>roiSize</i>	Size of the source and destination ROI in pixels.

## Description

The functions `ippiWinHamming` and `ippiWinHammingSep` are declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

These functions compute the Bartlett (triangle) window samples, multiply pixel values of the source image *pSrc* (*pSrcDst* for in-place flavors) with these samples, and store results in the destination image *pDst* (*pSrcDst* for in-place flavors).

The Hamming window function for one-dimensional case with *M* elements is defined as follows:

$$w_{\text{hamming}}(n) = 0.54 - 0.46 \cos\left(\frac{2\pi n}{len-1}\right)$$

The `ippiWinHammingSep` flavor applies the window function successively to the rows and then to the columns of the image.

### Return Values

<code>ippiStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippiStsNullPtrErr</code>	Indicates an error condition if <code>pSrc</code> or <code>pDst</code> pointer is <code>NULL</code> .
<code>ippiStsStepErr</code>	Indicates an error condition if <code>srcStep</code> or <code>dstStep</code> value is zero or negative.
<code>ippiStsSizeErr</code>	Indicates an error condition if width or height of images is less than or equal to zero.

## Discrete Cosine Transforms

Discrete Cosine Transform (DCT) of a real 2D image yields output results that are also real, which eliminates the need to use packed format for storing the transformed data. However, forward and inverse DCT functions `ippiDCTFwd` and `ippiDCTInv` need different context data structures to be initialized and filled in prior to their use. Consequently, the required workspace buffer size is different for these functions. In case of using an external buffer, its size must be determined by previously calling the respective support function. DCT functions that use context structures implement the modified computation algorithm proposed in [Rao90].

The DCT functions `ippiDCT8x8Fwd` and `ippiDCT8x8Inv` working on a fixed 8x8 image buffer need no context data or external workspace buffers. Functions `ippiDCT8x8Inv` meet IEEE-1180 standard requirements (see [IEEE]).

Intel IPP Discrete Cosine Transform functions working on a fixed 8x8 image buffer use Feig and Winograd algorithm ([Feig92]) modified for taking advantage of SIMD instructions. For details on algorithms used in DCT transforms and for more references, see [AP922].

### Optimization Notice

Intel's compilers may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include SSE2, SSE3, and SSSE3 instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on microprocessors not manufactured by Intel. Microprocessor-dependent optimizations in this product are intended for use with Intel microprocessors. Certain optimizations not specific to Intel microarchitecture are reserved for Intel microprocessors. Please refer to the applicable product User and Reference Guides for more information regarding the specific instruction sets covered by this notice.

Notice revision #20110804

## DCTFwdInitAlloc

*Allocates memory and fills in context data needed for the forward DCT function to operate.*

### Syntax

```
ippiStatus ippiDCTFwdInitAlloc_32f (IppiDCTFwdSpec_32f** ppDCTSpec, IppiSize roiSize,
ippiHintAlgorithm hint);
```

## Parameters

<i>ppDFTSpec</i>	Pointer to the forward DCT context structure being initialized.
<i>roiSize</i>	Size of the source and destination ROI in pixels.
<i>hint</i>	Option to select the algorithmic implementation of the transform function.

## Description

The function `ippiDCTFwdInitAlloc` is declared in the `ippi.h` file. This function allocates memory and initializes the context structure *pDCTSpec* needed to compute the forward DCT of a two-dimensional image data. The `ippiDCTFwd` function called with the pointer to the initialized *pDCTSpec* structure as an argument will compute the forward discrete cosine transform for points in the ROI of size *roiSize* using computation algorithm indicated by *hint* parameter (see [Table "Hint Arguments for Linear Transform Functions"](#)).

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pDCTSpec</i> pointer is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.
<code>ippStsMemAllocErr</code>	Indicates an error condition if memory allocation fails.

## DCTInvInitAlloc

*Allocates memory and fills in context data needed for the inverse DCT function to operate.*

---

## Syntax

```
IppStatus ippiDCTInvInitAlloc_32f (IppiDCTInvSpec_32f** ppDCTSpec, IppiSize roiSize,
IppHintAlgorithm hint);
```

## Parameters

<i>ppDFTSpec</i>	Pointer to the inverse DCT context structure being initialized.
<i>roiSize</i>	Size of the source and destination ROI in pixels.
<i>hint</i>	Option to select the algorithmic implementation of the transform function.

## Description

The function `ippiDCTInvInitAlloc` is declared in the `ippi.h` file. This function allocates memory and initializes the context structure *pDCTSpec* needed to compute the inverse DCT of a two-dimensional image data. The `ippiDCTInv` function called with the pointer to the initialized *pDCTSpec* structure as an argument will compute the inverse discrete cosine transform for points in the ROI of size *roiSize*, using computation algorithm indicated by *hint* (see [Table "Hint Arguments for Linear Transform Functions"](#)).



## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <code>pDCTSpec</code> pointer is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <code>roiSize</code> has a field with zero or negative value.
<code>ippStsMemAllocErr</code>	Indicates an error condition if memory allocation fails.

## DCTFwdFree

*Deallocates memory used by the forward DCT context structure.*

---

### Syntax

```
IppStatus ippIDCTFwdFree_32f (IppiDCTFwdSpec_32f* pDCTSpec);
```

### Parameters

<code>pDCTSpec</code>	Pointer to the forward DCT context structure that has to be released.
-----------------------	---

### Description

The function `ippIDCTFwdFree` is declared in the `ippi.h` file. This function releases memory used by the previously initialized forward DCT context structure `pDCTSpec`. Call this function after your application program has finished computing the forward discrete cosine transforms.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <code>pDCTSpec</code> pointer is <code>NULL</code> .
<code>ippStsContextMatchErr</code>	Indicates an error condition if a pointer to an invalid <code>pDCTSpec</code> structure is passed.

## DCTInvFree

*Deallocates memory used by the inverse DCT context structure.*

---

### Syntax

```
IppStatus ippIDCTInvFree_32f (IppiDCTInvSpec_32f* pDCTSpec);
```

### Parameters

<code>pDCTSpec</code>	Pointer to the inverse DCT context structure that has to be released.
-----------------------	---

## Description

The function `ippiDCTInvFree` is declared in the `ippi.h` file. This function releases memory used by the previously initialized inverse DCT context structure `pDCTSpec`. Call this function after your application program has finished computing the inverse discrete cosine transforms.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <code>pDCTSpec</code> pointer is <code>NULL</code> .
<code>ippStsContextMatchErr</code>	Indicates an error condition if a pointer to an invalid <code>pDCTSpec</code> structure is passed.

## DCTFwdGetBufSize

*Determines the size of work buffer can be used by the forward DCT function.*

---

## Syntax

```
IppStatus ippiDCTFwdGetBufSize_32f (IppDCTFwdSpec_32f* pDCTSpec, int* pBufferSize);
```

## Parameters

<code>pDCTSpec</code>	Pointer to the previously initialized forward DCT context structure.
<code>pBufferSize</code>	Pointer to the size of the external work buffer to be used by the forward DCT function.

## Description

The function `ippiDCTFwdGetBufSize` is declared in the `ippi.h` file. This function determines the size in bytes of an external memory buffer that can be used by the forward DCT function with context `pDCTSpec` as a workspace during calculations. If you want to use external buffering, call this function after `pDCTSpec` initialization. Use the computed `pBufferSize` value as the size in bytes of an external buffer to be allocated by means of the functions `ippMalloc` or `ippsMalloc_8u`. Note that the allocated memory can be freed only by the functions `ippFree` or `ippsFree` respectively. Refer to the chapter 3 “Support Functions” of the *Intel IPP Reference Manual, vol.1* for more details about the memory allocation functions.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <code>pDCTSpec</code> pointer is <code>NULL</code> .
<code>ippStsContextMatchErr</code>	Indicates an error condition if a pointer to an invalid <code>pDCTSpec</code> structure is passed.

## DCTInvGetBufSize

*Determines the size of work buffer that can be used by the inverse DCT function.*

---

### Syntax

```
IppStatus ippiDCTInvGetBufSize_32f (IppiDCTInvSpec_32f* pDCTSpec, int* pBufferSize);
```

### Parameters

<i>pDCTSpec</i>	Pointer to the previously initialized inverse DCT context structure.
<i>pBufferSize</i>	Pointer to the size of the external work buffer to be used by the inverse DCT function.

### Description

The function `ippiDCTInvGetBufSize` is declared in the `ippi.h` file. This function determines the size in bytes of an external memory buffer that can be used by the inverse DCT function with context *pDCTSpec* as a workspace during calculations.

If you want to use external buffering, call this function after *pDCTSpec* initialization. Use the computed *pBufferSize* value as the size in bytes of an external buffer to be allocated by means of the functions `ippMalloc` or `ippsMalloc_8u`. Note that the allocated memory can be freed only by the functions `ippFree` or `ippsFree` respectively. Refer to the chapter 3 “Support Functions” of the *Intel IPP Reference Manual, vol.1* for more details about the memory allocation functions.

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pDCTSpec</i> pointer is NULL.
<code>ippStsContextMatchErr</code>	Indicates an error condition if a pointer to an invalid <i>pDCTSpec</i> structure is passed.

## DCTFwd

*Applies a forward discrete cosine transform to an image.*

---

### Syntax

```
IppStatus ippiDCTFwd_<mod> (const Ipp32f* pSrc, int srcStep, Ipp32f* pDst, int dstStep, const IppiDCTFwdSpec_32f* pDCTSpec, Ipp8u* pBuffer);
```

Supported values for `mod`:

```
32f_C1R
32f_C3R
32f_C4R
```

32f\_AC4R

## Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>pDCTSpec</i>	Pointer to the previously initialized forward DCT context structure.
<i>pBuffer</i>	Pointer to the external work buffer, can be NULL.

## Description

The function `ippiDCTFwd` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)) of the size that is specified by the function `ippiDCTFwdInitAlloc`.

This function performs a forward DCT on each channel of the source image *pSrc* and writes the result into the corresponding channel of the destination image buffer *pDst*. Note that the function flavor with AC4 descriptor does not process alpha channel. This function uses the previously initialized *pDCTSpec* context structure to set the mode of calculations and retrieve support data.

The function may be used with the external work buffer *pBuffer* to avoid memory allocation within the functions. Once the work buffer is allocated, it can be used for all following calls to the functions computing DCT. As internal allocation of memory is too expensive operation and depends on operating system and/or runtime libraries used - the use of an external buffer improves performance significantly, especially for the small size transforms.

The size of the external buffer must be previously computed by the function `ippiDCTFwdGetBufSize`. If the external buffer is not specified (*pBuffer* is set to NULL), the function itself allocates the memory needed for operation.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pSrc</i> , <i>pDst</i> , or <i>pDCTSpec</i> pointer is NULL.
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> or <i>dstStep</i> value is zero or negative.
<code>ippStsContextMatchErr</code>	Indicates an error condition if a pointer to an invalid <i>pDCTSpec</i> structure is passed.
<code>ippStsMemAllocErr</code>	Indicates an error condition if memory allocation fails.

## DCTInv

Applies an inverse discrete cosine transform to an image.

### Syntax

```
IppStatus ippIDCTInv_<mod> (const Ipp32f* pSrc, int srcStep, Ipp32f* pDst, int dstStep,
const IppiDCTInvSpec_32f* pDCTSpec, Ipp8u* pBuffer);
```

Supported values for mod :

32f\_C1R

32f\_C3R

32f\_C4R

32f\_AC4R

### Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>pDCTSpec</i>	Pointer to the previously initialized inverse DCT context structure.
<i>pBuffer</i>	Pointer to the external work buffer, can be NULL.

### Description

The function `ippIDCTInv` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)) of the size that is specified by the function `ippIDCTInvInitAlloc`.

This function performs an inverse DCT on each channel of the input image *pSrc* and writes the result into the corresponding channel of the output image buffer *pDst*. Note that the function flavor with AC4 descriptor does not process alpha channel. This function uses the previously initialized *pDCTSpec* context structure to set the mode of calculations and retrieve support data.

The function may be used with the external work buffer *pBuffer* to avoid memory allocation within the functions. Once the work buffer is allocated, it can be used for all following calls to the functions computing DCT. As internal allocation of memory is too expensive operation and depends on operating system and/or runtime libraries used - the use of an external buffer improves performance significantly, especially for the small size transforms.

The size of the external buffer must be previously computed by the function `ippIDCTInvGetBufSize`. If the external buffer is not specified (*pBuffer* is set to NULL), the function itself allocates the memory needed for operation.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pSrc</i> , <i>pDst</i> , or <i>pDCTSpec</i> pointer is NULL.
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> or <i>dstStep</i> value is zero or negative.
<code>ippStsContextMatchErr</code>	Indicates an error condition if a pointer to an invalid <i>pDCTSpec</i> structure is passed.
<code>ippStsMemAllocErr</code>	Indicates an error condition if memory allocation fails.

## DCT8x8Fwd

*Performs a forward DCT on a 2D buffer of 8x8 size.*

---

### Syntax

#### Case 1: Not-in-place operation

```
IppStatus ippIDCT8x8Fwd_<mod>(const Ipp<datatype>* pSrc, Ipp<datatype>* pDst);
```

Supported values for *mod*:

`16s_C1`      `32f_C1`

#### Case 2: Not-in-place operation with ROI

```
IppStatus ippIDCT8x8Fwd_<mod>(const Ipp<srcDatatype>* pSrc, int srcStep, Ipp<dstDatatype>* pDst);
```

Supported values for *mod*:

`16s_C1R`      `8u16s_C1R`

#### Case 3: In-place operation

```
IppStatus ippIDCT8x8Fwd_<mod>(Ipp<datatype>* pSrcDst);
```

Supported values for *mod*:

`16s_C1I`      `32f_C1I`

## Parameters

<i>pSrc</i>	Pointer to the source image buffer.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image buffer for operations with ROI.
<i>pDst</i>	Pointer to the destination image buffer.

*pSrcDst*

Pointer to the source and destination image for in-place operations.

## Description

The function `ippiDCT8x8Fwd` is declared in the `ippi.h` file. Some flavors operate with ROI (see [Regions of Interest in Intel IPP](#)).

This function computes the forward discrete cosine transform of short integer or floating-point data in a 2D buffer of 8x8 size. No prerequisites are needed to use this transform function.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pSrc</i> , <i>pDst</i> , or <i>pSrcDst</i> pointer is NULL.
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> value is zero or negative.

## Example

The code example below illustrates the use of `ippiDCT8x8Fwd` function.

```

IppStatus dct16s( void ) {
    Ipp16s x[64] = {0};
    IppiSize roi = {8,8};
    int i;
    for( i=0; i<8; ++i ) {
        ippiSet_16s_C1R( (Ipp16s)i, x+8*i+i, 8*sizeof(Ipp16s), roi );
        --roi.width;
        --roi.height;
    }
    return ippiDCT8x8Fwd_16s_C1I( x );
}

```

The destination image *x* contains:

```

18 -9 -2 -1 -1  0  0  0
-9  7  0  0  0  0  0  0
-2  0  2  0  0  0  0  0
-1  0  0  1  0  0  0  0
 0  0  0  0  1  0  0  0
 0  0  0  0  0  0  0  0
 0  0  0  0  0  0  0  0
 0  0  0  0  0  0  0  0

```

## DCT8x8Inv, DCT8x8Inv\_A10

*Performs an inverse DCT on a 2D buffer of 8x8 size.*

---

### Syntax

#### Case 1: Not-in-place operation

```
IppStatus ippiDCT8x8Inv_<mod>(const Ipp<datatype>* pSrc, Ipp<datatype>* pDst);
```

Supported values for mod:

16s\_C1      32f\_C1

```
IppStatus ippiDCT8x8Inv_A10_16s_C1(const Ipp16s* pSrc, Ipp16s* pDst);
```

#### Case 2: Not-in-place operation with ROI

```
IppStatus ippiDCT8x8Inv_<mod>(const Ipp<srcDatatype>* pSrc, Ipp<dstDatatype>* pDst, int dstStep);
```

Supported values for mod:

16s\_C1R      16s8u\_C1R

#### Case 3: In-place operation

```
IppStatus ippiDCT8x8Inv_<mod>(Ipp<datatype>* pSrcDst);
```

Supported values for mod:

16s\_C1I      32f\_C1I

```
IppStatus ippiDCT8x8Inv_A10_16s_C1I( Ipp16s* pSrcDst);
```

### Parameters

<i>pSrc</i>	Pointer to the source image.
<i>pDst</i>	Pointer to the destination buffer.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination buffer for operations with ROI.
<i>pSrcDst</i>	Pointer to the source and destination image for in-place operations.

### Description

The function `ippiDCT8x8Inv` is declared in the `ippi.h` file. Some flavors operate with ROI (see [Regions of Interest in Intel IPP](#)).

This function computes the inverse discrete cosine transform of data in a 2D buffer of 8x8 size. No prerequisites are needed to use this transform function.





**CAUTION.** Source data for 16s functions must be the result of the forward discrete cosine transform of data from the range [-512, 511] for flavors with A10 modifier (`ippiDCT8x8Inv_A10`), and from the range [-256, 255] for flavors without A10 modifier (`ippiDCT8x8Inv`); they cannot be arbitrary data from the range [-32768, 32767].

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <code>pSrc</code> , <code>pDst</code> , or <code>pSrcDst</code> pointer is NULL.
<code>ippStsStepErr</code>	Indicates an error condition if <code>dstStep</code> value is zero or negative.

## DCT8x8FwdLS

*Performs a forward DCT on a 2D buffer of 8x8 size with prior data conversion and level shift.*

## Syntax

```
IppStatus ippiDCT8x8FwdLS_8u16s_C1R(const Ipp8u* pSrc, int srcStep, Ipp16s* pDst, Ipp16s addVal);
```

## Parameters

<code>pSrc</code>	Pointer to the source image buffer.
<code>pDst</code>	Pointer to the destination buffer.
<code>srcStep</code>	Distance in bytes between starts of consecutive lines in the source image buffer.
<code>addVal</code>	The level shift value.

## Description

The function `ippiDCT8x8FwdLS` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)) that is a 2D buffer of 8x8 size in this case, thus there is no need to specify its size.

This function first converts data in the buffer `pSrc` from unsigned `Ipp8u` type to the signed `Ipp16s` type and then performs level shift operation by adding the constant value `addVal` to each sample. After that, the function performs the forward discrete cosine transform of the modified data. The result is stored in `pDst`.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <code>pSrc</code> or <code>pDst</code> pointer is NULL.
<code>ippStsStepErr</code>	Indicates an error condition if <code>srcStep</code> value is zero or negative.

## DCT8x8InvLSClip

*Performs an inverse DCT on a 2D buffer of 8x8 size with further data conversion and level shift.*

---

### Syntax

```
IppStatus ippiDCT8x8InvLSClip_16s8u_C1R(const Ipp16s* pSrc, Ipp8u* pDst, int dstStep,
Ipp16s addVal, Ipp8u clipDown, Ipp8u clipUp);
```

### Parameters

<i>pSrc</i>	Pointer to the source image buffer.
<i>pDst</i>	Pointer to the destination image buffer.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image buffer.
<i>addVal</i>	The level shift value.
<i>clipDown</i>	The lower bound for the range of output values.
<i>clipUp</i>	The upper bound for the range of output values.

### Description

The function `ippiDCT8x8InvLSClip` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)) that is a 2D buffer of 8x8 size in this case, thus there is no need to specify its size.

This function performs the inverse discrete cosine transform of the buffer *pSrc*. After completing the DCT, this function performs level shift operation by adding the constant value *addVal* to each sample. Finally, the function converts data from the signed `Ipp16s` type to the unsigned `Ipp8u` type. The output data are clipped to the range [*clipDown* .. *clipUp*]. The result is stored in the destination buffer *pDst*.



---

**CAUTION.** Source data for 16s flavors must be the result of the forward discrete cosine transform of data from the range [-256, 255], they cannot be arbitrary data from the range [-32768, 32767].

---

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pSrc</i> or <i>pDst</i> pointer is NULL.
<code>ippStsStepErr</code>	Indicates an error condition if <i>dstStep</i> value is zero or negative.

## DCT8x8Inv\_2x2, DCT8x8Inv\_4x4

*Perform an inverse DCT on a top left quadrant of size 2x2 or 4x4 in the 2D buffer of size 8x8.*

---

### Syntax

```
IppStatus ippiDCT8x8Inv_2x2_16s_C1(const Ipp16s* pSrc, Ipp16s* pDst);
IppStatus ippiDCT8x8Inv_4x4_16s_C1(const Ipp16s* pSrc, Ipp16s* pDst);
```

```
IppStatus ippiDCT8x8Inv_2x2_16s_C1I(Ipp16s* pSrcDst);
IppStatus ippiDCT8x8Inv_4x4_16s_C1I(Ipp16s* pSrcDst);
```

### Parameters

<i>pSrc</i>	Pointer to the source image.
<i>pDst</i>	Pointer to the destination buffer.
<i>pSrcDst</i>	Pointer to the source and destination buffer for in-place operations.

### Description

The functions `ippiDCT8x8Inv_2x2` and `ippiDCT8x8Inv_4x4` are declared in the `ippi.h` file. These functions compute the inverse discrete cosine transform of non-zero elements in the top left quadrant of size 2x2 or 4x4 in the 2D buffer of 8x8 size. No prerequisites are needed to use this transform function.



**CAUTION.** Source data for 16s flavors must be the result of the forward discrete cosine transform of data from the range [-256, 255], they cannot be arbitrary data from the range [-32768, 32767].

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointers is <code>NULL</code> .

## DCT8x8To2x2Inv, DCT8x8To4x4Inv

*Perform an inverse DCT on a 2D buffer of 8x8 size with further downsampling to 2x2 or 4x4 size.*

### Syntax

```
IppStatus ippiDCT8x8To2x2Inv_16s_C1(const Ipp16s* pSrc, Ipp16s* pDst);
IppStatus ippiDCT8x8To4x4Inv_16s_C1(const Ipp16s* pSrc, Ipp16s* pDst);
IppStatus ippiDCT8x8To2x2Inv_16s_C1I(Ipp16s* pSrcDst);
IppStatus ippiDCT8x8To4x4Inv_16s_C1I(Ipp16s* pSrcDst);
```

### Parameters

<i>pSrc</i>	Pointer to the source image.
<i>pDst</i>	Pointer to the destination buffer.
<i>pSrcDst</i>	Pointer to the source and destination buffer for in-place operations.

## Description

The functions `ippiDCT8x8To2x2Inv` and `ippiDCT8x8To4x4Inv` are declared in the `ippi.h` file. These functions compute the inverse discrete cosine transform of the 2D buffer *pSrc* of 8x8 size. Then the functions perform downsampling of the result by averaging to the destination buffer *pDst* of size 2x2 or 4x4.

In-place flavors of the functions perform operations on the source and destination buffer *pSrcDst*.



---

**CAUTION.** Source data for 16s flavors must be the result of the forward discrete cosine transform of data from the range [-256, 255], they cannot be arbitrary data from the range [-32768, 32767].

---

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointers is <code>NULL</code> .

# Image Statistics Functions

This chapter describes the Intel® IPP image processing functions that can be used to compute the following statistical parameters of an image:

- sum, integrals, mean and standard deviation of pixel values
- intensity histogram of pixel values
- minimum and maximum pixel values
- spatial and central moments of order 0 to 3
- the infinity,  $L1$ , and  $L2$  norms of the image pixel values and of the differences between pixel values of two images
- relative error values for the infinity,  $L1$ , and  $L2$  norms of differences between pixel values of two images
- universal image quality index
- proximity measures of an image and a template (another image).

The table below lists the image statistics functions:

## Image Statistics Functions

Function Base Name	Operation
<code>Sum</code>	Computes the sum of image pixel values
<code>Integral</code>	Transforms an image to the integral representation
<code>SqrIntegral</code>	Transforms an image to integral and integral of pixel squares representations
<code>TiltedIntegral</code>	Transforms an image to the tilted integral representation
<code>TiltedSqrIntegral</code>	Transforms an image to tilted integral and tilted integral of pixel squares representations
<code>Mean</code>	Computes the mean of image pixel values
<code>MeanStdDev</code>	Computes the mean and standard deviation of image pixel values
<code>RectStdDev</code>	Computes the standard deviation of the integral images.
<code>TiltedRectStdDev</code>	Computes the standard deviation of the tilted integral images.
<code>HistogramRange</code>	Computes the intensity histogram of an image
<code>HistogramEven</code>	Computes the intensity histogram of an image with equal bins
<code>CountInRange</code>	Counts the number of pixels within the given intensity range
<b>Minimum and Maximum Operations</b>	
<code>Min</code>	Computes the minimum of image pixel values
<code>MinIndx</code>	Computes the minimum of image pixel values, and retrieves the coordinates of pixels with minimal intensity values
<code>Max</code>	Computes the maximum of image pixel values
<code>MaxIndx</code>	Computes the maximum of image pixel values, and retrieves the coordinates of pixels with maximal intensity values
<code>MinMax</code>	Computes the minimum and maximum of image pixel values

Function Base Name	Operation
<a href="#">MinMaxIndx</a>	Calculates minimum and maximum pixel values and their indexes
<a href="#">MaxEvery</a>	Computes maximum value for each pair of pixels of two images
<a href="#">MinEvery</a>	Computes minimum value for each pair of pixels of two images
<a href="#">FindPeaks3x3GetBufferSize</a>	Computes the size of the working buffer for the peak search
<a href="#">FindPeaks3x3</a>	Finds coordinates of peaks (maximums or minimums) with absolute value exceeding threshold value
<b>Moments</b>	
<a href="#">MomentInitAlloc</a>	Allocates memory and initializes the moment context structure
<a href="#">MomentFree</a>	Deallocates the previously initialized structure that stores image moments
<a href="#">MomentGetStateSize</a>	Computes the size of the external buffer for the moment context structure
<a href="#">MomentInit</a>	Initializes the moment context structure
<b>Moments</b>	Computes all image moments of order 0 to 3 and Hu moment invariants
<a href="#">GetSpatialMoment</a>	Retrieves image spatial moment computed by <code>ippiMoments</code>
<a href="#">GetCentralMoment</a>	Retrieves image central moment computed by <code>ippiMoments</code>
<a href="#">GetNormalizedSpatialMoment</a>	Retrieves the normalized value of the image spatial moment computed by <code>ippiMoments</code>
<a href="#">GetNormalizedCentralMoment</a>	Retrieves the normalized value of the image central moment computed by <code>ippiMoments</code>
<a href="#">GetHuMoments</a>	Retrieves image Hu moment invariants computed by <code>ippiMoments</code>
<b>Norms</b>	
<a href="#">Norm_Inf</a>	Computes the infinity norm of image pixel values
<a href="#">Norm_L1</a>	Computes the L1-norm of image pixel values
<a href="#">Norm_L2</a>	Computes the L2-norm of image pixel values
<a href="#">NormDiff_Inf</a>	Computes the infinity norm of differences between pixel values of two images
<a href="#">NormDiff_L1</a>	Computes the L1-norm of differences between pixel values of two images
<a href="#">NormDiff_L2</a>	Computes the L2-norm of differences between pixel values of two images
<a href="#">NormRel_Inf</a>	Computes the relative error for the infinity norm of differences between pixel values of two images
<a href="#">NormRel_L1</a>	Computes the relative error for the L1-norm of differences between pixel values of two images.
<a href="#">NormRel_L2</a>	Computes the relative error for the L2-norm of differences between pixel values of two images.
<b>Image Quality Index</b>	
<a href="#">QualityIndex</a>	Computes the universal image quality index
<b>Proximity Measures</b>	
<a href="#">SqrDistanceFull_Norm</a>	Computes normalized full Euclidean distance between an image and a template.
<a href="#">SqrDistanceSame_Norm</a>	Computes normalized Euclidean distance between an image and a template.
<a href="#">SqrDistanceValid_Norm</a>	Computes normalized valid Euclidean distance between an image and a template.
<a href="#">CrossCorrFull_Norm</a>	Computes normalized full cross-correlation between an image and a template.
<a href="#">CrossCorrSame_Norm</a>	Computes normalized cross-correlation between an image and a template.
<a href="#">CrossCorrValid_Norm</a>	Computes normalized valid cross-correlation between an image and a template.
<a href="#">CrossCorrValid</a>	Computes valid cross-correlation between an image and a template.

Function Base Name	Operation
<a href="#">CrossCorrFull_NormLevel</a>	Computes normalized full correlation coefficients between an image and a template.
<a href="#">CrossCorrSame_NormLevel</a>	Computes normalized correlation coefficients between an image and a template.
<a href="#">CrossCorrValid_NormLevel</a>	Computes normalized valid correlation coefficients between an image and a template.

## Sum

*Computes the sum of image pixel values.*

### Syntax

#### Case 1: Operation on one-channel integer data

```
IppStatus ippisum_<mod>(const Ipp<datatype>* pSrc, int srcStep, IppiSize roiSize, Ipp64f* pSum);
```

Supported values for mod:

8u\_C1R      16u\_C1R      16s\_C1R

#### Case 2: Operation on one-channel floating-point data

```
IppStatus ippisum_32f_C1R(const Ipp32f* pSrc, int srcStep, IppiSize roiSize, Ipp64f* pSum, IppHintAlgorithm hint);
```

#### Case 3: Operation on multi-channel integer data

```
IppStatus ippisum_<mod>(const Ipp<datatype>* pSrc, int srcStep, IppiSize roiSize, Ipp64f sum[3]);
```

Supported values for mod:

8u\_C3R      16u\_C3R      16s\_C3R  
8u\_AC4R      16u\_AC4R      16s\_AC4R

```
IppStatus ippisum_<mod>(const Ipp<datatype>* pSrc, int srcStep, IppiSize roiSize, Ipp64f sum[4]);
```

Supported values for mod:

8u\_C4R      16u\_C4R      16s\_C4R

#### Case 4: Operation on multi-channel floating-point data

```
IppStatus ippisum_<mod>(const Ipp32f* pSrc, int srcStep, IppiSize roiSize, Ipp64f sum[3], IppHintAlgorithm hint);
```

Supported values for mod:

32f\_C3R

32f\_AC4R

```
IppStatus ippiSum_32f_C4R(const Ipp32f* pSrc, int srcStep, IppiSize roiSize, Ipp64f sum[4],
IppHintAlgorithm hint);
```

## Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>roiSize</i>	Size of the source ROI in pixels.
<i>pSum</i>	Pointer to the computed sum of pixel values.
<i>sum</i>	Array containing computed sums of channel values of pixels in the source buffer.
<i>hint</i>	Option to select the algorithmic implementation of the function.

## Description

The function `ippiSum` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)). This function computes the sum of pixel values *pSum* for the source image *pSrc* using algorithm indicated by the *hint* argument (see [Table "Hint Arguments for Image Moment Functions"](#)). In case of a multi-channel image, the sum is computed over each channel and stored in the array *sum*.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if the <i>pSrc</i> or <i>pSum</i> pointer is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.



## Example

The code example below demonstrates the use of `ippiSum` function:

```
IppStatus sum( void ) {
    Ipp64f sum;
    Ipp8u x[5*4];
    IppiSize roi = {5,4};
    ippiSet_8u_C1R( 1, x, 5, roi );
    return ippiSum_8u_C1R( x, 5, roi,
    &sum);
}
```

## Integral

*Transforms an image to the integral representation.*

### Syntax

```
IppStatus ippiIntegral_8u32s_C1R(const Ipp8u* pSrc, int srcStep, Ipp32s* pDst, int dstStep,
IppiSize roiSize, Ipp32s val);

IppStatus ippiIntegral_8u32f_C1R(const Ipp8u* pSrc, int srcStep, Ipp32f* pDst, int dstStep,
IppiSize roiSize, Ipp32f val);
```

### Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the ROI in the destination integral image.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>roiSize</i>	Size of source and destination image ROI in pixels.
<i>val</i>	The value to add to <i>pDst</i> image pixels

### Description

The function `ippiIntegral` is declared in the `ippcv.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)). This function transforms a source image *pSrc* to the integral image *pDst*. Pixel values of the destination image *pDst* are computed using pixel values of the source image *pSrc* and the specified value *val* in accordance with the following formula:

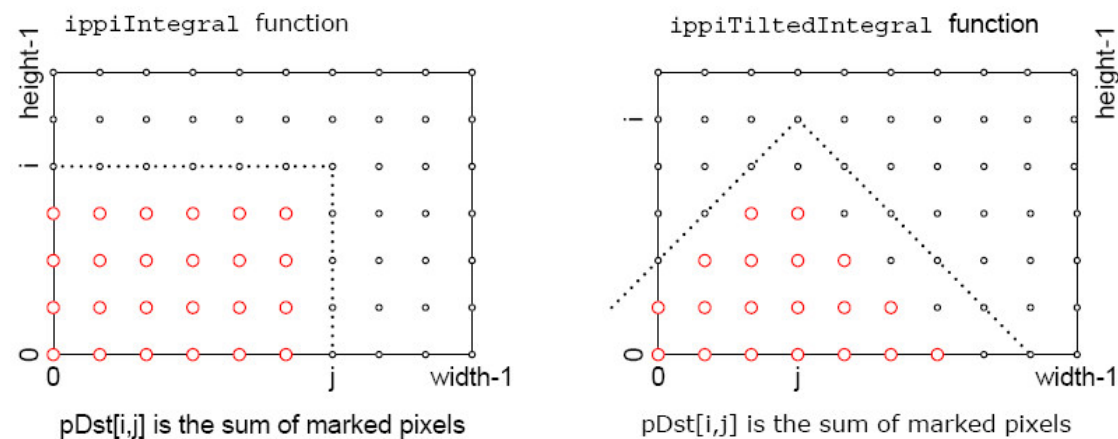
$$pDst[i, j] = val + \sum_{k < i} \sum_{l < j} pSrc[k, l]$$

where  $i, j$  are coordinates of the destination image pixels (see Figure 11-1 ) varying in the range  $i = 1, \dots, roiSize.height$ ,  $j = 0, \dots, roiSize.width$ . Pixel values of zero row and column of  $pDst$  ( $i=0$ ) is set to  $val$ . The size of the destination images is  $(roiSize.width + 1) \times (roiSize.height + 1)$ .

Figure “Operation of the Integral and TiltedIntegral functions” shows what pixels (red circles) of the source image are used in computation new pixel values in the  $i, j$  coordinates.

For large images the result of summation can exceed the upper bound of the output data type. Table “Maximum Image Size for Integral Functions” lists the maximum image size for different function flavors and values.

## Operation of the Integral and TiltedIntegral functions



## Maximum Image Size for Integral Functions

Function Flavor	Value $val$	Maximum Image Size
ippiIntegral_8u32s_C1R	0	$(2^{31}-1)/255$
	$-2^{31}$	$2^{32}/255$
ippiIntegral_8u32f_C1R	0	$2^{24}/255$
	$-2^{24}$	$(2^{25}+1)/255$

## Return Values

ippStsNoEr	Indicates no error.
ippStsNullPtrErr	Indicates an error if $pSrc$ or $pDst$ is NULL.
ippStsSizeErr	Indicates an error condition if $roiSize$ has a field with zero or negative value.
ippStsStepErr	Indicates an error condition if $srcStep$ is less than $roiSize.width * \langle pixelSize \rangle$ , or $dstStep$ is less than $(roiSize.width+1) * \langle pixelSize \rangle$ .
ippStsNotEvenStepErr	Indicates an error condition if one $dstStep$ is not divisible by 4.

## Example

The code example below demonstrates how to use the function `ippiIntegral`.

```
void func_inegral()
{
    Ipp8u pSrc[5*4];

    Ipp32s pDst[6*5];
    IppiSize ROI = {5,4};
    ippiSet_8u_C1R(1,pSrc,5,ROI);
    Ipp32s val = 1;
    ippiIntegral_8u32s_C1R(pSrc, 5, pDst, 6*4, ROI, val);
}
```

result:

pSrc -> 1 1 1 1 1	pDst -> 1 1 1 1 1 1
1 1 1 1 1	1 2 3 4 5 6
1 1 1 1 1	1 3 5 7 9 11
1 1 1 1 1	1 4 7 10 13 16
	1 5 9 13 17 21

## SqrIntegral

*Transforms an image to integral and integral of pixel squares representations.*

### Syntax

```
IppStatus ippiSqrIntegral_8u32s_C1R(const Ipp8u* pSrc, int srcStep, Ipp32s* pDst, int
dstStep, Ipp32s* pSqr, int sqrStep, IppiSize roiSize, Ipp32s val, Ipp32s valSqr);

IppStatus ippiSqrIntegral_8u32s64f_C1R(const Ipp8u* pSrc, int srcStep, Ipp32s* pDst, int
dstStep, Ipp64f* pSqr, int sqrStep, IppiSize roiSize, Ipp32s val, Ipp64f valSqr);

IppStatus ippiSqrIntegral_8u32f64f_C1R(const Ipp8u* pSrc, int srcStep, Ipp32f* pDst, int
dstStep, Ipp64f* pSqr, int sqrStep, IppiSize roiSize, Ipp32f val, Ipp64f valSqr);
```

### Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the ROI in the destination integral image.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.

<i>pSqr</i>	Pointer to the ROI of the destination integral image of pixel squares.
<i>sqrStep</i>	Distance in bytes between starts of consecutive lines in the destination integral image of pixel squares.
<i>roiSize</i>	Size of source image ROI in pixels.
<i>val</i>	The value to add to <i>pDst</i> image pixels.
<i>valSqr</i>	The value to add to <i>pSqr</i> image pixels

## Description

The function `ippiSqrIntegral` is declared in the `ippcv.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)). This function builds two destination images: integral image *pDst* and integral image of pixel squares *pSqr*. Pixel values of *pDst* are computed using pixel values of the source image *pSrc* and the specified value *val* in accordance with the following formula:

$$pDst[i, j] = val + \sum_{k < i} \sum_{l < j} pSrc[k, l]$$

Pixel values of *pSqr* are computed using pixel values of the source image *pSrc* and the specified value *valSqr* in accordance with the following formula:

$$pSqr[i, j] = valSqr + \sum_{k < i} \sum_{l < j} pSrc[k, l]^2$$

where *i, j* are coordinates of the destination image pixels (see [Figure “Operation of the Integral and TiltedIntegral functions”](#)) varying in the range *i* = 1, ..., *roiSize.height*, *j* = 0, ..., *roiSize.width*. Pixel values of zero row and column are set to *val* for *pDst*, and to *valSqr* for *pSqr*. The size of both destination images is (*roiSize.width* + 1) × (*roiSize.height* + 1).

Figure “Operation of the Integral and TiltedIntegral functions” shows what pixels (red circles) of the source image are used in computation new pixel values in the *i, j* coordinates.

## Return Values

<code>ippStsNoEr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error if one of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> is less than <i>roiSize.width</i> * <i>&lt;pixelSize&gt;</i> , or <i>dstStep</i> or <i>sqrStep</i> is less than ( <i>roiSize.width</i> +1) * <i>&lt;pixelSize&gt;</i> .
<code>ippStsNotEvenStepErr</code>	Indicates an error condition if <i>dstStep</i> is not divisible by 4, or <i>sqrStep</i> is not divisible by 8.

## TiltedIntegral

*Transforms an image to the tilted integral representation.*

### Syntax

```
IppStatus ippiTiltedIntegral_8u32s_C1R(const Ipp8u* pSrc, int srcStep, Ipp32s* pDst, int dstStep, IppiSize roiSize, Ipp32s val);
```

```
IppStatus ippiTiltedIntegral_8u32f_C1R(const Ipp8u* pSrc, int srcStep, Ipp32f* pDst, int dstStep, IppiSize roiSize, Ipp32f val);
```

### Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the ROI in the destination integral image.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>roiSize</i>	Size of source image ROI in pixels.
<i>val</i>	The value to add to pDst image pixels

### Description

The function `ippiTiltedIntegral` is declared in the `ippcv.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)). This function transforms a source image *pSrc* to the tilted integral image *pDst*. Pixel values of the destination image *pDst* are computed using pixel values of the source image *pSrc* and the specified value *val* in accordance with the following formula:

$$pDst[i, j] = val + \sum_{\substack{k+1 \leq i+j-2 \\ k-1 \leq i-j+1}} pSrc[k, l]$$

where *i, j* are coordinates of the destination image pixels (see [Figure "Operation of the Integral and TiltedIntegral functions"](#)) varying in the range  $i = 2, \dots, roiSize.height + 1$ ,  $j = 0, \dots, roiSize.width + 1$ . Pixel values of rows 0 and 1 of the destination image *pDst* ( $i=0$ ) is set to *val*.

The size of the destination images is  $(roiSize.width + 2) \times (roiSize.height + 2)$ .

[Figure "Operation of the Integral and TiltedIntegral functions"](#) shows what pixels (red circles) of the source image are used in computation new pixel values in the *i, j* coordinates.

### Return Values

<code>ippStsNoEr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error if <i>pSrc</i> or <i>pDst</i> is NULL.

<code>ippStsSizeErr</code>	Indicates an error condition if <code>roiSize</code> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <code>srcStep</code> is less than <code>roiSize.width * &lt;pixelSize&gt;</code> , or <code>dstStep</code> is less than <code>(roiSize.width+2) * &lt;pixelSize&gt;</code> .
<code>ippStsNotEvenStepErr</code>	Indicates an error condition if one <code>dstStep</code> is not divisible by 4.

## TiltedSqrIntegral

*Transforms an image to tilted integral and tilted integral of pixel squares representations.*

---

### Syntax

```

IppStatus ippiTiltedSqrIntegral_8u32s_C1R(const Ipp8u* pSrc, int srcStep, Ipp32s* pDst,
int dstStep, Ipp32s* pSqr, int sqrStep, IppiSize roiSize, Ipp32s val, Ipp32s valSqr);

IppStatus ippiTiltedSqrIntegral_8u32s64f_C1R(const Ipp8u* pSrc, int srcStep, Ipp32s* pDst,
int dstStep, Ipp64f* pSqr, int sqrStep, IppiSize roiSize, Ipp32s val, Ipp64f valSqr);

IppStatus ippiTiltedSqrIntegral_8u32f64f_C1R(const Ipp8u* pSrc, int srcStep, Ipp32f* pDst,
int dstStep, Ipp64f* pSqr, int sqrStep, IppiSize roiSize, Ipp32f val, Ipp64f valSqr);

```

### Parameters

<code>pSrc</code>	Pointer to the source image ROI.
<code>srcStep</code>	Distance in bytes between starts of consecutive lines in the source image.
<code>pDst</code>	Pointer to the ROI in the destination integral image.
<code>dstStep</code>	Distance in bytes between starts of consecutive lines in the destination image.
<code>pSqr</code>	Pointer to the ROI of the destination integral image of pixel squares.
<code>sqrStep</code>	Distance in bytes between starts of consecutive lines in the destination integral image of pixel squares.
<code>roiSize</code>	Size of source image ROI in pixels.
<code>val</code>	The value to add to <code>pDst</code> image pixels.
<code>valSqr</code>	The value to add to <code>pSqr</code> image pixels

### Description

The function `ippiTiltedSqrIntegral` is declared in the `ippcv.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)). This function builds two destination image: tilted integral image `pDst` and tilted integral image of pixel squares `pSqr`.

Pixel values of `pDst` are computed using pixel values of the source image `pSrc` and the specified value `val` in accordance with the following formula:

$$pDst[i, j] = val + \sum_{\substack{k+1 \leq i+j-2 \\ k-1 \leq i-j+1}} pSrc[k, l]$$

Pixel values of *pSqr* are computed using pixel values of the source image *pSrc* and the specified value *valSqr* in accordance with the following formula:

$$pSqr[i, j] = valSqr + \sum_{\substack{k+1 \leq i+j-2 \\ k-1 \leq i-j+1}} pSrc[k, l]^2$$

where *i, j* are coordinates of the destination image pixels (see [Figure "Operation of the Integral and TiltedIntegral functions"](#)) varying in the range *i* = 2 ..., *roiSize.height*, *j* = 0 ..., *roiSize.width*. Pixel values of zero and first rows (*i*=0,1) are set to *val* for *pDst*, and to *valSqr* for *pSqr*. The size of both destination images is (*roiSize.width* + 2) × (*roiSize.height* + 2).

[Figure "Operation of the Integral and TiltedIntegral functions"](#) shows what pixels (red circles) of the source image are used in computation new pixel values in the *i, j* coordinates.

## Return Values

<code>ippStsNoEr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error if one of the specified pointers is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> is less than <i>roiSize.width</i> * <i>&lt;pixelSize&gt;</i> , or <i>dstStep</i> or <i>sqrStep</i> is less than ( <i>roiSize.width</i> +2) * <i>&lt;pixelSize&gt;</i> .
<code>ippStsNotEvenStepErr</code>	Indicates an error condition if <i>dstStep</i> is not divisible by 4, or <i>sqrStep</i> is not divisible by 8.

## Mean

*Computes the mean of image pixel values.*

## Syntax

### Case 1: Operation on one-channel integer data

```
IppStatus ippMean_<mod>(const Ipp<datatype>* pSrc, int srcStep, IppiSize roiSize, Ipp64f* pMean);
```

Supported values for *mod*:

8u\_C1R      16u\_C1R      16s\_C1R

**Case 2: Operation on one-channel floating-point data**

```
ippStatus ippiMean_32f_C1R(const Ipp32f* pSrc, int srcStep, IppiSize roiSize, Ipp64f*
pMean, IppHintAlgorithm hint);
```

**Case 3: Masked operation on one-channel data**

```
IppStatus ippiMean_<mod>(const Ipp<datatype>* pSrc, int srcStep, const Ipp8u* pMask, int
maskStep, IppiSize roiSize, Ipp64f* pMean);
```

Supported values for mod:

8u_C1MR	8s_C1MR	16u_C1MR	32f_C1MR
---------	---------	----------	----------

**Case 4: Operation on multi-channel integer data**

```
IppStatus ippiMean_<mod>(const Ipp<datatype>* pSrc, int srcStep, IppiSize roiSize, Ipp64f
mean[3]);
```

Supported values for mod:

8u_C3R	16u_C3R	16s_C3R
8u_AC4R	16u_AC4R	16s_AC4R

```
IppStatus ippiMean_<mod>(const Ipp<datatype>* pSrc, int srcStep, IppiSize roiSize, Ipp64f
mean[4]);
```

Supported values for mod:

8u_C4R	16u_C4R	16s_C4R
--------	---------	---------

**Case 5: Operation on multi-channel floating-point data**

```
IppStatus ippiMean_<mod>(const Ipp32f* pSrc, int srcStep, IppiSize roiSize, Ipp64f mean[3],
IppHintAlgorithm hint);
```

Supported values for mod:

32f_C3R
32f_AC4R

```
IppStatus ippiMean_32f_C4R(const Ipp32f* pSrc, int srcStep, IppiSize roiSize, Ipp64f
mean[4], IppHintAlgorithm hint);
```

**Case 6: Masked operation on multi-channel data**

```
IppStatus ippiMean_<mod>(const Ipp<datatype>* pSrc, int srcStep, const Ipp8u* pMask, int
maskStep, IppiSize roiSize, int coi, Ipp64f* pMean);
```

Supported values for mod:

8u_C3CMR	8s_C3CMR	16u_C3CMR	32f_C3CMR
----------	----------	-----------	-----------



## Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pMask</i>	Pointer to the mask image.
<i>maskStep</i>	Distance in bytes between starts of consecutive lines in the mask image.
<i>roiSize</i>	Size of the source ROI in pixels.
<i>coi</i>	Channel of interest (for color images only); can be 1, 2, or 3.
<i>pMean</i>	Pointer to the computed mean of pixel values.
<i>mean</i>	Array containing computed mean values for each channel of a multi-channel image.
<i>hint</i>	Option to select the algorithmic implementation of the function.

## Description

The flavors of the function `ippiMean` that perform masked operations are declared in the `ippcv.h` file. All other function flavors are declared in the `ippi.h` file. This function operates with ROI (see [Regions of Interest in Intel IPP](#)). It computes the mean (average) of pixel values *pMean* for the source image *pSrc*. Computation algorithm is specified by the *hint* argument (see [Table "Hint Arguments for Image Moment Functions"](#)). For non-masked operations on a multi-channel image (Case 4, 5), the mean is computed over each channel and stored in the array *mean*. In the mask multi-channel mode (Case 6), the mean is computed for a single channel of interest specified by *coi*.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error when any of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition in mask mode, if <i>srcStep</i> or <i>maskStep</i> is less than <i>roiSize.width</i> * <code>&lt;pixelSize&gt;</code> .
<code>ippStsNotEvenStepErr</code>	Indicates an error condition in mask mode if steps for floating-point images cannot be divided by 4.
<code>ippStsCOIErr</code>	Indicates an error when <i>coi</i> is not 1, 2, or 3.

## Example

The code example below shows how to use the `ippiMean` function.

```

IppStatus mean( void ) {
    Ipp64f mean;
    Ipp8u x[5*4];
    IppiSize roi = {5,4};
    ippiSet_8u_C1R( 3, x, 5, roi );
    return ippiMean_8u_C1R( x, 5,
        roi, &mean );
}

```

## Mean\_StdDev

*Computes the mean and standard deviation of image pixel values.*

---

### Syntax

#### Case 1: Operation on one-channel data

```

IppStatus ippiMean_StdDev_<mod>(const Ipp<datatype>* pSrc, int srcStep, IppiSize roiSize,
    Ipp64f* pMean, Ipp64f* pStdDev);

```

Supported values for `mod`:

8u_C1R	8s_C1R	16u_C1R	32f_C1R
--------	--------	---------	---------

#### Case 2: Masked operation on one-channel data

```

IppStatus ippiMean_StdDev_<mod>(const Ipp<datatype>* pSrc, int srcStep, const Ipp8u* pMask,
    int maskStep, IppiSize roiSize, Ipp64f* pMean, Ipp64f* pStdDev);

```

Supported values for `mod`:

8u_C1MR	8s_C1MR	16u_C1MR	32f_C1MR
---------	---------	----------	----------

#### Case 3: Operation on multi-channel data

```

IppStatus ippiMean_StdDev_<mod>(const Ipp<datatype>* pSrc, int srcStep, IppiSize roiSize,
    int coi, Ipp64f* pMean, Ipp64f* pStdDev);

```

Supported values for `mod`:

8u_C3CR	8s_C3CR	16u_C3CR	32f_C3CR
---------	---------	----------	----------

**Case 4: Masked operation on multi-channel data**

```
IppStatus ippiMean_StdDev_<mod>(const Ipp<datatype>* pSrc, int srcStep, const Ipp8u* pMask,
int maskStep, IppiSize roiSize, int coi, Ipp64f* pMean, Ipp64f* pStdDev);
```

Supported values for mod:

8u\_C3CMR      8s\_C3CMR      16u\_C3CMR      32f\_C3CMR

**Parameters**

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pMask</i>	Pointer to the mask image.
<i>maskStep</i>	Distance in bytes between starts of consecutive lines in the mask image.
<i>roiSize</i>	Size of the source ROI in pixels
<i>coi</i>	Channel of interest (for color images only); can be 1, 2, or 3.
<i>pMean</i>	Pointer to the computed mean of pixel values.
<i>pStdDev</i>	Pointer to the computed standard deviation of pixel values in the image.

**Description**

The function `ippiMean_StdDev` is declared in the `ippcv.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)). This function computes the mean and standard deviation of pixel values in the ROI of the source image *pSrc*. In the mask mode, the computation is done over pixels selected by nonzero mask values. In the multi-channel mode, the mean is computed for a single channel of interest specified by *coi*. If any of the parameters *pMean* or *pStdDev* is not required, the zero pointer is to be passed to the corresponding parameter.

**Return Values**

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error when <i>pSrc</i> or <i>pMask</i> pointer is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> or <i>maskStep</i> is less than <i>roiSize.width</i> * <pixelSize>.
<code>ippStsNotEvenStepErr</code>	Indicates an error condition if steps for floating-point images cannot be divided by 4.
<code>ippStsCOIErr</code>	Indicates an error when <i>coi</i> is not 1, 2, or 3.

## RectStdDev

Computes the standard deviation of the integral images.

### Syntax

```
IppStatus ippiRectStdDev_32f_C1R(const Ipp32f* pSrc, int srcStep, const Ipp64f* pSqr, int
sqrStep, Ipp32f* pDst, int dstStep, IppiSize roiSize, IppiRect rect);
```

```
IppStatus ippiRectStdDev_32s_C1RSfs(const Ipp32s* pSrc, int srcStep, const Ipp32s* pSqr,
int sqrStep, Ipp32s* pDst, int dstStep, IppiSize roiSize, IppiRect rect, int scaleFactor);
```

```
IppStatus ippiRectStdDev_32s32f_C1R(const Ipp32s* pSrc, int srcStep, const Ipp64f* pSqr,
int sqrStep, Ipp32f* pDst, int dstStep, IppiSize roiSize, IppiRect rect);
```

### Parameters

<i>pSrc</i>	Pointer to the ROI in the source integral image.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source integral image.
<i>pSqr</i>	Pointer to the ROI in the source integral image of pixel squares.
<i>sqrStep</i>	Distance in bytes between starts of consecutive lines in the source integral image of pixel squares.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>roiSize</i>	Size of destination image ROI in pixels.
<i>rect</i>	Rectangular window.
<i>scaleFactor</i>	Scale factor (see <a href="#">Integer Result Scaling</a> ).

### Description

The function `ippiRectStdDev` is declared in the `ippcv.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)). This function computes the standard deviation for each pixel in the rectangular window *rect* using the integral image *pSrc* and integral image of pixel squares *pSqr*. The computations are performed in accordance with the following formulas:

$$pDst[i, j] = \sqrt{\max\left(0, \frac{sumSqr \cdot numPix - sum^2}{numPix^2}\right)}$$

where *i, j* are coordinates of the destination image pixels varying in the range *i* = 0 ..., *roiSize.height* - 1, *j* = 0 ..., *roiSize.width* - 1;

```
sum = pSrc[ i + rect.y + rect.height, j + rect.x + rect.width] - pSrc[ i + rect.y, j + rect.x +
rect.width] - pSrc[ i + rect.y + rect.height, j + rect.x] + pSrc[ i + rect.y, j + rect.x];
```

```
sumSqr = pSqr[ i + rect.y + rect.height, j + rect.x + rect.width] - pSqr[i + rect.y, j + rect.x +
rect.width] - pSqr[i + rect.y + rect.height, j + rect.x] + pSqr[i + rect.y, j + rect.x];
```

```
numPix = rect.height * rect.width.
```

The minimum size of each source images *pSrc* and *pSqr* should be  $(roiSize.width + rect.x + rect.width) \times (roiSize.height + rect.y + rect.height)$ .

The source images *pSrc* and *pSqr* can be obtained by using the functions [ippiIntegral](#) or [ippiSqrIntegral](#).

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error if one of the specified pointers is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>rect.width</i> or <i>rect.height</i> is less than or equal to zero, or if <i>rect.x</i> or <i>rect.y</i> is less than zero.
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> or <i>sqrStep</i> is less than $(roiSize.width + rect.x + rect.width + 1) \times \langle pixelSize \rangle$ , or <i>dstStep</i> is less than $roiSize.width \times \langle pixelSize \rangle$ .
<code>ippStsNotEvenStepErr</code>	Indicates an error condition if <i>sqrStep</i> is not divisible by 8, or one of <i>pSrc</i> and <i>dstStep</i> is not divisible by 4.

## TiltedRectStdDev

*Computes the standard deviation of the tilted integral images.*

### Syntax

```
IppStatus ippiTiltedRectStdDev_32f_C1R(const Ipp32f* pSrc, int srcStep, const Ipp64f*
pSqr, int sqrStep, Ipp32f* pDst, int dstStep, IppiSize roiSize, IppiRect rect);
```

```
IppStatus ippiTiltedRectStdDev_32s_C1RSfs(const Ipp32s* pSrc, int srcStep, const Ipp32s*
pSqr, int sqrStep, Ipp32s* pDst, int dstStep, IppiSize roiSize, IppiRect rect, int
scaleFactor);
```

```
IppStatus ippiTiltedRectStdDev_32s32f_C1R(const Ipp32s* pSrc, int srcStep, const Ipp64f*
pSqr, int sqrStep, Ipp32f* pDst, int dstStep, IppiSize roiSize, IppiRect rect);
```

### Parameters

<i>pSrc</i>	Pointer to the ROI in the source integral image.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source integral image.
<i>pSqr</i>	Pointer to the ROI in the source integral image of pixel squares.
<i>sqrStep</i>	Distance in bytes between starts of consecutive lines in the source integral image of pixel squares.
<i>pDst</i>	Pointer to the destination image ROI.

<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>roiSize</i>	Size of destination image ROI in pixels.
<i>rect</i>	Rectangular window.
<i>scaleFactor</i>	Scale factor (see <a href="#">Integer Result Scaling</a> ).

## Description

The function `ippiTiltedRectStdDev` is declared in the `ippcv.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)). This function computes the standard deviation for each pixel in the rectangular window *rect* using the tilted integral image *pSrc* and tilted integral image of pixel squares *pSqr*. The computations are performed in accordance with the following formulas:

$$pDst[i, j] = \sqrt{\max\left(0, \frac{sumSqr \cdot numPix - sum^2}{numPix^2}\right)}$$

where *i, j* are coordinates of the destination image pixels varying in the range *i* = 0 ..., *roiSize.height* - 1, *j* = 0 ..., *roiSize.width* - 1;

```
sum = pSrc[ i + rect.x - rect.y + rect.height + rect.width, j + rect.x + rect.y - rect.height +
rect.width] - pSrc[ i + rect.x - rect.y + rect.width, j + rect.x + rect.y + rect.width] - pSrc[ i +
rect.x - rect.y + rect.height, j + rect.x - rect.y - rect.height] + pSrc[ i + rect.x - rect.y, j +
rect.x + rect.y];
```

```
sumSqr = pSqr[ i + rect.x - rect.y + rect.height + rect.width, j + rect.x + rect.y - rect.height
+ rect.width] - pSqr[ i + rect.x - rect.y + rect.width, j + rect.x + rect.y + rect.width] - pSqr[ i
+ rect.x - rect.y + rect.height, j + rect.x - rect.y - rect.height] + pSqr[ i + rect.x - rect.y, j
+ rect.x + rect.y];
```

```
numPix = 2 * rect.height * rect.width.
```

The minimum size of each source images *pSrc* and *pSqr* should be  $(roiSize.width + rect.height + rect.width - 2) \times (roiSize.height + rect.x + rect.y + rect.height + rect.width - 2)$ .

The source images *pSrc* and *pSqr* can be obtained by using the functions [ippiTiltedIntegral](#) or [ippiTiltedSqrIntegral](#).

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error if one of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>rect.width</i> or <i>rect.height</i> is less than or equal to zero, or if <i>rect.x</i> or <i>rect.y</i> is less than zero.
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> or <i>sqrStep</i> is less than $(roiSize.width + rect.x + rect.width + 1) * <pixelSize>$ , or <i>dstStep</i> is less than $roiSize.width * <pixelSize>$ .

`ippStsNotEvenStepErr`

Indicates an error condition if *srcStep* is not divisible by 8, or one of *pSrc* and *dstStep* is not divisible by 4.

## HistogramRange

*Computes the intensity histogram of an image.*

---

### Syntax

#### Case 1: Operation on one-channel integer data

```
IppStatus ippHistogramRange_<mod>(const Ipp<datatype>* pSrc, int srcStep, IppiSize roiSize,
Ipp32s* pHist, const Ipp32s* pLevels, int nLevels);
```

Supported values for *mod*:

8u\_C1R          16u\_C1R          16s\_C1R

#### Case 2: Operation on multi-channel integer data

```
IppStatus ippHistogramRange_<mod>(const Ipp<datatype>* pSrc, int srcStep, IppiSize roiSize,
Ipp32s* pHist[3], const Ipp32s* pLevels[3], int nLevels[3]);
```

Supported values for *mod* :

8u\_C3R          16u\_C3R          16s\_C3R  
8u\_AC4R          16u\_AC4R          16s\_AC4R

```
IppStatus ippHistogramRange_<mod>(const Ipp<datatype>* pSrc, int srcStep, IppiSize roiSize,
Ipp32s* pHist[4], const Ipp32s* pLevels[4], int nLevels[4]);
```

Supported values for *mod* :

8u\_C4R          16u\_C4R          16s\_C4R

#### Case 3: Operation on one-channel floating-point data

```
IppStatus ippHistogramRange_32f_C1R(const Ipp32f* pSrc, int srcStep, IppiSize roiSize,
Ipp32s* pHist, const Ipp32f* pLevels, int nLevels);
```

#### Case 4: Operation on multi-channel floating-point data

```
IppStatus ippHistogramRange_<mod>(const Ipp32f* pSrc, int srcStep, IppiSize roiSize,
Ipp32s* pHist[3], const Ipp32f* pLevels[3], int nLevels[3]);
```

Supported values for *mod*:

32f\_C3R  
32f\_AC4R

```
IppStatus ippiHistogramRange_32f_C4R(const Ipp32f* pSrc, int srcStep, IppiSize roiSize,
Ipp32s* pHist[4], const Ipp32f* pLevels[4], int nLevels[4]);
```

## Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>roiSize</i>	Size of the source ROI in pixels.
<i>pHist</i>	Pointer to the computed histogram. In case of multi-channel data, <i>pHist</i> is an array of pointers to the histogram for each channel.
<i>pLevels</i>	Pointer to the array of level values. In case of multi-channel data, <i>pLevels</i> is an array of pointers to the level values array for each channel.
<i>nLevels</i>	Number of levels, separate for each channel.

## Description

The function `ippiHistogramRange` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)). This function computes the intensity histogram of an image in the ranges specified by the array (or arrays, in case of multi-channel data) *pLevels*. The histograms computed separately for each channel are stored in the arrays *pHist*. Before calling this function, the array *pLevels* should be initialized to determine bounds of the bins for histogram computing. Length of the arrays *pLevels* and *pHist* is defined by the *nLevels* parameter. Since *nLevels* is the number of levels, the number of histogram bins is *nLevels* - 1. Similarly, the number of values in the array *pHist* is also *nLevels* - 1. The meaning of *pHist* and *pLevels* values can be illustrated by the following example: *pHist* [k] is the number of source image pixels *pSrc*(*x*,*y*) that satisfy the condition

$$pLevels[k] \leq pSrc(x,y) < pLevels[k+1].$$

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error if one of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.
<code>ippStsMemAllocErr</code>	Indicates an error if there is not enough memory for the inner histogram.
<code>ippStsHistoNofLevelsErr</code>	Indicates an error if <i>nLevels</i> is less than 2.



## Example

The code example below shows how to use the `ippiHistogramRange` function:

```
// Compute histogram of an image for 4 bins in the range [28, 127]
{
    Ipp8u img[WIDTH*HEIGHT];
    IppiSize imgSize = {WIDTH, HEIGHT};
    const Ipp32s
    levels[5] = {28, 50, 70, 90, 128};
    Ipp32s histo[5];
    ippiHistogramRange_8u_C1R(img, WIDTH, imgSize, levels, histo, 5);
    // after executing the function the array histo

    // will contain a histogram in specified range.
    // Actually, four result values will be stored

    // compute cumulative histogram
    for (i = 1; i < 4; i++)
        histo[i] += histo[i-1];
}
```

## HistogramEven

*Computes the intensity histogram of an image using equal bins.*

---

### Syntax

#### Case 1: Operation on one-channel integer data

```
IppStatus ippiHistogramEven_<mod>(const Ipp<datatype>* pSrc, int srcStep, IppiSize roiSize,
Ipp32s* pHist, Ipp32s* pLevels, int nLevels, Ipp32s lowerLevel, Ipp32s upperLevel);
```

**HistogramRange**Supported values for mod:

8u\_C1R            16u\_C1R            16s\_C1R

#### Case 2: Operation on multi-channel integer data

```
IppStatus ippiHistogramEven_<mod>(const Ipp<datatype>* pSrc, int srcStep, IppiSize roiSize,
Ipp32s* pHist[3], Ipp32s* pLevels[3], int nLevels[3], Ipp32s lowerLevel[3], Ipp32s
upperLevel[3]);
```

Supported values for mod:

8u\_C3R            16u\_C3R            16s\_C3R

8u\_AC4R                  16u\_AC4R                  16s\_AC4R

```
IppStatus ippiHistogramEven_<mod>(const Ipp<datatype>* pSrc, int srcStep, IppiSize roiSize,
Ipp32s* pHist[4], Ipp32s* pLevels[4], int nLevels[4], Ipp32s lowerLevel[4], Ipp32s
upperLevel[4]);
```

Supported values for mod:

8u\_C4R                  16u\_C4R                  16s\_C4R

## Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>roiSize</i>	Size of the source ROI in pixels
<i>pHist</i>	Pointer to the computed histogram. In case of multi-channel data, <i>pHist</i> is an array of pointers to the histogram for each channel.
<i>pLevels</i>	Pointer to the array of level values. In case of multi-channel data, <i>pLevels</i> is an array of pointers to the level values array for each channel.
<i>nLevels</i>	Number of levels, separate for each channel.
<i>lowerLevel</i>	Lower level boundary, separate for each channel.
<i>upperLevel</i>	Upper level boundary, separate for each channel.

## Description

The function `ippiHistogramEven` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)). This function computes the intensity histogram of an image in the ranges specified by the values *lowerLevel* (inclusive), *upperLevel* (exclusive), and *nLevels*. The function operates on the assumption that all histogram bins have the same width and equal boundary values of the bins (levels). The function stores computed histograms in the array *pHist* separately for each channel. The calculated levels are stored in the array *pLevels*.

Length of the arrays *pLevels* and *pHist* is defined by the *nLevels* parameter. Since *nLevels* is the number of levels, the number of histogram bins is *nLevels* - 1. Similarly, the number of values in the array *pHist* is also *nLevels* - 1. The meaning of *pHist* and *pLevels* values can be illustrated by the following example: *pHist* [k] is a number of source image pixels *pSrc*(*x*,*y*) that satisfy the condition

$$pLevels[k] \leq pSrc(x,y) < pLevels[k+1].$$

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error if one of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with a zero or negative value.

---

<code>ippStsMemAllocErr</code>	Indicates an error if there is not enough memory for the inner histogram.
<code>ippStsHistoNofLevelsErr</code>	Indicates an error if <i>nLevels</i> is less than 2.

### Example

The code example below shows how to use the `ippiHistogramEven` function.

```
// Compute histogram of an image for 4 bins in the range [28, 127];
// compute level values for the bins.
{
    Ipp8u img[WIDTH*HEIGHT];
    IppiSize imgSize = {WIDTH, HEIGHT};
    Ipp32s levels[5], histo[5];
    ippiHistogramEven_8u_C1R(img, WIDTH, imgSize, levels,
                             histo, 5, 28, 128);

    // When the function completes operation the array histo will
    // contain a histogram in specified range, and the array

    // levels will contain the level values {28, 53, 78, 103, 128}
}
```

## CountInRange

*Computes the number of pixels within the given intensity range.*

---

### Syntax

#### Case 1: Operation on one-channel data

```
IppStatus ippiCountInRange_<mod>(const Ipp<datatype>* pSrc, int srcStep, IppiSize roiSize,
int* counts, Ipp<datatype> lowerBound, Ipp<datatype> upperBound);
```

Supported values for mod:

`8u_C1R`      `32f_C1R`

#### Case 2: Operation on multi-channel data

```
IppStatus ippiCountInRange_<mod>(const Ipp<datatype>* pSrc, int srcStep, IppiSize roiSize,
int counts[3], Ipp<datatype> lowerBound[3], Ipp<datatype> upperBound[3]);
```

Supported values for mod:

`8u_C3R`      `32f_C3R`  
`8u_AC4R`      `32f_AC4R`

## Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>roiSize</i>	Size of the source ROI in pixels.
<i>counts</i>	The computed number of pixels within the given intensity range. An array of 3 values in case of multi-channel data.
<i>lowerBound</i>	Lower limit of the intensity range.
<i>upperBound</i>	Upper limit of the intensity range.

## Description

The function `ippiCountInRange` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)). This function computes the number of pixels in the image which have intensity values in the range between *lowerBound* and *upperBound* (inclusive).

In case of a multi-channel image, pixels are counted within intensity range for each color channel separately, and the array *counts* of three resulting values is returned. The alpha channel values, if present, are not processed.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pSrc</i> pointer is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> has a zero or negative value.
<code>ippStsRangeErr</code>	Indicates an error condition if <i>lowerBound</i> exceeds <i>upperBound</i> .

## Min

*Computes the minimum of image pixel values.*

---

## Syntax

### Case 1: Operation on one-channel data

```
IppStatus ippiMin_<mod>(const Ipp<datatype>* pSrc, int srcStep, IppiSize roiSize,
Ipp<datatype>* pMin);
```

Supported values for *mod*:

8u_C1R	16u_C1R	16s_C1R	32f_C1R
--------	---------	---------	---------

**Case 2: Operation on multi-channel data**

```
IppStatus ippiMin_<mod>(const Ipp<datatype>* pSrc, int srcStep, IppiSize roiSize,
Ipp<datatype> min[3]);
```

Supported values for mod:

8u_C3R	16u_C3R	16s_C3R	32f_C3R
8u_AC4R	16u_AC4R	16s_AC4R	32f_AC4R

```
IppStatus ippiMin_<mod>(const Ipp<datatype>* pSrc, int srcStep, IppiSize roiSize,
Ipp<datatype> min[4]);
```

Supported values for mod:

8u_C4R	16u_C4R	16s_C4R	32f_C4R
--------	---------	---------	---------

**Parameters**

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>roiSize</i>	Size of the source ROI in pixels.
<i>pMin</i>	Pointer to the minimum pixel value (for one-channel data).
<i>min</i>	Array containing minimum channel values of pixels in the source buffer (for multi-channel data).

**Description**

The function `ippiMin` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)). This function computes the minimum pixel value *pMin* for the source image *pSrc*. In case of a multi-channel image, the minimum is computed over each channel and stored in the array *min*.

**Return Values**

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pSrc</i> or <i>pMin</i> pointer is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.

## Example

The code example below demonstrates how to use the function `ippiMin`.

```

Ipp8u src[4*1] = { 40, 20, 60, 80 };
IppiSize roiSize = { 4, 1 };
Ipp8u min;
ippiMin_8u_C1R ( src, 4, roiSize, &min );
result: min = 20

```

## MinIndx

*Computes the minimum of image pixel values and retrieves the x and y coordinates of pixels with minimal intensity values.*

---

### Syntax

#### Case 1: Operation on one-channel data

```

IppStatus ippiMinIndx_<mod>(const Ipp<datatype>* pSrc, int srcStep, IppiSize roiSize,
Ipp<datatype>* pMin, int* pIndexX, int* pIndexY);

```

Supported values for `mod`:

8u_C1R	16u_C1R	16s_C1R	32f_C1R
--------	---------	---------	---------

#### Case 2: Operation on multi-channel data

```

IppStatus ippiMinIndx_<mod>(const Ipp<datatype>* pSrc, int srcStep, IppiSize roiSize,
Ipp<datatype> min[3], int indexX[3], int indexY[3]);

```

Supported values for `mod`:

8u_C3R	16u_C3R	16s_C3R	32f_C3R
8u_AC4R	16u_AC4R	16s_AC4R	32f_AC4R

```

IppStatus ippiMinIndx_<mod>(const Ipp<datatype>* pSrc, int srcStep, IppiSize roiSize,
Ipp<datatype> min[4], int indexX[4], int indexY[4]);

```

Supported values for `mod`:

8u_C4R	16u_C4R	16s_C4R	32f_C4R
--------	---------	---------	---------

### Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.

<i>roiSize</i>	Size of the source ROI in pixels.
<i>pMin</i>	Pointer to the minimum pixel value (for one-channel data).
<i>min</i>	Array containing minimum color channel values of pixels in the source buffer (for multi-channel data).
<i>pIndexX</i> , <i>pIndexY</i>	Pointers to the x and y coordinates of the pixel with minimum value.
<i>indexX</i> , <i>indexY</i>	Arrays containing the x and y coordinates of pixels with minimum channel values.

## Description

The function `ippiMinIndx` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function computes the minimum pixel value *pMin* for the source image *pSrc*. In case of a multi-channel image, the minimum is computed over each channel and stored in the array *min*. The function also retrieves the *x* and *y* coordinates of pixels on which the minimum is reached. If several pixels have equal minimum values, the coordinates of the first pixel from the start of the source buffer is returned. For multi-channel data, *indexX[k]* and *indexY[k]* are the *x* and *y* coordinates of the pixel that has the minimal intensity value of the *k*-th channel, *k* = 1,2,3,4.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of <i>pSrc</i> , <i>pMin</i> , <i>pIndexX</i> , or <i>pIndexY</i> pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.

## Max

*Computes the maximum of image pixel values.*

## Syntax

### Case 1: Operation on one-channel data

```
IppStatus ippiMax_<mod>(const Ipp<datatype>* pSrc, int srcStep, IppiSize roiSize,
Ipp<datatype>* pMax);
```

Supported values for *mod*:

8u_C1R	16u_C1R	16s_C1R	32f_C1R
--------	---------	---------	---------

**Case 2: Operation on multi-channel data**

```
IppStatus ippiMax_<mod>(const Ipp<datatype>* pSrc, int srcStep, IppiSize roiSize,
Ipp<datatype> max[3]);
```

Supported values for mod:

8u_C3R	16u_C3R	16s_C3R	32f_C3R
8u_AC4R	16u_AC4R	16s_AC4R	32f_AC4R

```
IppStatus ippiMax_<mod>(const Ipp<datatype>* pSrc, int srcStep, IppiSize roiSize,
Ipp<datatype> max[4]);
```

Supported values for mod:

8u_C4R	16u_C4R	16s_C4R	32f_C4
--------	---------	---------	--------

**Parameters**

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>roiSize</i>	Size of the source ROI in pixels
<i>pMax</i>	Pointer to the maximum pixel value (for one-channel data).
<i>max</i>	Array containing maximum channel values of pixels in the source buffer (for multi-channel data).

**Description**

The function `ippiMax` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)). This function computes the maximum pixel value *pMax* for the source image *pSrc*. In case of a multi-channel image, the maximum is computed over each channel and stored in the array *max*.

**Return Values**

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pSrc</i> or <i>pMax</i> pointer is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.



## MaxIndx

*Computes the maximum of image pixel values and retrieves the x and y coordinates of pixels with maximal intensity values.*

---

### Syntax

#### Case 1: Operation on one-channel data

```
IppStatus ippiMaxIndx_<mod>(const Ipp<datatype>* pSrc, int srcStep, IppiSize roiSize,
Ipp<datatype>* pMax, int* pIndexX, int* pIndexY);
```

Supported values for mod:

8u_C1R	16u_C1R	16s_C1R	32f_C1R
--------	---------	---------	---------

#### Case 2: Operation on multi-channel data

```
IppStatus ippiMaxIndx_<mod>(const Ipp<datatype>* pSrc, int srcStep, IppiSize roiSize,
Ipp<datatype> max[3], int indexX[3], int indexY[3]);
```

Supported values for mod:

8u_C3R	16u_C3R	16s_C3R	32f_C3R
8u_AC4R	16u_AC4R	16s_AC4R	32f_AC4R

```
IppStatus ippiMaxIndx_<mod>(const Ipp<datatype>* pSrc, int srcStep, IppiSize roiSize,
Ipp<datatype> max[4], int indexX[4], int indexY[4]);
```

Supported values for mod:

8u_C4R	16u_C3R	16s_C3R	32f_C4R
--------	---------	---------	---------

### Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>roiSize</i>	Size of the source ROI in pixels.
<i>pMax</i>	Pointer to the maximum pixel value (for one-channel data).
<i>max</i>	Array containing maximum channel values of pixels in the source buffer (for multi-channel data).
<i>pIndexX</i> , <i>pIndexY</i>	Pointers to the x and y coordinates of the pixel with maximum value.
<i>indexX</i> , <i>indexY</i>	Arrays containing the x and y coordinates of pixels with maximum channel values.

## Description

The function `ippiMaxIndx` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)). This function computes the maximum pixel value `pMax` for the source image `pSrc`. In case of a multi-channel image, the maximum is computed over each channel and stored in the array `max`. The function also retrieves the `x` and `y` coordinates of pixels on which the maximum is reached. If several pixels have equal maximum values, the coordinates of the first pixel from the start of the source buffer is returned. For multi-channel data, `indexX[k]` and `indexY[k]` are the `x` and `y` coordinates of the pixel that has the maximal intensity value of the `k`-th channel, `k = 1,2,3,4`.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if any of <code>pSrc</code> , <code>pMax</code> , <code>pIndexX</code> , or <code>pIndexY</code> pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <code>roiSize</code> has a field with zero or negative value.

## Example

The code example below demonstrates how to use the function `ippiMaxIndx`.

```

Ipp8u src[4*1] = { 40, 20, 60, 80 };
IppiSize roiSize = { 4, 1 };
Ipp8u max;
int IndexX;
int IndexY;

ippiMaxIndx_8u_C1R ( src, 4, roiSize, &max, &IndexX, &IndexY );
result: max = 80  IndexX = 3  IndexY = 0

```

## MinMax

*Computes the minimum and maximum of image pixel values.*

---

## Syntax

### Case 1: Operation on one-channel data

```

IppStatus ippiMinMax_<mod>(const Ipp<datatype>* pSrc, int srcStep, IppiSize roiSize,
Ipp<datatype>* pMin, Ipp<datatype>* pMax);

```

Supported values for `mod`:

<code>8u_C1R</code>	<code>16u_C1R</code>	<code>16s_C1R</code>	<code>32f_C1R</code>
---------------------	----------------------	----------------------	----------------------

**Case 2: Operation on multi-channel data**

```
IppStatus ippiMinMax_<mod>(const Ipp<datatype>* pSrc, int srcStep, IppiSize roiSize,
Ipp<datatype> min[3], Ipp<datatype> max[3]);
```

Supported values for mod:

8u_C3R	16u_C3R	16s_C3R	32f_C3R
8u_AC4R	16u_AC4R	16s_AC4R	32f_AC4R

```
IppStatus ippiMinMax_<mod>(const Ipp<datatype>* pSrc, int srcStep, IppiSize roiSize,
Ipp<datatype> min[4], Ipp<datatype> max[4]);
```

Supported values for mod:

8u_C4R	16u_C4R	16s_C4R	32f_C4R
--------	---------	---------	---------

**Parameters**

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>roiSize</i>	Size of the source ROI in pixels.
<i>pMin</i> , <i>pMax</i>	Pointers to the minimum and maximum pixel values (for one-channel data).
<i>min</i> , <i>max</i>	Arrays containing minimum and maximum channel values of pixels in the source buffer (for multi-channel data).

**Description**

The function `ippiMinMax` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)). This function computes the minimum and maximum pixel values *pMin* and *pMax* for the source image *pSrc*. In case of a multi-channel image, the minimum and maximum is computed over each channel and stored in the arrays *min* and *max*.

**Return Values**

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pSrc</i> , <i>pMin</i> , or <i>pMax</i> pointer is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.

## MinMaxIndx

*Calculates minimum and maximum pixel values and their indexes in selected image rectangle.*

---

### Syntax

#### Case 1: Operation on one-channel data

```
IppStatus ippiMinMaxIndx_<mod>(const Ipp<datatype>* pSrc, int srcStep, IppiSize roiSize,
Ipp32f* pMinVal, Ipp32f* pMaxVal, IppiPoint* pMinIndex, IppiPoint* pMaxIndex);
```

Supported values for mod:

8u\_C1R            8s\_C1R            16u\_C1R            32f\_C1R

#### Case 2: Masked operation on one-channel data

```
IppStatus ippiMinMaxIndx_<mod>(const Ipp<datatype>* pSrc, int srcStep, const Ipp8u* pMask,
int maskStep, IppiSize roiSize, Ipp32f* pMinVal, Ipp32f* pMaxVal, IppiPoint* pMinIndex,
IppiPoint* pMaxIndex);
```

Supported values for mod:

8u\_C1MR            8s\_C1MR            16u\_C1MR            32f\_C1MR

#### Case 3: Operation on multi-channel data

```
IppStatus ippiMinMaxIndx_<mod>(const Ipp<datatype>* pSrc, int srcStep, IppiSize roiSize,
int coi, Ipp32f* pMinVal, Ipp32f* pMaxVal, IppiPoint* pMinIndex, IppiPoint* pMaxIndex);
```

Supported values for mod:

8u\_C3CR            8s\_C3CR            16u\_C3CR            32f\_C3CR

#### Case 4: Masked operation on multi-channel data

```
IppStatus ippiMinMaxIndx_<mod>(const Ipp<datatype>* pSrc, int srcStep, const Ipp8u* pMask,
int maskStep, IppiSize roiSize, int coi, Ipp32f* pMinVal, Ipp32f* pMaxVal, IppiPoint*
pMinIndex, IppiPoint* pMaxIndex);
```

Supported values for mod:

8u\_C3CMR            8s\_C3CMR            16u\_C3CMR            32f\_C3CMR

### Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pMask</i>	Pointer to the mask image.

<i>maskStep</i>	Distance in bytes between starts of consecutive lines in the mask image.
<i>roiSize</i>	Size of the image ROI in pixels.
<i>coi</i>	Channel of interest (for color images only); can be 1, 2, or 3.
<i>pMinVal</i>	Pointer to the variable that returns the value of the minimum pixel.
<i>pMaxVal</i>	Pointer to the variable that returns the value of the maximum pixel.
<i>pMinIndex</i>	Pointer to the variable that returns the index of the minimum value found.
<i>pMaxIndex</i>	Pointer to the variable that returns the index of the maximum value found.

## Description

The function `ippiMinMaxIndx` is declared in the `ippcv.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)). This function finds minimum and maximum pixel values and their indexes in an image ROI or in an arbitrary image region defined by nonzero mask values. If there are several minima and maxima in the selected area, the function returns the top leftmost positions. If the specified region in the mask mode is empty, that is, the mask image is filled with zeros, then the function returns  $\{minIndex, maxIndex\} = \{0, 0\}$ ,  $minVal = maxVal = 0$ . If any of the parameters *pMinVal*, *pMaxVal*, *pMinIndex*, or *pMaxIndex* is not required, the zero pointer is to be passed to the corresponding parameter.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pSrc</i> or <i>pMask</i> pointer is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error for masked operations when <i>srcStep</i> or <i>maskStep</i> is less than <i>roiSize.width</i> * <i>&lt;pixelSize&gt;</i> .
<code>ippStsNotEvenStepErr</code>	Indicates an error when steps for floating-point images cannot be divided by 4.
<code>ippStsCOIErr</code>	Indicates an error when <i>coi</i> is not 1, 2, or 3.

## MaxEvery

*Computes maximum value for each pair of pixels of two images.*

---

### Syntax

```
IppStatus ippiMaxEvery_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pSrcDst,
int srcDstStep, IppiSize roiSize);
```

Supported values for `mod`:

8u_C1IR	16u_C1IR	16s_C1IR	32f_C1IR
8u_C3IR	16u_C3IR	16s_C3IR	32f_C3IR
8u_C4IR	16u_C4IR	16s_C4IR	32f_C4IR
8u_AC4IR	16u_AC4IR	16s_AC4IR	32f_AC4IR

### Parameters

<i>pSrc</i>	Pointer to the first source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the first source image.
<i>pSrcDst</i>	Pointer to the second source and destination image ROI.
<i>srcDstStep</i>	Distance in bytes between starts of consecutive lines in the second source and destination image.
<i>roiSize</i>	Size of the image ROI in pixels.

### Description

The function `ippiMaxEvery` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function computes the maximum value for each pair of the corresponding pixels of two source images *pSrc* and *pSrcDst*, and stores the result in the *pSrcDst*:

$pSrcDst(i, j) = \max(pSrc(i, j), pSrcDst(i, j)).$

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> or <i>srcDstStep</i> is less than <i>roiSize.width*&lt;pixelSize&gt;</i> .
<code>ippStsNotEvenStepErr</code>	Indicates an error condition if one of step values for floating-point images are not divisible by 4.

## MinEvery

Computes minimum value for each pair of pixels of two images.

### Syntax

```
IppStatus ippiMinEvery_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pSrcDst,
int srcDstStep, IppiSize roiSize);
```

Supported values for mod:

8u_C1IR	16u_C1IR	16s_C1IR	32f_C1IR
8u_C3IR	16u_C3IR	16s_C3IR	32f_C3IR
8u_C4IR	16u_C4IR	16s_C4IR	32f_C4IR
8u_AC4IR	16u_AC4IR	16s_AC4IR	32f_AC4IR

### Parameters

<i>pSrc</i>	Pointer to the first source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the first source image.
<i>pSrcDst</i>	Pointer to the second source and destination image ROI.
<i>srcDstStep</i>	Distance in bytes between starts of consecutive lines in the second source and destination image.
<i>roiSize</i>	Size of the image ROI in pixels.

### Description

The function `ippiMinEvery` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function computes the minimum value for each pair of the corresponding pixels of two source images *pSrc* and *pSrcDst*, and stores the result in the *pSrcDst*:

$$pSrcDst(i, j) = \min(pSrc(i, j), pSrcDst(i, j)).$$

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> or <i>srcDstStep</i> is less than <i>roiSize.width*&lt;pixelSize&gt;</i> .
<code>ippStsNotEvenStepErr</code>	Indicates an error condition if one of step values for floating-point images are not divisible by 4.

## FindPeaks3x3GetBufferSize

*Computes the size of the working buffer for the peak search.*

---

### Syntax

```
IppStatus ippiFindPeaks3x3GetBufferSize_32s_C1R(int roiWidth, int* pBufferSize);
IppStatus ippiFindPeaks3x3GetBufferSize_32f_C1R(int roiWidth, int* pBufferSize);
```

### Parameters

<i>roiWidth</i>	Maximum width of the image, in pixels.
<i>pBufferSize</i>	Pointer to the size of the working buffer.

### Description

The function `ippiFindPeaks3x3GetBufferSize` is declared in the `ippcv.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function computes the size of the working buffer required for the function [ippiFindPeaks3x3](#). The buffer with the length `pBufferSize[0]` can be used to filter images with width that is less than or equal to *roiWidth*.

[Example](#) shows how to use the function `ippiFindPeaks3x3GetBufferSize_32f_C1R`.

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if the pointer <i>pBufferSize</i> is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiWidth</i> is less than 1.

## FindPeaks3x3

*Finds coordinates of peaks (maximums or minimums) with absolute value exceeding threshold value.*

---

### Syntax

```
IppStatus ippiFindPeaks3x3_32s_C1R(const Ipp32s* pSrc, int srcStep, IppiSize roiSize,
Ipp32s threshold, IppiPoint* pPeak, int maxPeakCount, int* pPeakCount, IppiNorm norm, int
border, Ipp8u* pBuffer);
IppStatus ippiFindPeaks3x3_32f_C1R(const Ipp32f* pSrc, int srcStep, IppiSize roiSize,
Ipp32f threshold, IppiPoint* pPeak, int maxPeakCount, int* pPeakCount, IppiNorm norm, int
border, Ipp8u* pBuffer);
```

### Parameters

<i>pSrc</i>	Pointer to the first source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the first source image.



<i>roiSize</i>	Size of the image ROI in pixels.
<i>threshold</i>	Threshold value.
<i>pPeak</i>	Pointer to the coordinates peaks [ <i>maxPeakCount</i> ].
<i>maxPeakCount</i>	Maximum number of peaks.
<i>pPeakCount</i>	Pointer to the number of the detected peaks.
<i>border</i>	Border value, only pixel with distance from the edge of the image greater than <i>border</i> are processed.
<i>norm</i>	Specifies type of the norm to form the mask for extremum search:
<i>ippiNormInf</i>	Infinity norm (8-connectivity, 3x3 rectangular mask);
<i>ippiNormL1</i>	L1 norm (4-connectivity, 3x3 cross mask).
<i>pBuffer</i>	Pointer to the working buffer.

## Description

The function *ippiFindPeaks3x3* is declared in the *ippcv.h* file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function detects local maximum and minimum pixels in the source image:

$$pSrc(i_m, j_m) = \max_{(k, l) \in O(i_m, j_m)} pSrc(k, l), \quad pSrc(i_m, j_m) \geq threshold$$

$$pSrc(i_m, j_m) = \min_{(k, l) \in O(i_m, j_m)} pSrc(k, l), \quad |pSrc(i_m, j_m)| \geq threshold$$

and stores their coordinates in the *pPeak* array *pPeak*[*m*].*x* = *jm*, *pPeak*[*m*].*y* = *im*, *m* = 0, ...  
*pPeakCount* [0], *pPeakCount*[0] ≤ *maxPeakCount*

The neighborhood *O*(*i*, *j*) for the extremum search is defined by the parameter *norm*. The number of detected extremums is returned in *pPeakCount*[0]. The operation is stopped when the *maxPeakCount* extremums are found.

The function requires the working buffer *pBuffer* whose size should be computed by the function [ippiFindPeaks3x3GetBufferSize](#) beforehand.

## Return Values

<i>ippStsNoErr</i>	Indicates no error. Any other value indicates an error or a warning.
<i>ippStsNullPtrErr</i>	Indicates an error condition if one of the specified pointers is NULL.

<code>ippStsSizeErr</code>	Indicates an error condition if <code>roiSize</code> has a field with zero or negative value; or if <code>maxPeakCount</code> is less than or equal to 0; or if <code>border</code> is less than 1 or greater than one of $0.5 * \text{roiSize.width}$ or of $0.5 * \text{roiSize.height}$ .
<code>ippStsStepErr</code>	Indicates an error condition if <code>srcStep</code> is less than $\text{roiSize.width} * \text{pixelSize}$ .
<code>ippStsNotEvenStepErr</code>	Indicates an error condition if <code>srcStep</code> is not divisible by 4.

## Example

The code example below shows how to use the function `ippiFindPeaks3x3_32f_C1R` for calculations of the SIFT (Scale Invariant Feature Transform) features [Lowe04].

```

Ipp32s sigma[NIMAGE];      // sigmas of Gauss blur (increasing
Ipp32s kerSize[NIMAGE];    // kernel size of Gauss blur (increasing)
Ipp32f *pDst[NIMAGE];      // blurred images
Ipp32f *pSrc;              // initial image
Ipp8u *pBuffer;            // working buffer
IppiPoint *pPeak;          // array of peaks
IppiSize roi;              // image size
int step;                  // row step in bytes
int bufSize;               // working buffer size
int maxPeaks=500;          // max peaks number for image
int peakCount;             // peaks number for image
Ipp32f threshold=0.02f;    // peak threshold
ippiFilterGaussGetBufferSize_32f_C1R(roi,kerSize[NIMAGE-1],&bufSize);
pBuffer = ippsMalloc_8u(bufSize);
for (i=0; i<NIMAGE;i++) { //blur initial image with different Gauss kernels
    ippiFilterGaussBorder_32f_C1R(pSrc,step,pDst[i],step,roi,kerSize[i],
        sigma[i],ippBorderRepl,0,pBuffer);
}
ippsFree(pBuffer);
for (i=0; i<NIMAGE-1;i++) { // calculate difference of Gaussian (DOG) images
    ippiSub_32f_C1R(pDst[i+1],step,pDst[i],step,roi);
}
ippiFindPeaks3x3GetBufferSize_32f_C1R(roi.width,&bufSize);
pBuffer = ippsMalloc_8u(bufSize);
pPeak = (IppiPoint*)ippsMalloc_8u(maxPeaks*sizeof(IppiPoint));
for (i=1; i<NIMAGE-2;i++) { // find and process peaks in DOG images
    ippiFindPeaks3x3_32f_C1R(pDst[i],step,roi,threshold,pPeak,maxPeaks,
        &peakCount,ippiNormL1,kerSize[0]/2,pBuffer);
    for (j=0;j<peakCount;j++) {
        Ipp32fpix = pDst[i][pPeak[j].y*step/sizeof(Ipp32f)+pPeak[j].x];
        Ipp32fpup = pDst[i+1][pPeak[j].y*step/sizeof(Ipp32f)+pPeak[j].x];
        Ipp32fpdn = pDst[i-1][pPeak[j].y*step/sizeof(Ipp32f)+pPeak[j].x];
    }
}

```

```

        if (pix>0)if ((pix<pup)|| (pix<pdn)) continue;
        if (pix<0)if ((pix>pup)|| (pix>pdn)) continue;
//      calculateSIFT feature description for a peak pPeak[j] in DOG image pDst[i]
    }
}
ippsFree(pBuffer);
ippsFree(pPeak);

```

## Image Moments

Spatial and central moments are important statistical properties of an image. The spatial moment  $M_U(m, n)$  of order  $(m, n)$  is defined as follows:

$$M_U(m, n) = \sum_{j, k} x_k^m y_j^n P_{j, k}$$

where the summation is performed for all rows and columns in the image;  $P_{j, k}$  are pixel values;  $x_k$  and  $y_j$  are pixel coordinates;  $m$  and  $n$  are integer power exponents that define the moment order.

The central moment  $U_U(m, n)$  is the spatial moment computed relative to the “center of gravity”  $(x_0, y_0)$ :

$$U_U(m, n) = \sum_{j, k} (x_k - x_0)^m (y_j - y_0)^n P_{j, k}$$

where  $x_0 = M_U(1, 0) / M_U(0, 0)$  and  $y_0 = M_U(0, 1) / M_U(0, 0)$ .

The normalized spatial moment  $M(m, n)$  and central moment  $U(m, n)$  are defined as follows:

$$M(m, n) = \frac{M_U(m, n)}{M_U(0, 0)^{\frac{m+n+2}{2}}}$$

$$U(m, n) = \frac{U_U(m, n)}{U_U(0, 0)^{\frac{m+n+2}{2}}}$$

The Intel IPP functions support moments of order  $(m, n)$  with  $0 \leq m + n \leq 3$ . The computation of seven invariant Hu moments derived from the second and third order moments is also supported. All computed moments are stored in context structures of type `IppiMomentState_64s` (for integer versions) or `IppiMomentState_64f` (for floating point versions).

Most Intel IPP functions for computing image moments have code branches that implement different algorithms to compute the results. You can choose the desired code variety to be used by the given function by setting the *hint* argument to one of the following values that are listed in [Table “Hint Arguments for Image Moment Functions”](#):

#### Hint Arguments for Image Moment Functions

Value	Description
<code>ippAlgHintNone</code>	The computation algorithm will be chosen by the internal function logic.
<code>ippAlgHintFast</code>	Fast algorithm must be used. The output results will be less accurate.
<code>ippAlgHintAccurate</code>	High accuracy algorithm must be used. The function will need more time to execute.

## MomentInitAlloc

*Allocates memory and initializes the moment context structure.*

### Syntax

```
ippStatus ippMomentInitAlloc_64f(IppiMomentState_64f** pState, IppHintAlgorithm hint);
ippStatus ippMomentInitAlloc_64s(IppiMomentState_64s** pState, IppHintAlgorithm hint);
```

### Parameters

*pState*                                      Pointer to the structure for storing moment values.  
*hint*                                         Option to select the algorithmic implementation of the function.

### Description

The function `ippMomentInitAlloc` is declared in the `ippi.h` file. This function initializes the structure that is needed for the function `ippMoments` to store the computed image moments. Different functions are used to allocate structures for integer and floating-point data. Computation algorithm is specified by *hint* argument (see [Table “Hint Arguments for Image Moment Functions”](#)).

If the initialization is successful, the `ippMomentInitAlloc` function sets the *pState* pointer to the allocated structure of `IppiMomentState` type.

If memory allocation failed, the function returns `ippStsMemAllocErr` error status code.

### Return Values

`ippStsNoErr`                                      Indicates no error. Any other value indicates an error or a warning.  
`ippStsMemAllocErr`                               Indicates an error condition in case of memory allocation failure.

## MomentFree

*Frees memory allocated by the function  
ippiMomentInitAlloc.*

---

### Syntax

```

IppStatus ippiMomentFree_64f(IppiMomentState_64f* pState);
IppStatus ippiMomentFree_64s(IppiMomentState_64s* pState);

```

### Parameters

*pState*                                      Pointer to the structure for image moments storage.

### Description

The function `ippiMomentFree` is declared in the `ippi.h` file. Use this function to deallocate the previously initialized *pState* structure that stored the image moments data. Different functions are used to deallocate structures for integer and floating-point data.

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pState</i> pointer is <code>NULL</code> .
<code>ippStsContextMatchErr</code>	Indicates an error condition if a pointer to an invalid structure is passed.

## MomentGetStateSize

*Computes the size of the external buffer for the moment  
context structure.*

---

### Syntax

```

IppStatus ippiMomentGetStateSize_64s(IppHintAlgorithm hint, int* pBufferSize);

```

### Parameters

*pBufferSize*                              Pointer to the computed value of the buffer size.  
*hint*                                        Option to select the algorithmic implementation of the function.

### Description

The function `ippiMomentGetStateSize` is declared in the `ippi.h` file. Use this function to determine the size of the external work buffer for the moment context structure to be initialized by the function `ippiMomentInit`. Computation algorithm is specified by *hint* argument (see [Table "Hint Arguments for Image Moment Functions"](#)).

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pBufferSize</i> pointer is <code>NULL</code> .

## MomentInit

*Initializes the moment context structure.*

---

### Syntax

```
IppStatus ippMomentInit_64s(IppiMomentState_64s* pState, IppHintAlgorithm hint);
```

### Parameters

<i>pState</i>	Pointer to the structure for storing moment values.
<i>hint</i>	Option to select the algorithmic implementation of the function.

### Description

The function `ippMomentInit` is declared in the `ippi.h` file. This function initializes the structure that is needed for the function `ippiMoments` to store the computed image moments. Computation algorithm is specified by *hint* argument (see [Table "Hint Arguments for Image Moment Functions"](#)).

The structure is allocated in the external buffer. The size of this buffer can be computed by the function `ippMomentGetStateSize`.

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pState</i> pointer is <code>NULL</code> .

## Moments

*Computes all image moments of order 0 to 3 and Hu moment invariants.*

---

### Syntax

#### Case 1: Computation of floating-point results

```
IppStatus ippMoments64f_<mod>(const Ipp<datatype>* pSrc, int srcStep, IppiSize roiSize, IppiMomentState_64f* pState);
```

Supported values for `mod`:

8u_C1R	16u_C1R	32f_C1R
8u_C3R	16u_C3R	32f_C3R
8u_AC4R	16u_AC4R	32f_AC4R

## Case 2: Computation of integer results

```
IppStatus ippIMoments64s_<mod>(const Ipp8u* pSrc, int srcStep, IppiSize roiSize,
IppiMomentState_64s* pState);
```

Supported values for `mod`:

8u_C1R	16u_C1R
8u_C3R	16u_C3R
8u_AC4R	16u_AC4R

## Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>roiSize</i>	Size of the source ROI in pixels.
<i>pState</i>	Pointer to the structure that stores image moments.

## Description

The function `ippIMoments` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)). This function computes all spatial and central moments of order 0 to 3 for the source image *pSrc*. The seven Hu moment invariants are also computed. Different functions, `ippIMoments64s` and `ippIMoments64f`, are used to compute image moments in integer and floating-point formats, respectively.

The `ippIMoments` function computes spatial moment values relative to the image point referred to by *pSrc*. Note that this point is the ROI origin and may not coincide with the entire image origin. If you need to obtain spatial moment values relative to the actual image origin, use [ippiGetSpatialMoment](#) functions to recalculate them.

The moments' values are stored in the *pState* structure. To retrieve a particular moment value, use one of the functions described in the sections that follow.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pSrc</i> or <i>pState</i> pointer is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> has a zero or negative value.
<code>ippStsContextMatchErr</code>	Indicates an error condition if a pointer to an invalid structure is passed.



## GetSpatialMoment

*Retrieves image spatial moment of the specified order, computed by `ippiMoments`.*

---

### Syntax

```
IppStatus ippiGetSpatialMoment_64f(const IppiMomentState_64f* pState, int mOrd, int nOrd,
int nChannel, IppiPoint roiOffset, Ipp64f* pValue);
```

```
IppStatus ippiGetSpatialMoment_64s(const IppiMomentState_64s* pState, int mOrd, int nOrd,
int nChannel, IppiPoint roiOffset, Ipp64s* pValue, int scaleFactor);
```

### Parameters

<code>pState</code>	Pointer to the structure that stores image moments.
<code>mOrd, nOrd</code>	Integer power exponents defining the moment order. These arguments must satisfy the condition $0 \leq mOrd + nOrd \leq 3$ .
<code>nChannel</code>	The channel for which the moment is returned.
<code>roiOffset</code>	Offset in pixels of the ROI origin (top left corner) from the image origin.
<code>pValue</code>	Pointer to the retrieved moment value.
<code>scaleFactor</code>	Scale factor (see <a href="#">Integer Result Scaling</a> ).

### Description

The function `ippiGetSpatialMoment` is declared in the `ippi.h` file. This function returns the pointer `pValue` to the spatial moment that was previously computed by the `ippiMoments` function. All spatial moment values are computed by `ippiMoments` relative to the image ROI origin. You may also obtain spatial moment values relative to different point in the image, using the appropriate `roiOffset` settings.

The moment order is specified by the integer exponents `mOrd`, `nOrd`.

Different functions are used to retrieve spatial moment in integer and floating-point formats, respectively. In case of integer data, the result may be scaled by the specified `scaleFactor`.

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <code>pState</code> or <code>pValue</code> pointer is <code>NULL</code> .
<code>ippStsContextMatchErr</code>	Indicates an error condition if a pointer to an invalid structure is passed.
<code>ippStsSizeErr</code>	Indicates an error condition if $mOrd + nOrd$ is greater than 3, or <code>nChannel</code> has an illegal value.

## GetCentralMoment

*Retrieves image central moment computed by `ippiMoments`.*

---

### Syntax

```
IppStatus ippiGetCentralMoment_64f(const IppiMomentState_64f* pState, int mOrd, int nOrd,
int nChannel, Ipp64f* pValue);
```

```
IppStatus ippiGetCentralMoment_64s(const IppiMomentState_64s* pState, int mOrd, int nOrd,
int nChannel, Ipp64s* pValue, int scaleFactor);
```

### Parameters

<i>pState</i>	The structure that stores image moments.
<i>mOrd</i> , <i>nOrd</i>	Integer power exponents defining the moment order. These arguments must satisfy the condition $0 \leq mOrd + nOrd \leq 3$ .
<i>nChannel</i>	The channel for which the moment is returned.
<i>pValue</i>	Pointer to the returned moment value.
<i>scaleFactor</i>	Scale factor (see <a href="#">Integer Result Scaling</a> ).

### Description

The function `ippiGetCentralMoment` is declared in the `ippi.h` file. This function returns the pointer *pValue* to the central moment previously computed by the [ippiMoments](#) function. The moment order is specified by the integer exponents *mOrd*, *nOrd*.

Different functions are used to retrieve central moment in integer and floating-point formats, respectively. In case of integer data, the result may be scaled by the specified *scaleFactor*.

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pState</i> or <i>pValue</i> pointer is <code>NULL</code> .
<code>ippStsContextMatchErr</code>	Indicates an error condition if a pointer to an invalid structure is passed.
<code>ippStsSizeErr</code>	Indicates an error condition if $mOrd + nOrd$ is greater than 3, or <i>nChannel</i> has an illegal value.

## GetNormalizedSpatialMoment

*Retrieves the normalized value of the image spatial moment computed by `ippiMoments`.*

---

### Syntax

```
IppStatus ippiGetNormalizedSpatialMoment_64f(IppiMomentState_64f* pState, int mOrd, int
nOrd, int nChannel, IppiPoint roiOffset, Ipp64f* pValue);
```

```
IppStatus ippiGetNormalizedSpatialMoment_64s(IppiMomentState_64s* pState, int mOrd, int nOrd, int nChannel, IppiPoint roiOffset, Ipp64s* pValue, int scaleFactor);
```

## Parameters

<i>pState</i>	The structure that stores image moments.
<i>mOrd</i> , <i>nOrd</i>	Integer power exponents defining the moment order. These arguments must satisfy the condition $0 \leq mOrd + nOrd \leq 3$ .
<i>nChannel</i>	The channel for which the moment is returned.
<i>roiOffset</i>	Offset in pixels of the ROI origin (top left corner) from the image origin.
<i>pValue</i>	Pointer to the returned normalized moment value.
<i>scaleFactor</i>	Scale factor (see <a href="#">Integer Result Scaling</a> ).

## Description

The function `ippiGetNormalizedSpatialMoment` is declared in the `ippi.h` file. This function normalizes the spatial moment value that was previously computed by the `ippiMoments` function, and returns the pointer *pValue* to the normalized moment. See [Image Moments](#) for details of moments normalization. The moment order (*mOrd*, *nOrd*) is specified by integer power exponents. All spatial moment values are computed by `ippiMoments` relative to the image ROI origin. You may also obtain normalized spatial moment values relative to different point in the image, using the appropriate *roiOffset* settings.

Different functions are used to retrieve normalized spatial moment in integer and floating-point formats, respectively. In case of integer data, the result may be scaled by the specified *scaleFactor*.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pState</i> or <i>pValue</i> pointer is NULL.
<code>ippStsContextMatchErr</code>	Indicates an error condition if a pointer to an invalid structure is passed.
<code>ippStsMoment00ZeroErr</code>	Indicates an error condition if $M(0,0)$ value is close to zero.
<code>ippStsSizeErr</code>	Indicates an error condition if $mOrd + nOrd$ is greater than 3, or <i>nChannel</i> has an illegal value.

## GetNormalizedCentralMoment

*Retrieves the normalized value of the image central moment computed by `ippiMoments`.*

---

### Syntax

```
IppStatus ippiGetNormalizedCentralMoment_64f(IppiMomentState_64f* pState, int mOrd, int nOrd, int nChannel, Ipp64f* pValue);
```

```
IppStatus ippiGetNormalizedCentralMoment_64s(IppiMomentState_64s* pState, int mOrd, int nOrd, int nChannel, Ipp64s* pValue, int scaleFactor);
```

## Parameters

<i>pState</i>	The structure that stores image moments.
<i>mOrd</i> , <i>nOrd</i>	Integer power exponents defining the moment order. These arguments must satisfy the condition $0 \leq mOrd + nOrd \leq 3$ .
<i>nChannel</i>	The channel for which the moment is returned.
<i>pValue</i>	Pointer to the returned moment value.
<i>scaleFactor</i>	Scale factor (see <a href="#">Integer Result Scaling</a> ).

## Description

The function `ippiGetNormalizedCentralMoment` is declared in the `ippi.h` file. This function normalizes the central moment value that was previously computed by the [ippiMoments](#) function, and returns the pointer *pValue* to the normalized moment. The moment order (*mOrd*, *nOrd*) is specified by the integer power exponents. See [Image Moments](#) for details of moments normalization. Different functions are used to retrieve normalized central moment in integer and floating-point formats, respectively. In case of integer data, the result may be scaled by the specified *scaleFactor*.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pState</i> or <i>pValue</i> pointer is <code>NULL</code> .
<code>ippStsContextMatchErr</code>	Indicates an error condition if a pointer to an invalid structure is passed.
<code>ippStsMoment00ZeroErr</code>	Indicates an error condition if $M(0,0)$ value is close to zero.

## GetHuMoments

*Retrieves image Hu moment invariants computed by `ippiMoments` function.*

---

## Syntax

```
IppStatus ippiGetHuMoments_64f(IppiMomentState_64f* pState, int nChannel, IppiHuMoment_64f* pHm);
```

```
IppStatus ippiGetHuMoments_64s(IppiMomentState_64s* pState, int nChannel, IppiHuMoment_64s* pHm, int scaleFactor);
```

## Parameters

<i>pState</i>	Pointer to the structure that stores image moments.
<i>nChannel</i>	The channel for which the moment is returned.
<i>pHm</i>	Pointer to the array containing the Hu moment invariants.
<i>scaleFactor</i>	Scale factor (see <a href="#">Integer Result Scaling</a> ).

## Description

The function `ippiGetHuMoments` is declared in the `ippi.h` file. This function returns the pointer `pHm` to the array of seven Hu moment invariants previously computed by the `ippiMoments` function. Different functions are used to retrieve Hu moments in integer and floating-point formats, respectively. In case of integer data, the result may be scaled by the specified `scaleFactor`.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <code>pState</code> or <code>pHm</code> pointer is <code>NULL</code> .
<code>ippStsContextMatchErr</code>	Indicates an error condition if a pointer to an invalid structure is passed.
<code>ippStsMoment00ZeroErr</code>	Indicates an error condition if $M(0,0)$ value is close to zero.

## Example

The code example below shows how to compute Hu moment values for images that have horizontal and vertical lines.

```
IppStatus mom8u( void) {
    IppStatus stH, stV;
    Ipp8u x[64];
    IppiSize roiA={8,8}, roiH={8,1}, roiV={1,8};
    IppiHuMoment_64f hmH, hmV;
    IppiMomentState_64f*ctx;
    stH = ippiMomentInitAlloc_64f(&ctx, ippAlgHintNone );
    ippiSet_8u_C1R( 0, x, 8, roiA );
    ippiSet_8u_C1R( 3, x, 8, roiH );
    stH = ippiMoments64f_8u_C1R( x, 8, roiA, ctx );
    ippiGetHuMoments_64f( ctx, 0, hmH );
    ippiSet_8u_C1R( 0, x, 8, roiA );
    ippiSet_8u_C1R( 3, x, 8, roiV );
    stV = ippiMoments64f_8u_C1R( x, 8, roiA, ctx );
    ippiGetHuMoments_64f( ctx, 0, hmV );
    ippiMomentFree_64f( ctx );
    return ippStsNoErr==stH ? stV : stH;
}
```

## Image Norms

The functions described in this section compute the following norms of the image pixel values:

- Infinity norm (the largest absolute pixel value)
- L1 norm (the sum of absolute pixel values)
- L2 norm (the square root of the sum of squared pixel values).

Functions of this group also help you compute the norm of differences in pixel values of two input images as well as the relative error for two input images.

## Norm\_Inf

*Computes the infinity norm of image pixel values.*

---

### Syntax

#### Case 1: Operation on one-channel data

```
ippStatus ippNorm_Inf_<mod>(const Ipp<datatype>* pSrc, int srcStep, IppiSize roiSize,
Ipp64f* pValue);
```

Supported values for mod:

8u\_C1R      16u\_C1R      16s\_C1R      32s\_C1R      32f\_C1R

#### Case 2: Masked operation on one-channel data

```
ippStatus ippNorm_Inf_<mod>(const Ipp<datatype>* pSrc, int srcStep, const Ipp8u* pMask,
int maskStep, IppiSize roiSize, Ipp64f* pNorm);
```

Supported values for mod:

8u\_C1MR      8s\_C1MR      16u\_C1MR      32f\_C1MR

#### Case 3: Operation on multi-channel data

```
ippStatus ippNorm_Inf_<mod>(const Ipp<datatype>* pSrc, int srcStep, IppiSize roiSize,
Ipp64f value[3]);
```

Supported values for mod:

8u\_C3R      16u\_C3R      16s\_C3R      32f\_C3R  
8u\_AC4R      16u\_AC4R      16s\_AC4R      32f\_AC4R

```
ippStatus ippNorm_Inf_<mod>(const Ipp<datatype>* pSrc, int srcStep, IppiSize roiSize,
Ipp64f value[4]);
```

Supported values for mod:

8u\_C4R      16u\_C4R      16s\_C4R      32f\_C4R

**Case 4: Masked operation on multi-channel data**

```
IppStatus ippiNorm_Inf_<mod>(const Ipp<datatype>* pSrc, int srcStep, const Ipp8u* pMask,
int maskStep, IppiSize roiSize, int coi, Ipp64f* pNorm);
```

Supported values for `mod`:

8u\_C3CMR

8s\_C3CMR

16u\_C3CMR

32f\_C3CMR

**Parameters**

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pMask</i>	Pointer to the mask image.
<i>maskStep</i>	Distance in bytes between starts of consecutive lines in the mask image.
<i>roiSize</i>	Size of the source ROI in pixels.
<i>coi</i>	Channel of interest (for color images only); can be 1, 2, or 3.
<i>pValue</i>	Pointer to the computed infinity norm of pixel values.
<i>value</i>	An array containing the computed infinity norms of channel values in case of multi-channel data.
<i>pNorm</i>	Pointer to the computed norm value in the mask mode.

**Description**

The flavors of the function `ippiNorm_Inf` that perform masked operation are declared in the `ippcv.h` file. All other function flavors are declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)) and computes the infinity norm *pValue* (*pNorm* for the mask mode) for the source image *pSrc*. This norm is defined as the largest absolute pixel value in an image. In the mask mode, the computation is done over pixels selected by non-zero mask values.

For non-masked operations on a multi-channel image (*Case 3*), the norm is computed separately for each channel and stored in the array *value*.

In the mask multi-channel mode (*Case 4*), the norm is computed for a single channel of interest specified by *coi*.

**Return Values**

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error when any of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition in mask mode, if <i>srcStep</i> or <i>maskStep</i> is less than <i>roiSize.width</i> * <code>&lt;pixelSize&gt;</code> .
<code>ippStsNotEvenStepErr</code>	Indicates an error condition in mask mode if steps for floating-point images cannot be divided by 4.

`ippStsCOIErr`Indicates an error when `coi` is not 1, 2, or 3.

## Norm\_L1

*Computes the L1- norm of image pixel values.*

---

### Syntax

#### Case 1: Operation on one-channel integer data

```
IppStatus ippNorm_L1_<mod>(const Ipp<datatype>* pSrc, int srcStep, IppiSize roiSize,
Ipp64f* pValue);
```

Supported values for `mod`:

8u_C1R	16u_C1R	16s_C1R
--------	---------	---------

#### Case 2: Operation on one-channel floating-point data

```
IppStatus ippNorm_L1_32f_C1R(const Ipp32f* pSrc, int srcStep, IppiSize roiSize, Ipp64f*
pValue, IppHintAlgorithm hint);
```

#### Case 3: Masked operation on one-channel data

```
IppStatus ippNorm_L1_<mod>(const Ipp<datatype>* pSrc, int srcStep, const Ipp8u* pMask,
int maskStep, IppiSize roiSize, Ipp64f* pNorm);
```

Supported values for `mod`:

8u_C1MR	8s_C1MR	16u_C1MR	32f_C1MR
---------	---------	----------	----------

#### Case 4: Operation on multi-channel integer data

```
IppStatus ippNorm_L1_<mod>(const Ipp<datatype>* pSrc, int srcStep, IppiSize roiSize,
Ipp64f value[3]);
```

Supported values for `mod`:

8u_C3R	16u_C3R	16s_C3R
8u_AC4R	16u_AC4R	16s_AC4R

```
IppStatus ippNorm_L1_<mod>(const Ipp<datatype>* pSrc, int srcStep, IppiSize roiSize,
Ipp64f value[4]);
```

Supported values for `mod`:

8u_C4R	16u_C4R	16s_C4R
--------	---------	---------



**Case 5: Operation on multi-channel floating-point data**

```
IppStatus ippiNorm_L1_<mod>(const Ipp32f* pSrc, int srcStep, IppiSize roiSize, Ipp64f
value[3], IppHintAlgorithm hint);
```

Supported values for `mod`:

32f\_C3R

32f\_AC4R

```
IppStatus ippiNorm_L1_32f_C4R(const Ipp32f* pSrc, int srcStep, IppiSize roiSize, Ipp64f
value[4], IppHintAlgorithm hint);
```

**Case 6: Masked operation on multi-channel data**

```
IppStatus ippiNorm_L1_<mod>(const Ipp<datatype>* pSrc, int srcStep, const Ipp8u* pMask,
int maskStep, IppiSize roiSize, int coi, Ipp64f* pNorm);
```

Supported values for `mod`:

8u\_C3CMR

8s\_C3CMR

16u\_C3CMR

32f\_C3CMR

**Parameters**

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pMask</i>	Pointer to the mask image.
<i>maskStep</i>	Distance in bytes between starts of consecutive lines in the mask image.
<i>roiSize</i>	Size of the source ROI in pixels.
<i>coi</i>	Channel of interest (for color images only); can be 1, 2, or 3.
<i>pValue</i>	Pointer to the computed L1- norm of pixel values.
<i>value</i>	An array containing the computed L1- norms of channel values in case of multi-channel data.
<i>pNorm</i>	Pointer to the computed norm value in the mask mode.
<i>hint</i>	Option to select the algorithmic implementation of the function.

**Description**

The flavors of the function `ippiNorm_L1` that perform masked operation are declared in the `ippcv.h` file. All other function flavors are declared in the `ippi.h` file. The function operates with ROI (see [Regions of Interest in Intel IPP](#)). It computes the L1- norm *pValue* (*pNorm* in mask mode) for the source image *pSrc*. This norm is defined as the sum of absolute pixel values in an image. Computation algorithm is specified by the *hint* argument (see [Table “Hint Arguments for Image Moment Functions”](#)). In the mask mode, the computation is done over pixels selected by nonzero mask values.

For non-masked operations on a multi-channel image (*Case 4, 5*), the norm is computed separately for each channel and stored in the array *value*.

In the mask multi-channel mode (*Case 6*), the norm is computed for a single channel of interest specified by *coi*.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error when any of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition in mask mode, if <i>srcStep</i> or <i>maskStep</i> is less than <i>roiSize.width</i> * <i>&lt;pixelSize&gt;</i> .
<code>ippStsNotEvenStepErr</code>	Indicates an error condition in mask mode if steps for floating-point images cannot be divided by 4.
<code>ippStsCOIErr</code>	Indicates an error when <i>coi</i> is not 1, 2, or 3.

## Example

The code example below demonstrates how an image norm can be computed.

```
IppStatus norm( void ){
    Ipp64f sum, normL1;
    Ipp8u x[5*4];
    IppiSize roi = {5,4};
    ippiSet_8u_C1R( 1, x, 5, roi );
    ippiSum_8u_C1R( x, 5, roi, &sum);
    return ippiNorm_L1_8u_C1R( x, 5, roi, &normL1 );
}
```

## Norm\_L2

*Computes the L2- norm of image pixel values.*

---

### Syntax

#### Case 1: Operation on one-channel integer data

```
IppStatus ippiNorm_L2_<mod>(const Ipp<datatype>* pSrc, int srcStep, IppiSize roiSize,
Ipp64f* pValue);
```

Supported values for *mod*:

8u\_C1R            16u\_C1R            16s\_C1R

**Case 2: Operation on one-channel floating-point data**

```
IppStatus ippiNorm_L2_32f_C1R(const Ipp32f* pSrc, int srcStep, IppiSize roiSize, Ipp64f* pValue, IppHintAlgorithm hint);
```

**Case 3: Masked operation on one-channel data**

```
IppStatus ippiNorm_L2_<mod>(const Ipp<datatype>* pSrc, int srcStep, const Ipp8u* pMask, int maskStep, IppiSize roiSize, Ipp64f* pNorm);
```

Supported values for mod:

8u_C1MR	8s_C1MR	16u_C1MR	32f_C1MR
---------	---------	----------	----------

**Case 4: Operation on multi-channel integer data**

```
IppStatus ippiNorm_L2_<mod>(const Ipp<datatype>* pSrc, int srcStep, IppiSize roiSize, Ipp64f value[3]);
```

Supported values for mod:

8u_C3R	16u_C3R	16s_C3R
8u_AC4R	16u_AC4R	16s_AC4R

```
IppStatus ippiNorm_L2_<mod>(const Ipp<datatype>* pSrc, int srcStep, IppiSize roiSize, Ipp64f value[4]);
```

Supported values for mod:

8u_C4R	16u_C4R	16s_C4R
--------	---------	---------

**Case 5: Operation on multi-channel floating-point data**

```
IppStatus ippiNorm_L2_<mod>(const Ipp32f* pSrc, int srcStep, IppiSize roiSize, Ipp64f value[3], IppHintAlgorithm hint);
```

Supported values for mod:

32f_C3R
32f_AC4R

```
IppStatus ippiNorm_L2_32f_C4R(const Ipp32f* pSrc, int srcStep, IppiSize roiSize, Ipp64f value[3], IppHintAlgorithm hint);
```

**Case 6: Masked operation on multi-channel data**

```
IppStatus ippiNorm_L2_<mod>(const Ipp<datatype>* pSrc, int srcStep, const Ipp8u* pMask, int maskStep, IppiSize roiSize, int coi, Ipp64f* pNorm);
```

Supported values for mod:

8u_C3CMR	8s_C3CMR	16u_C3CMR	32f_C3CMR
----------	----------	-----------	-----------

## Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pMask</i>	Pointer to the mask image.
<i>maskStep</i>	Distance in bytes between starts of consecutive lines in the mask image.
<i>roiSize</i>	Size of the source ROI in pixels.
<i>coi</i>	Channel of interest (for color images only); can be 1, 2, or 3.
<i>pValue</i>	Pointer to the computed L2- norm of pixel values.
<i>value</i>	An array containing the computed L2- norms of channel values in case of multi-channel data.
<i>pNorm</i>	Pointer to the computed norm value in the mask mode.
<i>hint</i>	Option to select the algorithmic implementation of the function.

## Description

The flavors of the function `ippiNorm_L2` that perform masked operation are declared in the `ippcv.h` file. All other function flavors are declared in the `ippi.h` file. The function operates with ROI (see [Regions of Interest in Intel IPP](#)). It computes the L2- norm *pValue* (*pNorm* in mask mode) for the source image *pSrc*. This norm is defined as the square root of the sum of squared pixel values in an image. Computation algorithm is specified by the *hint* argument (see [Table “Hint Arguments for Image Moment Functions”](#)). In the mask mode, the computation is done over pixels selected by nonzero mask values.

For non-masked operations on a multi-channel image (*Case 4,5*), the norm is computed separately for each channel and stored in the array *value*.

In the mask multi-channel mode (*Case 6*), the norm is computed for a single channel of interest specified by *coi*.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error when any of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition in mask mode, if <i>srcStep</i> or <i>maskStep</i> is less than <i>roiSize.width</i> * <code>&lt;pixelSize&gt;</code> .
<code>ippStsNotEvenStepErr</code>	Indicates an error condition in mask mode if steps for floating-point images cannot be divided by 4.
<code>ippStsCOIErr</code>	Indicates an error when <i>coi</i> is not 1, 2, or 3.

## NormDiff\_Inf

*Computes the infinity norm of differences between pixel values of two images.*

---

### Syntax

#### Case 1: Operation on one-channel data

```
IppStatus ippiNormDiff_Inf_<mod>(const Ipp<datatype>* pSrc1, int src1Step, const
Ipp<datatype>* pSrc2, int src2Step, IppiSize roiSize, Ipp64f* pValue);
```

Supported values for mod:

8u_C1R	16u_C1R	16s_C1R	32f_C1R
--------	---------	---------	---------

#### Case 2: Masked operation on one-channel data

```
IppStatus ippiNormDiff_Inf_<mod>(const Ipp<datatype>* pSrc1, int src1Step, const
Ipp<datatype>* pSrc2, int src2Step, const Ipp8u* pMask, int maskStep, IppiSize roiSize,
Ipp64f* pNorm);
```

Supported values for mod:

8u_C1MR	8s_C1MR	16u_C1MR	32f_C1MR
---------	---------	----------	----------

#### Case 3: Operation on multi-channel data

```
IppStatus ippiNormDiff_Inf_<mod>(const Ipp<datatype>* pSrc1, int src1Step, const
Ipp<datatype>* pSrc2, int src2Step, IppiSize roiSize, Ipp64f value[3]);
```

Supported values for mod:

8u_C3R	16u_C3R	16s_C3R	32f_C3R
8u_AC4R	16u_AC4R	16s_AC4R	32f_AC4R

```
IppStatus ippiNormDiff_Inf_<mod>(const Ipp<datatype>* pSrc1, int src1Step, const
Ipp<datatype>* pSrc2, int src2Step, IppiSize roiSize, Ipp64f value[4]);
```

Supported values for mod:

8u_C4R	16u_C4R	16s_C4R	32f_C4R
--------	---------	---------	---------

## Case 4: Masked operation on multi-channel data

```
IppStatus ippiNormDiff_Inf_<mod>(const Ipp<datatype>* pSrc1, int src1Step, const
Ipp<datatype>* pSrc2, int src2Step, const Ipp8u* pMask, int maskStep, IppiSize roiSize,
int coi, Ipp64f* pNorm);
```

Supported values for `mod`:

8u\_C3CMR      8s\_C3CMR      16u\_C3CMR      32f\_C3CMR

## Parameters

<code>pSrc1, pSrc2</code>	Pointers to the source images ROI.
<code>src1Step, src2Step</code>	Distance in bytes between starts of consecutive lines in the source images.
<code>pMask</code>	Pointer to the mask image.
<code>maskStep</code>	Distance in bytes between starts of consecutive lines in the mask image.
<code>roiSize</code>	Size of the source ROI in pixels.
<code>pValue</code>	Pointer to the computed infinity norm of difference between pixel values.
<code>value</code>	An array containing the computed infinity norms of difference between corresponding channel values in case of multi-channel data.
<code>coi</code>	Channel of interest (for color images only); can be 1, 2, or 3.
<code>pNorm</code>	Pointer to the computed norm value in the mask mode.

## Description

The flavors of the function `ippiNormDiff_Inf` that perform masked operation are declared in the `ippcv.h` file. All other function flavors are declared in the `ippi.h` file. The function operates with ROI (see [Regions of Interest in Intel IPP](#)). It computes the infinity norm `pValue` (`pNorm` in the mask mode) of differences between pixel values of the two source images `pSrc1` and `pSrc2`. This norm is defined as the largest absolute value of differences:

$$\text{norm} = \max |pSrc1 - pSrc2|$$

In the mask mode, the computation is done over pixels selected by nonzero mask values.

For non-masked operations on multi-channel images (*Case 3*), the norm is computed separately for each pair of corresponding channels and stored in the array `value`.

In the mask multi-channel mode (*Case 4*), the norm is computed for a single channel of interest specified by `coi`.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error when any of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <code>roiSize</code> has a field with zero or negative value.

<code>ippStsStepErr</code>	Indicates an error condition in mask mode, if <code>src1Step</code> , <code>src2Step</code> , or <code>maskStep</code> is less than <code>roiSize.width * &lt;pixelSize&gt;</code> .
<code>ippStsNotEvenStepErr</code>	Indicates an error condition in mask mode if steps for floating-point images cannot be divided by 4.
<code>ippStsCOIErr</code>	Indicates an error when <code>coi</code> is not 1, 2, or 3.

## NormDiff\_L1

*Computes the L1- norm of differences between pixel values of two images.*

### Syntax

#### Case 1: Operation on one-channel integer data

```
IppStatus ippiNormDiff_L1_<mod>(const Ipp<datatype>* pSrc1, int src1Step, const Ipp<datatype>* pSrc2, int src2Step, IppiSize roiSize, Ipp64f* pValue);
```

Supported values for `mod`:

8u\_C1R      16u\_C1R      16s\_C1R

#### Case 2: Operation on one-channel floating-point data

```
IppStatus ippiNormDiff_L1_32f_C1R(const Ipp32f* pSrc1, int src1Step, const Ipp32f* pSrc2, int src2Step, IppiSize roiSize, Ipp64f* pValue, IppHintAlgorithm hint);
```

#### Case 3: Masked operation on one-channel data

```
IppStatus ippiNormDiff_L1_<mod>(const Ipp<datatype>* pSrc1, int src1Step, const Ipp32f* pSrc2, int src2Step, const Ipp8u* pMask, int maskStep, IppiSize roiSize, Ipp64f* pNorm);
```

Supported values for `mod`:

8u\_C1MR      8s\_C1MR      16u\_C1MR      32f\_C1MR

#### Case 4: Operation on multi-channel integer data

```
IppStatus ippiNormDiff_L1_<mod>(const Ipp<datatype>* pSrc1, int src1Step, const Ipp<datatype>* pSrc2, int src2Step, IppiSize roiSize, Ipp64f value[3]);
```

Supported values for `mod`:

8u\_C3R      16u\_C3R      16s\_C3R  
8u\_AC4R      16u\_AC4R      16s\_AC4R

```
IppStatus ippiNormDiff_L1_<mod>(const Ipp<datatype>* pSrc1, int src1Step, const
Ipp<datatype>* pSrc2, int src2Step, IppiSize roiSize, Ipp64f value[4]);
```

Supported values for mod:

8u\_C4R            16u\_C4R            16s\_C4R

## Case 5: Operation on multi-channel floating-point data

```
IppStatus ippiNormDiff_L1_<mod>(const Ipp32f* pSrc1, int src1Step, const Ipp32f* pSrc2,
int v, IppiSize roiSize, Ipp64f value[3], IppHintAlgorithm hint);
```

Supported values for mod:

32f\_C3R  
32f\_AC4R

```
IppStatus ippiNormDiff_L1_32f_C4R(const Ipp32f* pSrc1, int src1Step, const Ipp32f* pSrc2,
int src2Step, IppiSize roiSize, Ipp64f value[4], IppHintAlgorithm hint);
```

## Case 6: Masked operation on multi-channel data

```
IppStatus ippiNormDiff_L1_<mod>(const Ipp<datatype>* pSrc1, int src1Step, const Ipp32f*
pSrc2, int src2Step, const Ipp8u* pMask, int maskStep, IppiSize roiSize, int coi, Ipp64f*
pNorm);
```

Supported values for mod:

8u\_C3CMR            8s\_C3CMR            16u\_C3CMR            32f\_C3CMR

## Parameters

<i>pSrc1, pSrc2</i>	Pointers to the source images ROI.
<i>src1Step, src2Step</i>	Distance in bytes between starts of consecutive lines in the source images.
<i>pMask</i>	Pointer to the mask image.
<i>maskStep</i>	Distance in bytes between starts of consecutive lines in the mask image.
<i>roiSize</i>	Size of the source ROI in pixels.
<i>coi</i>	Channel of interest (for color images only); can be 1, 2, or 3.
<i>pValue</i>	Pointer to the computed L1- norm of difference between pixel values.
<i>value</i>	An array containing the computed L1- norms of difference between channel values in case of multi-channel data.
<i>pNorm</i>	Pointer to the computed norm value in the mask mode.
<i>hint</i>	Option to select the algorithmic implementation of the function.



## Description

The flavors of the function `ippiNormDiff_L1` that perform masked operation are declared in the `ippcv.h` file. All other function flavors are declared in the `ippi.h` file. The function operates with ROI (see [Regions of Interest in Intel IPP](#)). It computes the L1-norm *pValue* (*pNorm* in the mask mode) of differences between pixel values of the two source image buffers *pSrc1* and *pSrc2*. This norm is defined as the sum of absolute values of differences:

$$\text{norm} = \sum |pSrc1 - pSrc2|$$

Computation algorithm is specified by the *hint* argument (see [Table "Hint Arguments for Image Moment Functions"](#)). In the mask mode, the computation is done over pixels selected by nonzero mask values.

For non-masked operations on multi-channel images (*Case 4,5*), the norm is computed separately for each pair of the corresponding channels and stored in the array *value*.

In the mask multi-channel mode (*Case 6*), the norm is computed for a single channel of interest specified by *coi*.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error when any of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition in mask mode, if <i>src1Step</i> , <i>src2Step</i> , or <i>maskStep</i> is less than <i>roiSize.width</i> * <i>&lt;pixelSize&gt;</i> .
<code>ippStsNotEvenStepErr</code>	Indicates an error condition in mask mode if steps for floating-point images cannot be divided by 4.
<code>ippStsCOIErr</code>	Indicates an error when <i>coi</i> is not 1, 2, or 3.

## Example

The code example below shows how to use the function `ippiNormDiff_L1_8u_C1R`.

```
void func_normdiff_l1()
{
    Ipp8u pSrc1[8*4];
    Ipp8u pSrc2[8*4];

    Ipp64f Value;
    int src1Step = 8;
    int src2Step = 8;
    IppiSize roi = {8,4};
    IppiSize roiSize = {5,4};
```

```
ippiSet_8u_C1R(1, pSrc1, src1Step, roi);
ippiSet_8u_C1R(2, pSrc2, src2Step, roi);
```

```
    ippiNormDiff_L1_8u_C1R( pSrc1, src1Step, pSrc2, src2Step, roiSize, &Value);
}
Result -> 20.0
```

## NormDiff\_L2

*Computes the L2- norm of differences between pixel values of two images.*

---

### Syntax

#### Case 1: Operation on one-channel integer data

```
IppStatus ippiNormDiff_L2_<mod>(const Ipp<datatype>* pSrc1, int src1Step, const
Ipp<datatype>* pSrc2, int src2Step, IppiSize roiSize, Ipp64f* pValue);
```

Supported values for mod:

8u\_C1R            16u\_C1R            16s\_C1R

#### Case 2: Operation on one-channel floating-point data

```
IppStatus ippiNormDiff_L2_32f_C1R(const Ipp32f* pSrc1, int src1Step, const Ipp32f* pSrc2,
int src2Step, IppiSize roiSize, Ipp64f* pValue, IppHintAlgorithm hint);
```

#### Case 3: Masked operation on one-channel data

```
IppStatus ippiNormDiff_L2_<mod>(const Ipp<datatype>* pSrc1, int src1Step, const Ipp32f*
pSrc2, int src2Step, const Ipp8u* pMask, int maskStep, IppiSize roiSize, Ipp64f* pNorm);
```

Supported values for mod:

8u\_C1MR            8s\_C1MR            16u\_C1MR            32f\_C1MR

#### Case 4: Operation on multi-channel integer data

```
IppStatus ippiNormDiff_L2_<mod>(const Ipp<datatype>* pSrc1, int src1Step, const
Ipp<datatype>* pSrc2, int src2Step, IppiSize roiSize, Ipp64f value[3]);
```

Supported values for mod:

8u\_C3R            16u\_C3R            16s\_C3R  
8u\_AC4R            16u\_AC4R            16s\_AC4R

```
IppStatus ippiNormDiff_L2_<mod>(const Ipp<datatype>* pSrc1, int src1Step, const
Ipp<datatype>* pSrc2, int src2Step, IppiSize roiSize, Ipp64f value[4]);
```

Supported values for mod:

8u\_C4R            16u\_C4R            16s\_C4R

### Case 5: Operation on multi-channel floating-point data

```
IppStatus ippiNormDiff_L2_<mod>(const Ipp32f* pSrc1, int src1Step, const Ipp32f* pSrc2,
int src2Step, IppiSize roiSize, Ipp64f value[3], IppHintAlgorithm hint);
```

Supported values for mod:

32f\_C3R

32f\_AC4R

```
IppStatus ippiNormDiff_L2_32f_C4R(const Ipp32f* pSrc1, int src1Step, const Ipp32f* pSrc2,
int src2Step, IppiSize roiSize, Ipp64f value[4], IppHintAlgorithm hint);
```

### Case 6: Masked operation on multi-channel data

```
IppStatus ippiNormDiff_L2_<mod>(const Ipp<datatype>* pSrc1, int src1Step, const Ipp32f*
pSrc2, int src2Step, const Ipp8u* pMask, int maskStep, IppiSize roiSize, int coi, Ipp64f*
norm);
```

Supported values for mod:

8u\_C3CMR            8s\_C3CMR            16u\_C3CMR            32f\_C3CMR

## Parameters

<i>pSrc1, pSrc2</i>	Pointers to the source images ROI.
<i>src1Step, src2Step</i>	Distance in bytes between starts of consecutive lines in the source images.
<i>pMask</i>	Pointer to the mask image.
<i>maskStep</i>	Distance in bytes between starts of consecutive lines in the mask image.
<i>roiSize</i>	Size of the source ROI in pixels.
<i>coi</i>	Channel of interest (for color images only); can be 1, 2, or 3.
<i>pValue</i>	Pointer to the computed L2- norm of difference between pixel values.
<i>value</i>	An array containing the computed L2- norms of difference between channel values in case of multi-channel data.
<i>pNorm</i>	Pointer to the computed norm value in the mask mode.
<i>hint</i>	Option to select the algorithmic implementation of the function.

## Description

The flavors of the function `ippiNormDiff_L2` that perform masked operation are declared in the `ippcv.h` file. All other function flavors are declared in the `ippi.h` file. The function operates with ROI (see [Regions of Interest in Intel IPP](#)). It computes the L2-norm *pValue* (*pNorm* in the mask mode) of differences between pixel values of the two source image buffers *pSrc1* and *pSrc2*. This norm is defined as the square root of the sum of squared differences:

$$\text{norm} = \sqrt{\sum |pSrc1 - pSrc2|^2}.$$

Computation algorithm is specified by the *hint* argument (see [Table "Hint Arguments for Image Moment Functions"](#)). In the mask mode, the computation is done over pixels selected by nonzero mask values.

For non-masked operations on multi-channel images (*Case 4,5*), the norm is computed separately for each pair of the corresponding channels and stored in the array *value*.

In the mask multi-channel mode (*Case 6*), the norm is computed for a single channel of interest specified by *coi*.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error when any of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition in mask mode, if <i>src1Step</i> , <i>src2Step</i> , or <i>maskStep</i> is less than <i>roiSize.width</i> * <i>&lt;pixelSize&gt;</i> .
<code>ippStsNotEvenStepErr</code>	Indicates an error condition in mask mode if steps for floating-point images cannot be divided by 4.
<code>ippStsCOIErr</code>	Indicates an error when <i>coi</i> is not 1, 2, or 3.

## NormRel\_Inf

*Computes the relative error for the infinity norm of differences between pixel values of two images.*

---

## Syntax

### Case 1: Operation on one-channel data

```
ippStatus ippiNormRel_Inf_<mod>(const Ipp<datatype>* pSrc1, int src1Step, const
Ipp<datatype>* pSrc2, int src2Step, IppiSize roiSize, Ipp64f* pValue);
```

Supported values for *mod*:

8u\_C1R      16u\_C1R      16s\_C1R      32f\_C1R

**Case 2: Masked operation on one-channel data**

```
IppStatus ippiNormRel_Inf_<mod>(const Ipp<datatype>* pSrc1, int src1Step, const
Ipp<datatype>* pSrc2, int src2Step, const Ipp8u* pMask, int maskStep, IppiSize roiSize,
Ipp64f* pNorm);
```

Supported values for mod:

8u\_C1MR      8s\_C1MR      16u\_C1MR      32f\_C1MR

**Case 3: Operation on multi-channel data**

```
IppStatus ippiNormRel_Inf_<mod>(const Ipp<datatype>* pSrc1, int src1Step, const
Ipp<datatype>* pSrc2, int src2Step, IppiSize roiSize, Ipp64f value[3]);
```

Supported values for mod:

8u\_C3R      16u\_C3R      16s\_C3R      32f\_C3R  
8u\_AC4R      16u\_AC4R      16s\_AC4R      32f\_AC4R

```
IppStatus ippiNormRel_Inf_<mod>(const Ipp<datatype>* pSrc1, int src1Step, const
Ipp<datatype>* pSrc2, int src2Step, IppiSize roiSize, Ipp64f value[4]);
```

Supported values for mod:

8u\_C4R      16u\_C4R      16s\_C4R      32f\_C4R

**Case 4: Masked operation on multi-channel data**

```
IppStatus ippiNormRel_Inf_<mod>(const Ipp<datatype>* pSrc1, int src1Step, const
Ipp<datatype>* pSrc2, int src2Step, const Ipp8u* pMask, int maskStep, IppiSize roiSize,
int coi, Ipp64f* pNorm);
```

Supported values for mod:

8u\_C3CMR      8s\_C3CMR      16u\_C3CMR      32f\_C3CMR

**Parameters**

<i>pSrc1, pSrc2</i>	Pointers to the source images ROI.
<i>src1Step, src2Step</i>	Distance in bytes between starts of consecutive lines in the source images.
<i>pMask</i>	Pointer to the mask image.
<i>maskStep</i>	Distance in bytes between starts of consecutive lines in the mask image.
<i>roiSize</i>	Size of the source ROI in pixels.
<i>pValue</i>	Pointer to the computed relative error value.
<i>value</i>	An array containing the computed relative error values for separate channels in case of multi-channel data.

<i>coi</i>	Channel of interest (for color images only); can be 1, 2, or 3.
<i>pNorm</i>	Pointer to the computed relative norm value in the mask mode.

## Description

The flavors of the function `ippiNormRel_Inf` that perform masked operation are declared in the `ippcv.h` file. All other function flavors are declared in the `ippi.h` file. The function operates with ROI (see [Regions of Interest in Intel IPP](#)). It computes the infinity norm of differences between pixel values of two source buffers *pSrc1* and *pSrc2*. This norm is defined as the largest absolute pixel value in an image. The output relative error *pValue* (*pNorm* in the mask mode) is then formed by dividing the computed norm of differences by the infinity norm of the second source image buffer *pSrc2*. In the mask mode, the computation is done over pixels selected by nonzero mask values.

For non-masked operations on multi-channel images (Case 3), the relative norm is computed separately for each pair of corresponding channels and stored in the array *value*.

In the mask multi-channel mode (Case 4), the relative norm is computed for a single channel of interest specified by *coi*.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error when any of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition in mask mode, if <i>src1Step</i> , <i>src2Step</i> , or <i>maskStep</i> is less than <i>roiSize.width</i> * <i>&lt;pixelSize&gt;</i> .
<code>ippStsNotEvenStepErr</code>	Indicates an error condition in mask mode if steps for floating-point images cannot be divided by 4.
<code>ippStsCOIErr</code>	Indicates an error when <i>coi</i> is not 1, 2, or 3.
<code>ippStsDivByZero</code>	Indicates a warning when the infinity norm of <i>pSrc2</i> has a zero value.

## NormRel\_L1

*Computes the relative error for the L1 norm of differences between pixel values of two images.*

---

## Syntax

### Case 1: Operation on one-channel integer data

```
ippStatus ippiNormRel_L1_<mod>(const Ipp<datatype>* pSrc1, int src1Step, const
Ipp<datatype>* pSrc2, int src2Step, IppiSize roiSize, Ipp64f* pValue);
```

Supported values for *mod*:

8u\_C1R      16u\_C1R      16s\_C1R

**Case 2: Operation on one-channel floating-point data**

```
IppStatus ippiNormRel_L1_32f_C1R(const Ipp32f* pSrc1, int src1Step, const Ipp32f* pSrc2,
int src2Step, IppiSize roiSize, Ipp64f* pValue, IppHintAlgorithm hint);
```

**Case 3: Masked operation on one-channel data**

```
IppStatus ippiNormRel_L1_<mod>(const Ipp<datatype>* pSrc1, int src1Step, const Ipp32f*
pSrc2, int src2Step, const Ipp8u* pMask, int maskStep, IppiSize roiSize, Ipp64f* pNorm);
```

Supported values for mod:

8u_C1MR	8s_C1MR	16u_C1MR	32f_C1MR
---------	---------	----------	----------

**Case 4: Operation on multi-channel integer data**

```
IppStatus ippiNormRel_L1_<mod>(const Ipp<datatype>* pSrc1, int src1Step, const
Ipp<datatype>* pSrc2, int src2Step, IppiSize roiSize, Ipp64f value[3]);
```

Supported values for mod:

8u_C3R	16u_C3R	16s_C3R
8u_AC4R	16u_AC4R	16s_AC4R

```
IppStatus ippiNormRel_L1_<mod>(const Ipp<datatype>* pSrc1, int src1Step, const
Ipp<datatype>* pSrc2, int src2Step, IppiSize roiSize, Ipp64f value[4]);
```

Supported values for mod:

8u_C4R	16u_C4R	16s_C4R
--------	---------	---------

**Case 5: Operation on multi-channel floating-point data**

```
IppStatus ippiNormRel_L1_<mod>(const Ipp32f* pSrc1, int src1Step, const Ipp32f* pSrc2,
int src2Step, IppiSize roiSize, Ipp64f value[3], IppHintAlgorithm hint);
```

Supported values for mod:

32f_C3R
32f_AC4R

```
IppStatus ippiNormRel_L1_32f_C4R(const Ipp32f* pSrc1, int src1Step, const Ipp32f* pSrc2,
int src2Step, IppiSize roiSize, Ipp64f value[4], IppHintAlgorithm hint);
```

## Case 6: Masked operation on multi-channel data

```
IppStatus ippiNormRel_L1_<mod>(const Ipp<datatype>* pSrc1, int src1Step, const Ipp32f*
pSrc2, int src2Step, const Ipp8u* pMask, int maskStep, IppiSize roiSize, int coi, Ipp64f*
norm);
```

Supported values for mod:

8u\_C3CMR      8s\_C3CMR      16u\_C3CMR      32f\_C3CMR

## Parameters

<i>pSrc1, pSrc2</i>	Pointers to the source images ROI.
<i>src1Step, src2Step</i>	Distance in bytes between starts of consecutive lines in the source images.
<i>pMask</i>	Pointer to the mask image.
<i>maskStep</i>	Distance in bytes between starts of consecutive lines in the mask image.
<i>roiSize</i>	Size of the source ROI in pixels.
<i>coi</i>	Channel of interest (for color images only); can be 1, 2, or 3.
<i>pValue</i>	Pointer to the computed relative error value.
<i>value</i>	An array containing the computed relative error values for separate channels in case of multi-channel data.
<i>pNorm</i>	Pointer to the computed relative norm value in the mask mode.
<i>hint</i>	Option to select the algorithmic implementation of the function.

## Description

The flavors of the function `ippiNormRel_L1` that perform masked operation are declared in the `ippcv.h` file. All other function flavors are declared in the `ippi.h` file. The function operates with ROI (see [Regions of Interest in Intel IPP](#)). It computes the L1- norm of differences between pixel values of two source buffers *pSrc1* and *pSrc2*. This norm is defined as the sum of absolute pixel values in an image. The output relative error *pValue* (*pNorm* in the mask mode) is then formed by dividing the computed norm of differences by the L1- norm of the second source image buffer *pSrc2*. Computation algorithm is specified by the *hint* argument (see [Table "Hint Arguments for Image Moment Functions"](#)). In the mask mode, the computation is done over pixels selected by nonzero mask values.

For non-masked operations on multi-channel images (*Cases 4, 5*), the relative norm is computed separately for each pair of corresponding channels and stored in the array *value*.

In the mask multi-channel mode (*Case 6*), the relative norm is computed for a single channel of interest specified by *coi*.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error when any of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.



<code>ippStsStepErr</code>	Indicates an error condition in mask mode, if <code>src1Step</code> , <code>src2Step</code> , or <code>maskStep</code> is less than <code>roiSize.width * &lt;pixelSize&gt;</code> .
<code>ippStsNotEvenStepErr</code>	Indicates an error condition in mask mode if steps for floating-point images cannot be divided by 4.
<code>ippStsCOIErr</code>	Indicates an error when <code>coi</code> is not 1, 2, or 3.
<code>ippStsDivByZero</code>	Indicates a warning when the L1 norm of <code>pSrc2</code> has a zero value.

## Example

The code example below shows how to use the function `ippiNormRel_L1_8u_C1R`.

```
void func_normrel_l1()
{
    Ipp8u pSrc1[8*4];
    Ipp8u pSrc2[8*4];
    Ipp64f Value;
    int src1Step = 8;
    int src2Step = 8;
    IppiSize roi = {8,4};
    IppiSize roiSize = {5,4};

    ippiSet_8u_C1R(1, pSrc1, src1Step, roi);
    ippiSet_8u_C1R(2, pSrc2, src2Step, roi);
    ippiNormRel_L1_8u_C1R( pSrc1, src1Step, pSrc2, src2Step, roiSize, &Value);

}
Result -> 0.5
```

## NormRel\_L2

*Computes the relative error for the L2 norm of differences between pixel values of two images.*

### Syntax

#### Case 1: Operation on one-channel integer data

```
IppStatus ippiNormRel_L2_<mod>(const Ipp<datatype>* pSrc1, int src1Step, const
Ipp<datatype>* pSrc2, int src2Step, IppiSize roiSize, Ipp64f* pValue);
```

Supported values for `mod`:

8u\_C1R      16u\_C1R      16s\_C1R

**Case 2: Operation on one-channel floating-point data**

```
IppStatus ippiNormRel_L2_32f_C1R(const Ipp32f* pSrc1, int src1Step, const Ipp32f* pSrc2,
int src2Step, IppiSize roiSize, Ipp64f* pValue, IppHintAlgorithm hint);
```

**Case 3: Masked operation on one-channel data**

```
IppStatus ippiNormRel_L2_<mod>(const Ipp<datatype>* pSrc1, int src1Step, const Ipp32f*
pSrc2, int src2Step, const Ipp8u* pMask, int maskStep, IppiSize roiSize, Ipp64f* pNorm);
```

Supported values for mod:

8u_C1MR	8s_C1MR	16u_C1MR	32f_C1MR
---------	---------	----------	----------

**Case 4: Operation on multi-channel integer data**

```
IppStatus ippiNormRel_L2_<mod>(const Ipp<datatype>* pSrc1, int src1Step, const
Ipp<datatype>* pSrc2, int src2Step, IppiSize roiSize, Ipp64f value[3]);
```

Supported values for mod:

8u_C3R	16u_C3R	16s_C3R
8u_AC4R	16u_AC4R	16s_AC4R

```
IppStatus ippiNormRel_L2_<mod>(const Ipp<datatype>* pSrc1, int src1Step, const
Ipp<datatype>* pSrc2, int src2Step, IppiSize roiSize, Ipp64f value[4]);
```

Supported values for mod:

8u_C4R	16u_C4R	16s_C4R
--------	---------	---------

**Case 5: Operation on multi-channel floating-point data**

```
IppStatus ippiNormRel_L2_<mod>(const Ipp32f* pSrc1, int src1Step, const Ipp32f* pSrc2,
int src2Step, IppiSize roiSize, Ipp64f value[3], IppHintAlgorithm hint);
```

Supported values for mod:

32f_C3R
32f_AC4R

```
IppStatus ippiNormRel_L2_32f_C4R(const Ipp32f* pSrc1, int src1Step, const Ipp32f* pSrc2,
int src2Step, IppiSize roiSize, Ipp64f value[4], IppHintAlgorithm hint);
```

**Case 6: Masked operation on multi-channel data**

```
IppStatus ippiNormRel_L2_<mod>(const Ipp<datatype>* pSrc1, int src1Step, const Ipp32f*
pSrc2, int src2Step, const Ipp8u* pMask, int maskStep, IppiSize roiSize, int coi, Ipp64f*
pNorm);
```

Supported values for mod:

8u\_C3CMR      8s\_C3CMR      16u\_C3CMR      32f\_C3CMR

**Parameters**

<i>pSrc1, pSrc2</i>	Pointers to the source images ROI.
<i>src1Step, src2Step</i>	Distance in bytes between starts of consecutive lines in the source images.
<i>pMask</i>	Pointer to the mask image.
<i>maskStep</i>	Distance in bytes between starts of consecutive lines in the mask image.
<i>roiSize</i>	Size of the source ROI in pixels.
<i>coi</i>	Channel of interest (for color images only); can be 1, 2, or 3.
<i>pValue</i>	Pointer to the computed relative error value.
<i>value</i>	An array containing the computed relative error values for separate channels in case of multi-channel data.
<i>pNorm</i>	Pointer to the computed relative norm value in the mask mode.
<i>hint</i>	Option to select the algorithmic implementation of the function.

**Description**

The flavors of the function `ippiNormRel_L2` that perform masked operation are declared in the `ippcv.h` file. All other function flavors are declared in the `ippi.h` file. The function operates with ROI (see [Regions of Interest in Intel IPP](#)). It computes the L2- norm of differences between pixel values of two source buffers *pSrc1* and *pSrc2*. This norm is defined as the square root of the sum of squared pixel values in an image. The output relative error *pValue* (*pNorm* in the mask mode) is then formed by dividing the computed norm of differences by the L2- norm of the second source image buffer *pSrc2*. Computation algorithm is specified by the *hint* argument (see [Table “Hint Arguments for Image Moment Functions”](#)). In the mask mode, the computation is done over pixels selected by nonzero mask values.

For non-masked operations on multi-channel images (*Cases 4, 5*), the relative norm is computed separately for each pair of corresponding channels and stored in the array *value*.

In the mask multi-channel mode (*Case 6*), the relative norm is computed for a single channel of interest specified by *coi*.

**Return Values**

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error when any of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.

<code>ippStsStepErr</code>	Indicates an error condition in mask mode, if <code>src1Step</code> , <code>src2Step</code> , or <code>maskStep</code> is less than <code>roiSize.width * &lt;pixelSize&gt;</code> .
<code>ippStsNotEvenStepErr</code>	Indicates an error condition in mask mode if steps for floating-point images cannot be divided by 4.
<code>ippStsCOIErr</code>	Indicates an error when <code>coi</code> is not 1, 2, or 3.
<code>ippStsDivByZero</code>	Indicates a warning when the L2 norm of <code>pSrc2</code> has a zero value.

### Example

The code example below shows how to use the function `ippiNormRel_L2_8u_C1R`.

```
void func_normrel_l1()
{
    Ipp8u pSrc1[8*4];
    Ipp8u pSrc2[8*4];
    Ipp64f Value;
    int src1Step = 8;
    int src2Step = 8;
    IppiSize roi = {8,4};
    IppiSize roiSize = {5,4};

    ippiSet_8u_C1R(1, pSrc1, src1Step, roi);
    ippiSet_8u_C1R(10, pSrc2, src2Step, roi);
    ippiNormRel_L2_8u_C1R( pSrc1, src1Step, pSrc2, src2Step, roiSize, &Value);

}
Result -> 0.9
```

## Image Quality Index

Intel IPP functions described in this section compute the universal image quality index [Wang02] that may be used as image and video quality distortion measure. It is mathematically defined by modeling the image distortion relative to the reference image as a combination of three factors: loss of correlation, luminance distortion, and contrast distortion.

If two images  $f$  and  $g$  are considered as a matrices with  $M$  column and  $N$  rows containing pixel values  $f[i, j]$ ,  $g[i, j]$ , respectively ( $0 \leq i < M$ ,  $0 \leq j < N$ ), the universal image quality index  $Q$  may be calculated as a product of three components:

$$Q = \frac{\sigma_{fg}}{\sigma_f \sigma_g} \cdot \frac{2\bar{f}\bar{g}}{(\bar{f})^2 + (\bar{g})^2} \cdot \frac{2\sigma_f \sigma_g}{\sigma_f^2 + \sigma_g^2}$$

where

$$\bar{f} = \frac{1}{MN} \sum_{i=0}^{M-1} \sum_{j=0}^{N-1} f[i,j] \quad \bar{g} = \frac{1}{MN} \sum_{i=0}^{M-1} \sum_{j=0}^{N-1} g[i,j]$$

$$\sigma_{fg} = \frac{1}{M+N-1} \sum_{i=0}^{M-1} \sum_{j=0}^{N-1} (f[i,j] - \bar{f})(g[i,j] - \bar{g})$$

$$\sigma_f^2 = \frac{1}{M+N-1} \sum_{i=0}^{M-1} \sum_{j=0}^{N-1} (f[i,j] - \bar{f})^2 \quad \sigma_g^2 = \frac{1}{M+N-1} \sum_{i=0}^{M-1} \sum_{j=0}^{N-1} (g[i,j] - \bar{g})^2$$

The first component is the correlation coefficient, which measures the degree of linear correlation between images  $f$  and  $g$ . It varies in the range  $[-1, 1]$ . The best value 1 is obtained when  $f$  and  $g$  are linearly related, which means that  $g[i, j] = af[i, j] + b$  for all possible values of  $i$  and  $j$ . The second component, with a value range of  $[0, 1]$ , measures how close the mean luminance is between images. Since  $\sigma_f$  and  $\sigma_g$  can be considered as estimates of the contrast of  $f$  and  $g$ , the third component measures how similar the contrasts of the images are. The value range for this component is also  $[0, 1]$ .

The range of values for the index  $Q$  is  $[-1, 1]$ . The best value 1 is achieved if and only if the images are identical.

## QualityIndex

*Computes the universal image quality index.*

### Syntax

#### Case 1: Operation on one-channel data

```
IppStatus ippiQualityIndex_mod(const Ipp<srcDatatype>* pSrc1, int src1Step, const
Ipp<srcDatatype>* pSrc2, int src2Step, IppiSize roiSize, Ipp<dstDatatype>* pQualityIndex);
```

Supported values for mod:

8u32f\_C1R      16u32f\_C1R      32f\_C1R

**Case 2: Operation on multi-channel data**

```
IppStatus ippiQualityIndex_<mod>(const Ipp<srcDatatype>* pSrc1, int src1Step, const
Ipp<srcDatatype>* pSrc2, int src2Step, IppiSize roiSize, Ipp<dstDatatype> qualityIndex[3]);
```

Supported values for `mod`:

8u32f_C3R	16u32f_C3R	32f_C3R
8u32f_AC4R	16u32f_AC4R	32f_AC4R

**Parameters**

<i>pSrc1, pSrc2</i>	Pointers to the source images ROI.
<i>src1Step, src2Step</i>	Distance in bytes between starts of consecutive lines in the source images.
<i>roiSize</i>	Size of the source ROI in pixels.
<i>pQualityIndex</i>	Pointer to the computed quality index value.
<i>qualityIndex</i>	An array containing the computed quality index values for separate channels in case of multi-channel data.

**Description**

The function `ippiQualityIndex` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)). This function computes the universal image quality index for two images *pSrc1* and *pSrc2* according to the formula in the introduction section above. The computed value of the index for one-channel image is stored in *pQualityIndex*. For multi-channel images, the quality index is computed separately for each channel and stored in the array *qualityIndex*.

**Return Values**

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error when any of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <i>src1Step</i> or <i>src2Step</i> has a zero or negative value.
<code>ippStsQualityIndexErr</code>	Indicates an error condition if pixel values of one of the images are constant.
<code>ippStsMemAllocErr</code>	Indicates an error condition if memory allocation fails.

## Example

The code example below shows how to use the function `ippiQualityIndex_32f_C1R`.

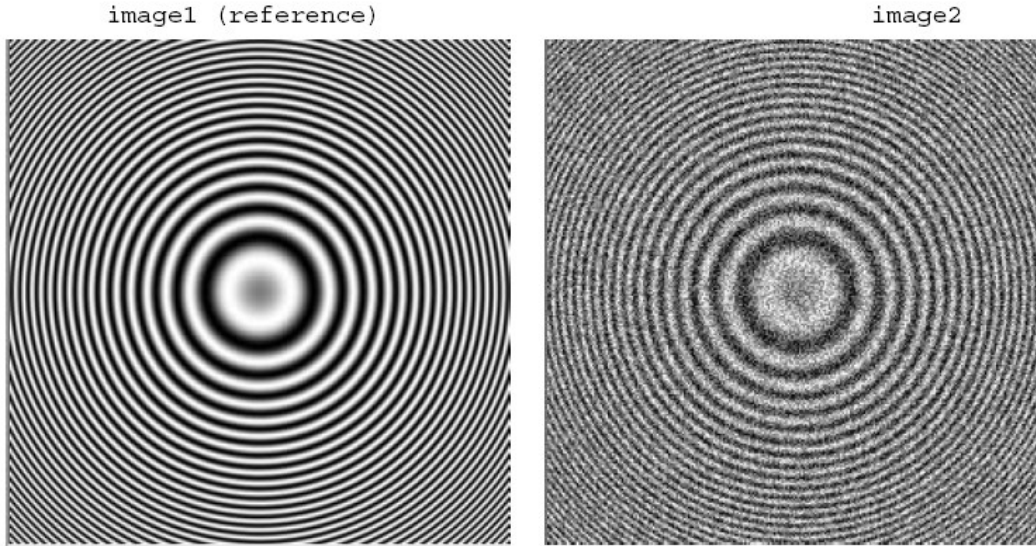
```
void func_qualityindex()
{
    Ipp32f pSrc1[256*256];
    Ipp32f pSrc2[256*256];
    int src1Step = 256*sizeof(Ipp32f);
    int src2Step = 256*sizeof(Ipp32f);
    Ipp32f pQIndex;
    IppiSize roiSize = {256,256};
    unsigned int pSeed = 3;
    // creating reference image (image1) and distorted image (image2)
    ippiImageJaehne_32f_C1R(pSrc1, src1Step, roiSize); // image1
    ippiImageJaehne_32f_C1R(pSrc2, src2Step, roiSize);

    ippiAddRandUniform_Direct_32f_C1R(pSrc2, src2Step, roiSize, 0, 1, &pSeed);
    //image2 = image1 + RandUniform

    ippiQualityIndex_32f_C1R(pSrc1, src1Step, pSrc2, src2Step, roiSize,

    &pQIndex);
}
```

Result:



Universal image quality index of the distorted image2 -> pQIndex = 0.6

---

## Image Proximity Measures

The functions described in this section compute the proximity (similarity) measure between an image and a template (another image). These functions may be used as feature detection functions, as well as the components of more sophisticated techniques.

There are several ways to compute the measure of similarity between two images. One way is to compute the Euclidean distance, or sum of the squared distances (SSD), of an image and a template. The smaller is the value of SSD at a particular pixel, the more similarity exists between the template and the image in the neighborhood of that pixel.

The squared Euclidean distance  $S_{tx}(r, c)$  between a template and an image for the pixel in row  $r$  and column  $c$  is given by the equation:

$$S_{tx}(r, c) = \sum_{j=0}^{tplRows-1} \sum_{i=0}^{tplCols-1} \left[ t(j, i) - x\left(r + j - \frac{tplRows}{2}, c + i - \frac{tplCols}{2}\right) \right]^2$$

where  $x(r, c)$  is the image pixel value in row  $r$  and column  $c$ , and  $t(j, i)$  is the template pixel value in row  $j$  and column  $i$ ; template size is  $tplCols$  by  $tplRows$  and its center is positioned at  $(r, c)$ .

The other similarity measure is the cross-correlation function: the higher is the cross-correlation at a particular pixel, the more similarity exists between the template and the image in the neighborhood of that pixel.

The cross-correlation  $R_{tx}(r, c)$  between a template and an image at the pixel in row  $r$  and column  $c$  is computed by the equation :



$$R_{tx}(r, c) = \sum_{j=0}^{tplRows-1} \sum_{i=0}^{tplCols-1} t(j, i) \cdot x\left(r+j-\frac{tplRows}{2}, c+i-\frac{tplCols}{2}\right)$$

The cross-correlation function is dependent on the brightness variation across the image. To avoid this dependence, the correlation coefficient function is used instead. It is defined as:

$$G_{tx}(r, c) = \sum_{j=0}^{tplRows-1} \sum_{i=0}^{tplCols-1} [t(j, i) - \bar{t}] \cdot \left[ x\left(r+j-\frac{tplRows}{2}, c+i-\frac{tplCols}{2}\right) - \bar{x}(r, c) \right]$$

where  $\bar{t}$  with the overline is the mean of the template, and  $\bar{x}$  with the overline is the mean of the image in the region just under the template.

All Intel IPP proximity functions compute *normalized* values of SSD, cross-correlation and correlation coefficient that are defined as follows:

normalized SSD:  $\sigma_{tx}(r, c)$

$$\sigma_{tx}(r, c) = \frac{S_{tx}(r, c)}{\sqrt{R_{xx}(r, c) R_{tt}\left(\frac{tplRows}{2}, \frac{tplCols}{2}\right)}}$$

normalized cross-correlation  $\rho_{tx}(r, c)$ :

$$\rho_{tx}(r, c) = \frac{R_{tx}(r, c)}{\sqrt{R_{xx}(r, c) R_{tt}\left(\frac{tplRows}{2}, \frac{tplCols}{2}\right)}}$$

Here  $R_{xx}$  and  $R_{tt}$  denote the autocorrelation of the image and the template, respectively:

$$R_{xx}(r, c) = \sum_{j=r-\frac{tplRows-1}{2}}^{r+\frac{tplRows-1}{2}} \sum_{i=c-\frac{tplCols-1}{2}}^{c+\frac{tplCols-1}{2}} x_{j, i} x_{j, i}$$

$$R_{tt}\left(\frac{tplRows}{2}, \frac{tplCols}{2}\right) = \sum_{j=0}^{tplRows-1} \sum_{i=0}^{tplCols-1} t_{j, i} t_{j, i}$$

Normalized correlation coefficient  $\gamma_{tx}(r, c)$ :

$$\gamma_{tx}(r, c) = \frac{G_{tx}(r, c)}{\sqrt{G_{xx}(r, c)G_{tt}\left(\frac{tplRows}{2}, \frac{tplCols}{2}\right)}}$$

Here  $G_{xx}$  and  $G_{tt}$  denote the autocorrelations of the image and the template without constant brightness component, respectively:

$$G_{xx}(r, c) = \sum_{j=r-\frac{tplRows-1}{2}}^{r+\frac{tplRows-1}{2}} \sum_{i=c-\frac{tplCols-1}{2}}^{c+\frac{tplCols-1}{2}} [x_{j,i} - \bar{x}(r, c)]^2$$

$$G_{tt}\left(\frac{tplRows}{2}, \frac{tplCols}{2}\right) = \sum_{j=0}^{tplRows-1} \sum_{i=0}^{tplCols-1} (t_{j,i} - \bar{t})^2$$

## SqrDistanceFull\_Norm

*Computes normalized Euclidean distance between an image and a template.*

---

### Syntax

#### Case 1: Operation with integer output

```
IppStatus ippiSqrDistanceFull_Norm_<mod>(const Ipp8u* pSrc, int srcStep, IppiSize
srcRoiSize, const Ipp8u* pTpl, int tplStep, IppiSize tplRoiSize, Ipp8u* pDst, int dstStep,
int scaleFactor);
```

Supported values for mod:

8u\_C1RSfs

8u\_C3RSfs

8u\_C4RSfs

8u\_AC4RSfs

**Case 2: Operation on data with floating-point output**

```
IppStatus ippiSqrDistanceFull_Norm<mod>(const Ipp<srcDatatype>* pSrc, int srcStep, IppiSize
srcRoiSize, const Ipp<srcDatatype>* pTpl, int tplStep, IppiSize tplRoiSize, Ipp32f* pDst,
int dstStep);
```

Supported values for mod:

32f_C1R	8u32f_C1R	8s32f_C1R	16u32f_C1R
32f_C3R	8u32f_C3R	8s32f_C3R	16u32f_C3R
32f_C4R	8u32f_C4R	8s32f_C4R	16u32f_C4R
32f_AC4R	8u32f_AC4R	8s32f_AC4R	16u32f_AC4R

**Parameters**

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>srcRoiSize</i>	Size of the source ROI in pixels.
<i>pTpl</i>	Pointer to the template image buffer.
<i>tplStep</i>	Distance in bytes between starts of consecutive lines in the template image.
<i>tplRoiSize</i>	Size of the template ROI in pixels.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>scaleFactor</i>	Scale factor (see <a href="#">Integer Result Scaling</a> ).

**Description**

The function `ippiSqrDistanceFull_Norm` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)). This function computes and returns the sum of squared distances (SSD) between the source and template images, that is, this function computes the normalized SSD value  $\sigma_{tx}(r, c)$  for each pixel in source and template buffers and stores the computed value in the corresponding pixel of the output image. The size of the resulting matrix with normalized SSD coefficients is

$$(W_S + W_t - 1) * (H_S + H_t - 1),$$

where  $W_S$ ,  $H_S$  denote the width and height of the source image, and  $W_t$ ,  $H_t$  denote the width and height of the template.

The template anchor for matching the image pixel is always at the geometric center of the template. (See the formula for  $\sigma_{tx}(r, c)$  in section [Image Proximity Measures](#)). There are no requirements for data outside the ROI. The function performs as if the source image ROI is zero padded.



**NOTE.** For multi-channel images, each channel is processed separately.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error when any of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition when <code>srcRoiSize</code> or <code>tplRoiSize</code> has a field with zero or negative value, or when <code>srcRoiSize</code> has a field with value smaller than value of the corresponding field of <code>tplRoiSize</code> .
<code>ippStsStepErr</code>	Indicates an error condition if at least one of <code>srcStep</code> , <code>tplStep</code> , or <code>dstStep</code> has a zero or negative value.
<code>ippStsMemAllocErr</code>	Indicates an error condition if memory allocation fails.

## Example

The code example below shows how to use the function `ippiSqrDistanceFull_Norm_32f_C1R`.

```
void finc_sqr_distance()
{
    Ipp32f pSrc[5*4];
    Ipp32f pTpl[5*4];
    Ipp32f pDst[9*7];
    IppiSize srcRoiSize = {5,4};
    IppiSize tplRoiSize = {5,4};
    int srcStep = 5*sizeof(Ipp32f);
    int tplStep = 5*sizeof(Ipp32f);
    int dstStep = 9*sizeof(Ipp32f);

    ippiSet_32f_C1R(2.0, pSrc, srcStep, srcRoiSize);
    ippiSet_32f_C1R(1.0, pTpl, tplStep, tplRoiSize);
    ippiSqrDistanceFull_Norm_32f_C1R( pSrc, srcStep, srcRoiSize, pTpl, tplStep,

        tplRoiSize, pDst, dstStep);
}

Result
```

```

1.58 1.12 0.91 0.79 0.71 0.79 0.91 1.12 1.58
1.29 0.91 0.75 0.65 0.58 0.65 0.75 0.91 1.92
1.12 0.79 0.65 0.56 0.50 0.56 0.65 0.79 1.12
1.29 0.91 0.75 0.65 0.58 0.65 0.75 0.91 1.29
1.58 1.12 0.91 0.79 0.71 0.79 0.91 1.12 1.58
2.24 1.58 1.29 1.12 1.00 1.12 1.29 1.58 2.24

```

## SqrDistanceSame\_Norm

*Computes normalized Euclidean distance between an image and a template.*

---

### Syntax

#### Case 1: Operation with integer output

```

IppStatus ippiSqrDistanceSame_Norm_<mod>(const Ipp8u* pSrc, int srcStep, IppiSize
srcRoiSize, const Ipp8u* pTpl, int tplStep, IppiSize tplRoiSize, Ipp8u* pDst, int dstStep,
int scaleFactor);

```

Supported values for mod:

8u\_C1RSfs

8u\_C3RSfs

8u\_C4RSfs

8u\_AC4RSfs

#### Case 2: Operation on data with floating-point output

```

IppStatus ippiSqrDistanceSame_Norm_<mod>(const Ipp<srcDatatype>* pSrc, int srcStep, IppiSize
srcRoiSize, const Ipp<srcDatatype>* pTpl, int tplStep, IppiSize tplRoiSize, Ipp32f* pDst,
int dstStep);

```

Supported values for mod:

32f\_C1R      8u32f\_C1R      8s32f\_C1R      16u32f\_C1R

32f\_C3R      8u32f\_C3R      8s32f\_C3R      16u32f\_C3R

32f\_C4R      8u32f\_C4R      8s32f\_C4R      16u32f\_C4R

32f\_AC4R      8u32f\_AC4R      8s32f\_AC4R      16u32f\_AC4R

## Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>srcRoiSize</i>	Size of the source ROI in pixels.
<i>pTpl</i>	Pointer to the template image buffer.
<i>tplStep</i>	Distance in bytes between starts of consecutive lines in the template image.
<i>tplRoiSize</i>	Size of the template ROI in pixels.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>scaleFactor</i>	Scale factor (see <a href="#">Integer Result Scaling</a> ).

## Description

The function `ippiSqrDistanceSame_Norm` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)). This function computes the normalized SSD values  $\sigma_{tx}(r, c)$  for the pixels that belong only to the source image. The size of the resulting matrix with normalized SSD coefficients is equal to the size of the source image:  $W_S * H_S$ , where  $W_S$ ,  $H_S$  denote the width and height of the source image, respectively.

The template anchor for matching the image pixel is always at the geometric center of the template. (See the formula for  $\sigma_{tx}(r, c)$  in section [Image Proximity Measures](#)). There are no requirements for data outside the ROI. The function performs as if the source image ROI is zero padded.




---

**NOTE.** For multi-channel images each channel is processed separately.

---

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error when any of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition when <i>srcRoiSize</i> or <i>tplRoiSize</i> has a field with zero or negative value, or when <i>srcRoiSize</i> has a field with value smaller than value of the corresponding field of <i>tplRoiSize</i> .
<code>ippStsStepErr</code>	Indicates an error condition if at least one of <i>srcStep</i> , <i>tplStep</i> , or <i>dstStep</i> has a zero or negative value.
<code>ippStsMemAllocErr</code>	Indicates an error condition if memory allocation fails.

## SqrDistanceValid\_Norm

*Computes normalized Euclidean distance between an image and a template.*

---

### Syntax

#### Case 1: Operation with integer output

```
IppStatus ippiSqrDistanceValid_Norm_<mod>(const Ipp8u* pSrc, int srcStep, IppiSize srcRoiSize, const Ipp8u* pTpl, int tplStep, IppiSize tplRoiSize, Ipp8u* pDst, int dstStep, int scaleFactor);
```

Supported values for mod:

```
8u_C1RSfs
8u_C3RSfs
8u_C4RSfs
8u_AC4RSfs
```

#### Case 2: Operation on data with floating-point output

```
IppStatus ippiSqrDistanceValid_Norm_<mod>(const Ipp<srcDatatype>* pSrc, int srcStep, IppiSize srcRoiSize, const Ipp<srcDatatype>* pTpl, int tplStep, IppiSize tplRoiSize, Ipp32f* pDst, int dstStep);
```

Supported values for mod:

32f_C1R	8u32f_C1R	8s32f_C1R	16u32f_C1R
32f_C3R	8u32f_C3R	8s32f_C3R	16u32f_C3R
32f_C4R	8u32f_C4R	8s32f_C4R	16u32f_C4R
32f_AC4R	8u32f_AC4R	8s32f_AC4R	16u32f_AC4R

### Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>srcRoiSize</i>	Size of the source ROI in pixels.
<i>pTpl</i>	Pointer to the template image buffer.
<i>tplStep</i>	Distance in bytes between starts of consecutive lines in the template image.
<i>tplRoiSize</i>	Size of the template ROI in pixels.
<i>pDst</i>	Pointer to the destination image ROI.

*dstStep* Distance in bytes between starts of consecutive lines in the destination image.

*scaleFactor* Scale factor (see [Integer Result Scaling](#)).

## Description

The function `ippiSqrDistanceValid_Norm` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)). This function computes the normalized SSD values  $\sigma_{tx}(r, c)$  for the pixels of the source image without zero-padded edges. The size of the resulting matrix with normalized SSD values is

$$(W_S - W_t + 1) * (H_S - H_t + 1),$$

where  $W_S, H_S$  denote the width and height of the source image, and  $W_t, H_t$  denote the width and height of the template. The template anchor for matching the image pixel is always at the geometric center of the template. (See the formula for  $\sigma_{tx}(r, c)$  in section [Image Proximity Measures](#)).



**NOTE.** For multi-channel images each channel is processed separately.

---

## Return Values

`ippStsNoErr` Indicates no error. Any other value indicates an error or a warning.

`ippStsNullPtrErr` Indicates an error when any of the specified pointers is `NULL`.

`ippStsSizeErr` Indicates an error condition when *srcRoiSize* or *tplRoiSize* has a field with zero or negative value, or when *srcRoiSize* has a field with value smaller than value of the corresponding field of *tplRoiSize*.

`ippStsStepErr` Indicates an error condition if at least one of *srcStep*, *tplStep*, or *dstStep* has a zero or negative value.

`ippStsMemAllocErr` Indicates an error condition if memory allocation fails.

## CrossCorrFull\_Norm

*Computes normalized cross-correlation between an image and a template.*

---

## Syntax

### Case 1: Operation with integer output

```
IppStatus ippiCrossCorrFull_Norm_<mod>(const Ipp8u* pSrc, int srcStep, IppiSize srcRoiSize,
const Ipp8u* pTpl, int tplStep, IppiSize tplRoiSize, Ipp8u* pDst, int dstStep, int
scaleFactor);
```

Supported values for `mod`:

`8u_C1RSfs`

`8u_C3RSfs`



8u\_C4RSfs

8u\_AC4RSfs

## Case 2: Operation on data with floating-point output

```
IppStatus ippiCrossCorrFull_Norm_<mod>(const Ipp<srcDatatype>* pSrc, int srcStep, IppiSize
srcRoiSize, const Ipp<srcDatatype>* pTpl, int tplStep, IppiSize tplRoiSize, Ipp32f* pDst,
int dstStep);
```

Supported values for mod:

32f_C1R	8u32f_C1R	8s32f_C1R	16u32f_C1R
32f_C3R	8u32f_C3R	8s32f_C3R	16u32f_C3R
32f_C4R	8u32f_C4R	8s32f_C4R	16u32f_C4R
32f_AC4R	8u32f_AC4R	8s32f_AC4R	16u32f_AC4R

## Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>srcRoiSize</i>	Size of the source ROI in pixels.
<i>pTpl</i>	Pointer to the template image buffer.
<i>tplStep</i>	Distance in bytes between starts of consecutive lines in the template image.
<i>tplRoiSize</i>	Size of the template ROI in pixels.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>scaleFactor</i>	Scale factor (see <a href="#">Integer Result Scaling</a> ).

## Description

The function `ippiCrossCorrFull_Norm` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)). This function computes and returns the cross correlation between the source and template images, that is, this function computes the normalized cross-correlation value  $\rho_{tx}(x, c)$  for each pixel in source and template buffers and stores the computed value in the corresponding pixel of the output image. The size of the resulting matrix with normalized cross-correlation coefficients is

$$(W_S + W_t - 1) * (H_S + H_t - 1),$$

where  $W_S$ ,  $H_S$  denote the width and height of the source image, and  $W_t$ ,  $H_t$  denote the width and height of the template.

The template anchor for matching the image pixel is always at the geometric center of the template. (See the formula for  $p_{tx}(r, c)$  in section [Image Proximity Measures](#)). There are no requirements for data outside the ROI. The function performs as if the source image ROI is zero padded.



**NOTE.** For multi-channel images, each channel is processed separately.

## Application Notes

Intel IPP does not provide *absolute* accuracy for rather sophisticated functions. As these functions are intended for 32f data type, all calculations are performed in floats. To avoid divisions by zeroes as well as overflows and underflows, substitutions are implemented for very small denominator values in the course of the normalization phase. Output for these functions is always correct when the source image pixels are normalized - that is, they must be in the range 0.0f - 1.0f. The following example demonstrates how the normalization can be done to have a correct result:

```
ippiMinMax_32f_C1R( pSrc, srcStep, tmpRoi, &min, &max );
normC = 1.0f/max;
ippiMulC_32f_C1R( normC, pSrc .. );
```

The only requirement is that both the nominator and the denominator (see the formulas above) must exceed `#define IPP_EPS23 ( 1.19209289e-07f )`, otherwise substitution is used.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error when any of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition when <code>srcRoiSize</code> or <code>tplRoiSize</code> has a field with zero or negative value, or when <code>srcRoiSize</code> has a field with value smaller than value of the corresponding field of <code>tplRoiSize</code> .
<code>ippStsStepErr</code>	Indicates an error condition if at least one of <code>srcStep</code> , <code>tplStep</code> , or <code>dstStep</code> has a zero or negative value.
<code>ippStsMemAllocErr</code>	Indicates an error condition if memory allocation fails.

## CrossCorrSame\_Norm

*Computes normalized cross-correlation between an image and a template.*

### Syntax

#### Case 1: Operation with integer output

```
IppStatus ippiCrossCorrSame_Norm_<mod>(const Ipp8u* pSrc, int srcStep, IppiSize srcRoiSize,
const Ipp8u* pTpl, int tplStep, IppiSize tplRoiSize, Ipp8u* pDst, int dstStep, int
scaleFactor);
```

Supported values for mod:

```
8u_C1RSfs
8u_C3RSfs
8u_C4RSfs
8u_AC4RSfs
```

#### Case 2: Operation on data with floating-point output

```
IppStatus ippiCrossCorrSame_Norm_<mod>(const Ipp<srcDatatype>*pSrc, int srcStep, IppiSize
srcRoiSize, const Ipp<srcDatatype>* pTpl, int tplStep, IppiSize tplRoiSize, Ipp32f* pDst,
int dstStep);
```

Supported values for mod:

```
32f_C1R      8u32f_C1R      8s32f_C1R      16u32f_C1R
32f_C3R      8u32f_C3R      8s32f_C3R      16u32f_C3R
32f_C4R      8u32f_C4R      8s32f_C4R      16u32f_C4R
32f_AC4R     8u32f_AC4R     8s32f_AC4R     16u32f_AC4R
```

### Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>srcRoiSize</i>	Size of the source ROI in pixels.
<i>pTpl</i>	Pointer to the template image buffer.
<i>tplStep</i>	Distance in bytes between starts of consecutive lines in the template image.
<i>tplRoiSize</i>	Size of the template ROI in pixels.
<i>pDst</i>	Pointer to the destination image ROI.

*dstStep* Distance in bytes between starts of consecutive lines in the destination image.

*scaleFactor* Scale factor (see [Integer Result Scaling](#)).

## Description

The function `ippiCrossCorrSame_Norm` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)). This function computes the normalized cross-correlation values  $\rho_{tX}(r, c)$  for the pixels that belong only to the source image. The size of the resulting matrix with normalized cross-correlation values is equal to the size of the source image:  $W_S * H_S$ , where  $W_S$ ,  $H_S$  denote the width and height of the source image, respectively.

The template anchor for matching the image pixel is always at the geometric center of the template. (See the formula for  $\rho_{tX}(r, c)$  in section [Image Proximity Measures](#)). There are no requirements for data outside the ROI. The function performs as if the source image ROI is zero padded.



**NOTE.** For multi-channel images each channel is processed separately.

## Return Values

`ippStsNoErr` Indicates no error. Any other value indicates an error or a warning.

`ippStsNullPtrErr` Indicates an error when any of the specified pointers is `NULL`.

`ippStsSizeErr` Indicates an error condition when `srcRoiSize` or `tplRoiSize` has a field with zero or negative value, or when `srcRoiSize` has a field with value smaller than value of the corresponding field of `tplRoiSize`.

`ippStsStepErr` Indicates an error condition if at least one of `srcStep`, `tplStep`, or `dstStep` has a zero or negative value.

`ippStsMemAllocErr` Indicates an error condition if memory allocation fails.

## Example

The code example below shows how to use the function `ippiCrossCorrSame_Norm_32f_C1R`.

```
void func_CrossCorrSame_Norm()  
{  
    Ipp32f pSrc[5*4] = { 1.0, 2.0, 1.5, 4.1, 3.6,  
                        0.2, 3.2, 2.5, 1.5, 10.0,  
                        5.0, 6.8, 0.5, 4.1, 1.1,  
                        7.1, 4.2, 2.2, 8.7, 10.0};  
    Ipp32f pTpl[3*3] = {2.1, 3.5, 7.7,  
                       0.4, 2.3, 5.5,  
                       1.4, 2.8, 3.1};  
    Ipp32f pDst[5*4];  
  
    IppiSize srcRoiSize = {5,4};  
    IppiSize tplRoiSize = {3,3};  
    int srcStep = 5*sizeof(Ipp32f);  
    int tplStep = 3*sizeof(Ipp32f);  
    int dstStep = 5*sizeof(Ipp32f);  
  
    ippiCrossCorrSame_Norm_32f_C1R(pSrc, srcStep, srcRoiSize, pTpl, tplStep, tplRoiSize, pDst, dstStep);  
}
```

Result: pDst ->

```
0.53 0.54 0.58 0.50 0.30  
0.68 0.62 0.68 0.83 0.38  
0.77 0.55 0.60 0.81 0.42  
0.81 0.46 0.70 0.62 0.24
```

## CrossCorrValid\_Norm

*Computes normalized cross-correlation between an image and a template.*

---

### Syntax

#### Case 1: Operation with integer output

```
IppStatus ippiCrossCorrValid_Norm_<mod>(const Ipp8u* pSrc, int srcStep, IppiSize srcRoiSize,
const Ipp8u* pTpl, int tplStep, IppiSize tplRoiSize, Ipp8u* pDst, int dstStep, int
scaleFactor);
```

Supported values for mod:

8u\_C1RSfs  
8u\_C3RSfs  
8u\_C4RSfs  
8u\_AC4RSfs

#### Case 2: Operation on data with floating-point output

```
IppStatus ippiCrossCorrValid_Norm_<mod>(const Ipp<srcDatatype>* pSrc, int srcStep, IppiSize
srcRoiSize, const Ipp<srcDatatype>* pTpl, int tplStep, IppiSize tplRoiSize, Ipp32f* pDst,
int dstStep);
```

Supported values for mod:

32f_C1R	8u32f_C1R	8s32f_C1R	16u32f_C1R
32f_C3R	8u32f_C3R	8s32f_C3R	16u32f_C3R
32f_C4R	8u32f_C4R	8s32f_C4R	16u32f_C4R
32f_AC4R	8u32f_AC4R	8s32f_AC4R	16u32f_AC4R

### Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>srcRoiSize</i>	Size of the source ROI in pixels.
<i>pTpl</i>	Pointer to the template image buffer.
<i>tplStep</i>	Distance in bytes between starts of consecutive lines in the template image.
<i>tplRoiSize</i>	Size of the template ROI in pixels.
<i>pDst</i>	Pointer to the destination image ROI.

<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>scaleFactor</i>	Scale factor (see <a href="#">Integer Result Scaling</a> ).

## Description

The function `ippiCrossCorrValid_Norm` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)). This function computes the normalized cross-correlation values  $\rho_{tx}(r, c)$  for the pixels of the source image without zero-padded edges. The size of the resulting matrix with normalized cross-correlation values is

$$(W_S - W_t + 1) * (H_S - H_t + 1),$$

where  $W_S, H_S$  denote the width and height of the source image, and  $W_t, H_t$  denote the width and height of the template. The template anchor for matching the image pixel is always at the geometric center of the template. (See the formula for  $\rho_{tx}(r, c)$  in section [Image Proximity Measures](#)).



## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error when any of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition when <code>srcRoiSize</code> or <code>tplRoiSize</code> has a field with zero or negative value, or when <code>srcRoiSize</code> has a field with value smaller than value of the corresponding field of <code>tplRoiSize</code> .
<code>ippStsStepErr</code>	Indicates an error condition if at least one of <code>srcStep</code> , <code>tplStep</code> , or <code>dstStep</code> has a zero or negative value.
<code>ippStsMemAllocErr</code>	Indicates an error condition if memory allocation fails.

## CrossCorrValid

*Computes cross-correlation between an image and a template.*

## Syntax

```
ippStatus ippiCrossCorrValid_<mod>(const Ipp<srcDatatype>* pSrc, int srcStep, IppiSize srcRoiSize, const Ipp<srcDatatype>* pTpl, int tplStep, IppiSize tplRoiSize, Ipp32f* pDst, int dstStep);
```

Supported values for `mod`:

32f\_C1R      8u32f\_C1R      8s32f\_C1R      16u32f\_C1R

## Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>srcRoiSize</i>	Size of the source ROI in pixels.
<i>pTpl</i>	Pointer to the template image buffer.
<i>tplStep</i>	Distance in bytes between starts of consecutive lines in the template image.
<i>tplRoiSize</i>	Size of the template ROI in pixels.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.

## Description

The function `ippiCrossCorrValid` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)). This function computes the cross-correlation values  $R_{tx}(r, c)$  for the pixels of the source image without zero-padded edges. The size of the resulting matrix with cross-correlation values is

$$(W_S - W_t + 1) * (H_S - H_t + 1),$$

where  $W_S, H_S$  denote the width and height of the source image, and  $W_t, H_t$  denote the width and height of the template. The template anchor for matching the image pixel is always at the geometric center of the template. (See the formula for  $R_{tx}(r, c)$  in section [Image Proximity Measures](#)).




---

**NOTE.** For multi-channel images, each channel is processed separately.

---

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error when any of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition when <i>srcRoiSize</i> or <i>tplRoiSize</i> has a field with zero or negative value, or when <i>srcRoiSize</i> has a field with value smaller than value of the corresponding field of <i>tplRoiSize</i> .
<code>ippStsStepErr</code>	Indicates an error condition if at least one of <i>srcStep</i> , <i>tplStep</i> , or <i>dstStep</i> has a zero or negative value.
<code>ippStsMemAllocErr</code>	Indicates an error condition if memory allocation fails.



## CrossCorrFull\_NormLevel

*Computes normalized correlation coefficient between an image and a template.*

---

### Syntax

#### Case 1: Operation with integer output

```
IppStatus ippiCrossCorrFull_NormLevel_<mod>(const Ipp8u* pSrc, int srcStep, IppiSize srcRoiSize, const Ipp8u* pTpl, int tplStep, IppiSize tplRoiSize, Ipp8u* pDst, int dstStep, int scaleFactor);
```

Supported values for mod:

```
8u_C1RSfs
8u_C3RSfs
8u_C4RSfs
8u_AC4RSfs
```

#### Case 2: Operation on data with floating-point output

```
IppStatus ippiCrossCorrFull_NormLevel_<mod>(const Ipp<srcDatatype>* pSrc, int srcStep, IppiSize srcRoiSize, const Ipp<srcDatatype>* pTpl, int tplStep, IppiSize tplRoiSize, Ipp<dstDatatype>* pDst, int dstStep);
```

Supported values for mod:

```
32f_C1R    8u32f_C1R    8s32f_C1R    16u32f_C1R    64f_C1R
32f_C3R    8u32f_C3R    8s32f_C3R    16u32f_C3R
32f_C4R    8u32f_C4R    8s32f_C4R    16u32f_C4R
32f_AC4R   8u32f_AC4R   8s32f_AC4R   16u32f_AC4R
```

### Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>srcRoiSize</i>	Size of the source ROI in pixels.
<i>pTpl</i>	Pointer to the template image buffer.
<i>tplStep</i>	Distance in bytes between starts of consecutive lines in the template image.
<i>tplRoiSize</i>	Size of the template ROI in pixels.
<i>pDst</i>	Pointer to the destination image ROI.

*dstStep* Distance in bytes between starts of consecutive lines in the destination image.

*scaleFactor* Scale factor (see [Integer Result Scaling](#)).

## Description

The function `ippiCrossCorrFull_NormLevel` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)). This function computes and returns the correlation coefficient between the source and template images, that is, this function computes the normalized correlation coefficient  $\gamma_{tx}(r, c)$  for each pixel in source and template buffers and stores the computed value in the corresponding pixel of the output image. The size of the resulting matrix with normalized cross-correlation coefficients is

$$(W_S + W_t - 1) * (H_S + H_t - 1),$$

where  $W_S$ ,  $H_S$  denote the width and height of the source image, and  $W_t$ ,  $H_t$  denote the width and height of the template.

The template anchor for matching the image pixel is always at the geometric center of the template. (See the formula for  $\gamma_{tx}(r, c)$  in section [Image Proximity Measures](#)). There are no requirements for data outside the ROI. The function performs as if the source image ROI is zero padded.

## Return Values

`ippStsNoErr` Indicates no error. Any other value indicates an error or a warning.

`ippStsNullPtrErr` Indicates an error when any of the specified pointers is `NULL`.

`ippStsSizeErr` Indicates an error condition when `srcRoiSize` or `tplRoiSize` has a field with zero or negative value, or when `srcRoiSize` has a field with value smaller than value of the corresponding field of `tplRoiSize`.

`ippStsStepErr` Indicates an error condition if at least one of `srcStep`, `tplStep`, or `dstStep` has a zero or negative value.

`ippStsMemAllocErr` Indicates an error condition if memory allocation fails.

## CrossCorrSame\_NormLevel

*Computes normalized correlation coefficient between an image and a template.*

---

## Syntax

### Case 1: Operation with integer output

```
ippStatus ippiCrossCorrSame_NormLevel_<mod>(const Ipp8u* pSrc, int srcStep, IppiSize srcRoiSize, const Ipp8u* pTpl, int tplStep, IppiSize tplRoiSize, Ipp8u* pDst, int dstStep, int scaleFactor);
```

Supported values for `mod`:

`8u_C1RSfs`

`8u_C3RSfs`

8u\_C4RSfs

8u\_AC4RSfs

## Case 2: Operation on data with floating-point output

```
IppStatus ippiCrossCorrSame_NormLevel_<mod>(const Ipp<srcDatatype>* pSrc, int srcStep,
IppiSize srcRoiSize, const Ipp<srcDatatype>* pTpl, int tplStep, IppiSize tplRoiSize,
Ipp32f* pDst, int dstStep);
```

Supported values for mod:

32f_C1R	8u32f_C1R	8s32f_C1R	16u32f_C1R
32f_C3R	8u32f_C3R	8s32f_C3R	16u32f_C3R
32f_C4R	8u32f_C4R	8s32f_C4R	16u32f_C4R
32f_AC4R	8u32f_AC4R	8s32f_AC4R	16u32f_AC4R

## Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>srcRoiSize</i>	Size of the source ROI in pixels.
<i>pTpl</i>	Pointer to the template image buffer.
<i>tplStep</i>	Distance in bytes between starts of consecutive lines in the template image.
<i>tplRoiSize</i>	Size of the template ROI in pixels.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>scaleFactor</i>	Scale factor (see <a href="#">Integer Result Scaling</a> ).

## Description

The function `ippiCrossCorrSame_NormLevel` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)). This function computes the normalized correlation coefficients  $\gamma_{tx}(r, c)$  for the pixels that belong only to the source image. The size of the resulting matrix with normalized correlation coefficients is equal to the size of the source image:  $\bar{W}_S * H_S$ , where  $\bar{W}_S$ ,  $H_S$  denote the width and height of the source image, respectively.

The template anchor for matching the image pixel is always at the geometric center of the template. (See the formula for  $\gamma_{tx}(r, c)$  in section [Image Proximity Measures](#)). There are no requirements for data outside the ROI. The function performs as if the source image ROI is zero padded.



---

**NOTE.** For multi-channel images, each channel is processed separately.

---

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error when any of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition when <code>srcRoiSize</code> or <code>tplRoiSize</code> has a field with zero or negative value, or when <code>srcRoiSize</code> has a field with value smaller than value of the corresponding field of <code>tplRoiSize</code> .
<code>ippStsStepErr</code>	Indicates an error condition if at least one of <code>srcStep</code> , <code>tplStep</code> , or <code>dstStep</code> has a zero or negative value.
<code>ippStsMemAllocErr</code>	Indicates an error condition if memory allocation fails.

## Example

The code example below shows how to use the function `ippiCrossCorrSame_NormLevel_32f_C1R`.

```
void func_CrossCorrSame_NormLevel()
{
    Ipp32f pSrc[5*4] = { 1.0, 2.0, 1.5, 4.1, 3.6,
                        0.2, 3.2, 2.5, 1.5, 10.0,
                        5.0, 6.8, 0.5, 4.1, 1.1,
                        7.1, 4.2, 2.2, 8.7, 10.0};

    Ipp32f pTpl[3*3] = {2.1, 3.5, 7.7,
                        0.4, 2.3, 5.5,
                        1.4, 2.8, 3.1};

    Ipp32f pDst[5*4];

    IppiSize srcRoiSize = {5, 4};
    IppiSize tplRoiSize = {3, 3};
    int srcStep = 5*sizeof(Ipp32f);
    int tplStep = 3*sizeof(Ipp32f);
    int dstStep = 5*sizeof(Ipp32f);

    ippiCrossCorrSame_NormLevel_32f_C1R(pSrc, srcStep, srcRoiSize,
    pTpl, tplStep, tplRoiSize, pDst, dstStep);
}
```

```
Result:          pDst
      0.16 -0.14 -0.17 -0.10 -0.36
      0.31 -0.08 -0.12  0.55 -0.43
      0.42 -0.49 -0.34  0.46 -0.39
      0.63 -0.38  0.31  0.21 -0.53
```

## CrossCorrValid\_NormLevel

*Computes normalized cross-correlation coefficient between an image and a template.*

---

### Syntax

#### Case 1: Operation with integer output

```
IppStatus ippiCrossCorrValid_NormLevel_<mod>(const Ipp8u* pSrc, int srcStep, IppiSize srcRoiSize, const Ipp8u* pTpl, int tplStep, IppiSize tplRoiSize, Ipp8u* pDst, int dstStep, int scaleFactor);
```

Supported values for mod:

8u\_C1RSfs  
8u\_C3RSfs  
8u\_C4RSfs  
8u\_AC4RSfs

#### Case 2: Operation on data with floating-point output

```
IppStatus ippiCrossCorrValid_NormLevel_<mod>(const Ipp<srcDatatype>* pSrc, int srcStep, IppiSize srcRoiSize, const Ipp<srcDatatype>* pTpl, int tplStep, IppiSize tplRoiSize, Ipp<dstDatatype>* pDst, int dstStep);
```

Supported values for mod:

32f_C1R	8u32f_C1R	8s32f_C1R	16u32f_C1R	64f_C1R
32f_C3R	8u32f_C3R	8s32f_C3R	16u32f_C3R	
32f_C4R	8u32f_C4R	8s32f_C4R	16u32f_C4R	
32f_AC4R	8u32f_AC4R	8s32f_AC4R	16u32f_AC4R	

### Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>srcRoiSize</i>	Size of the source ROI in pixels.
<i>pTpl</i>	Pointer to the template image buffer.
<i>tplStep</i>	Distance in bytes between starts of consecutive lines in the template image.
<i>tplRoiSize</i>	Size of the template ROI in pixels.
<i>pDst</i>	Pointer to the destination image ROI.

<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>scaleFactor</i>	Scale factor (see <a href="#">Integer Result Scaling</a> ).

## Description

The function `ippiCrossCorrValid_NormLevel` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)). This function computes the normalized correlation coefficients  $\gamma_{tx}(r, c)$  for the pixels of the source image without zero-padded edges. The size of the resulting matrix with normalized correlation coefficients is

$$(W_S - W_t + 1) * (H_S - H_t + 1),$$

where  $W_S$ ,  $H_S$  denote the width and height of the source image, and  $W_t$ ,  $H_t$  denote the width and height of the template. The template anchor for matching the image pixel is always at the geometric center of the template. (See the formula for  $\gamma_{tx}(r, c)$  in section [Image Proximity Measures](#)).



**NOTE.** For multi-channel images, each channel is processed separately.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error when any of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition when <code>srcRoiSize</code> or <code>tplRoiSize</code> has a field with zero or negative value, or when <code>srcRoiSize</code> has a field with value smaller than value of the corresponding field of <code>tplRoiSize</code> .
<code>ippStsStepErr</code>	Indicates an error condition if at least one of <code>srcStep</code> , <code>tplStep</code> , or <code>dstStep</code> has a zero or negative value.
<code>ippStsMemAllocErr</code>	Indicates an error condition if memory allocation fails.

## Example

The code example below shows how to use the function `ippiCrossCorrValid_NormLevel_32f_C1R`.

```
void func_ippiCrossCorrValid_NormLevel()
{
    Ipp32f pSrc[5*4] = { 1.0, 2.0, 1.5, 4.1, 3.6,
                        0.2, 3.2, 2.5, 1.5, 10.0,
                        5.0, 6.8, 0.5, 4.1, 1.1,
                        7.1, 4.2, 2.2, 8.7, 10.0};

    Ipp32f pTpl[3*3] = {2.1, 3.5, 7.7,
                       0.4, 2.3, 5.5,
                       1.4, 2.8, 3.1};

    IppiSize srcRoiSize = {5, 4};
    IppiSize tplRoiSize = {3, 3};
    int srcStep = 5*sizeof(Ipp32f);
    int tplStep = 3*sizeof(Ipp32f);
    int dstStep = 3*sizeof(Ipp32f);
```

```
// size of pDst is (srcRoiSize.width - tplRoiSize.width + 1)*(srcRoiSize.height - tplRoiSize.height + 1) =
3*2
Ipp32f pDst[3*2];

ippiCrossCorrValid_NormLevel_32f_C1R(pSrc, srcStep, srcRoiSize, pTpl, tplStep, tplRoiSize, pDst, dstStep);
}
```

Result:

		pDst
-0.08	-0.12	0.55
-0.49	-0.34	0.46



# Image Geometry Transforms

This chapter describes the Intel® IPP image processing functions that perform geometric operations of resizing, rotating, warping and remapping an image.

The table below lists the Intel IPP image geometric transform functions:

## Image Geometric Transform Functions

Function Base Name	Operation
<a href="#">Resize</a>	<b>THIS FUNCTION IS DEPRECATED.</b> Changes the size of an image.
<a href="#">ResizeCenter</a>	<b>THIS FUNCTION IS DEPRECATED.</b> Changes the size of an image and shifts it.
<a href="#">ResizeSqrPixel</a>	Changes the size of an image using different interpolation algorithm.
<a href="#">ResizeGetBufSize</a>	Computes the size of the external buffer for image resizing.
<a href="#">GetResizeRect</a>	Computes the coordinates and size of the resized image.
<a href="#">ResizeSqrPixelGetBufSize</a>	<b>THIS FUNCTION IS DEPRECATED.</b> Computes the size of the external buffer for image resizing.
<a href="#">GetResizeFract</a>	<b>THIS FUNCTION IS DEPRECATED.</b> Recalculates resize factors for a tiled image.
<a href="#">ResizeShift</a>	<b>THIS FUNCTION IS DEPRECATED.</b> Changes the size of an image tile.
<a href="#">SuperSampling</a>	Reduces the image size using the super sampling method.
<a href="#">SuperSamplingGetBufSize</a>	Computes the size of the external buffer for image reducing with the super sampling method.
<a href="#">ResizeYUV422</a>	Changes the size of an 2-channel image in 4:2:2 sampling format.
<a href="#">ResizeYUV420</a>	Changes the size of an 2-plane image in 4:2:0 sampling format.
<a href="#">ResizeYUV420GetBufSize</a>	Calculates the size of the external buffer for resizing a 2-plane 4:2:0 image.
<a href="#">ResizeFilterGetSize</a>	Calculates the size of the state structure for resizing filter.
<a href="#">ResizeFilterInit</a>	Initializes the state structurer for resizing filter.
<a href="#">ResizeFilter</a>	Changes the size of an image using a generic filter.
<a href="#">Mirror</a>	Mirrors an image.
<a href="#">Remap</a>	Performs arbitrary remapping of pixels in an image.
<a href="#">Rotate</a>	Rotates an image around the origin (0,0) and shifts the result.
<a href="#">GetRotateShift</a>	Computes the shift values for <code>ippiRotate</code> , given the rotation center and angle.

Function Base Name	Operation
<a href="#">AddRotateShift</a>	Computes shift values for an image rotation around the specified center with specified shifts.
<a href="#">GetRotateQuad</a>	Computes the quadrangle that results from an image ROI transformed by <code>ippiRotate</code> function.
<a href="#">GetRotateBound</a>	Computes the bounding rectangle for an image ROI transformed by <code>ippiRotate</code> function.
<a href="#">RotateCenter</a>	Rotates an image around an arbitrary center.
<a href="#">Shear</a>	Performs a shearing transform of an image.
<a href="#">GetShearQuad</a>	Computes the quadrangle that results from an image ROI transformed by <code>ippiShear</code> function.
<a href="#">GetShearBound</a>	Computes the bounding rectangle for an image ROI transformed by <code>ippiShear</code> function.
<a href="#">WarpAffine</a>	Warpes an image with an affine transform.
<a href="#">WarpAffineBack</a>	Performs an inverse affine transform of an image.
<a href="#">WarpAffineQuad</a>	Performs an affine transform of an image from the source quadrangle to the destination quadrangle.
<a href="#">GetAffineQuad</a>	Computes the quadrangle that results from an image ROI transformed by <code>ippiWarpAffine</code> function.
<a href="#">GetAffineBound</a>	Computes the bounding rectangle for an image ROI transformed by <code>ippiWarpAffine</code> function.
<a href="#">GetAffineTransform</a>	Computes the affine transform coefficients.
<a href="#">WarpPerspective</a>	Warpes an image with a perspective transform.
<a href="#">WarpPerspectiveBack</a>	Performs an inverse perspective transform of an image.
<a href="#">WarpPerspectiveQuad</a>	Performs a perspective transform of an image from the source quadrangle to the destination quadrangle.
<a href="#">GetPerspectiveQuad</a>	Computes the quadrangle that results from an image ROI transformed by <code>ippiWarpPerspective</code> function.
<a href="#">GetPerspectiveBound</a>	Computes the bounding rectangle for an image ROI transformed by <code>ippiWarpPerspective</code> function.
<a href="#">GetPerspectiveTransform</a>	Computes the perspective transform coefficients.
<a href="#">WarpBilinear</a>	Warpes an image with a bilinear transform.
<a href="#">WarpBilinearBack</a>	Performs an inverse bilinear transform of an image.
<a href="#">WarpBilinearQuad</a>	Performs a bilinear transform of an image from the source quadrangle to the destination quadrangle.
<a href="#">GetBilinearQuad</a>	Computes the quadrangle that results from an image ROI transformed by <code>ippiWarpBilinear</code> function.
<a href="#">GetBilinearBound</a>	Computes the bounding rectangle for an image ROI transformed by <code>ippiWarpBilinear</code> function.
<a href="#">GetBilinearTransform</a>	Computes the bilinear transform coefficients.

Most functions performing geometric transform of an image uses some interpolation algorithm to resample the image. The type of interpolation method to be used is passed to the function in the *interpolation* parameter.

The following interpolation algorithms are used:

- nearest neighbor
- linear interpolation
- cubic convolution
- supersampling
- interpolation using Lanczos window function

- interpolation with two-parameter cubic filters
- optional edge smoothing of the destination image.

The nearest neighbor algorithm is the fastest, while other methods yield higher quality results, but are slower.

Use one of the following constant identifiers for the applicable interpolation methods:

<code>IPPI_INTER_NN</code>	Nearest neighbor interpolation.
<code>IPPI_INTER_LINEAR</code>	Linear interpolation.
<code>IPPI_INTER_CUBIC</code>	Cubic interpolation.
<code>IPPI_INTER_LANCZOS</code>	Interpolation using 3-lobed Lanczos window function.
<code>IPPI_INTER_CUBIC2P_BSPLINE</code>	Interpolation using B-spline.
<code>IPPI_INTER_CUBIC2P_CATMULLROM</code>	Interpolation using Catmull-Rom spline.
<code>IPPI_INTER_CUBIC2P_B05C03</code>	Interpolation using special cubic filter.
<code>IPPI_INTER_SUPER</code>	Supersampling interpolation.

For certain functions, you can combine the above interpolation algorithms with additional smoothing (antialiasing) of edges to which the original image borders are transformed. To use this edge smoothing, set the parameter *interpolation* to the bitwise OR of `IPPI_SMOOTH_EDGE` or `IPPI_SUBPIXEL_EDGE` and the desired interpolation mode.



**CAUTION.** You can use interpolation with edge smoothing option only in those geometric transform functions where this option is explicitly listed in the parameters definition section.

See [appendix B “Interpolation in Image Geometric Transform Functions”](#) for more information on the interpolation algorithms that are used in the library.

Intel IPP contains several functions for resizing. The main function is `ippiResizeSqrPixel`. This function is the particular case of the general affine transform with the 3x2 transformation matrix. It works with different interpolation modes and can be used for general resize operations in image processing, in different mathematic and geometric tasks.

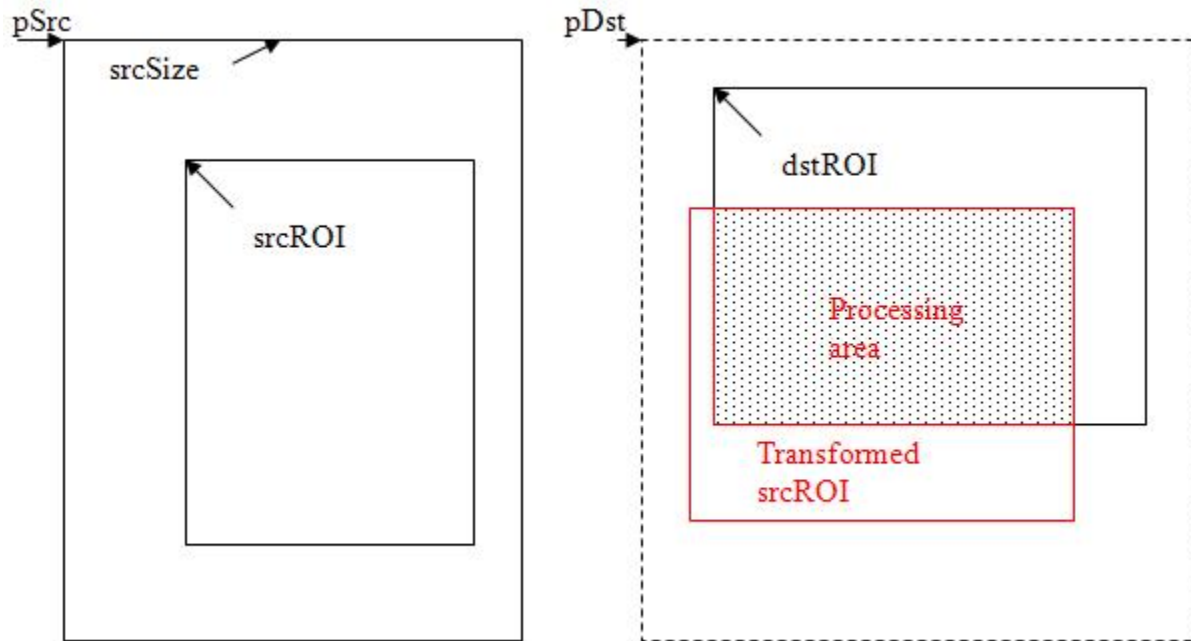
The function `SuperSampling` is the special function that perform only image reducing using only supersampling interpolation. It can be used for video processing, in which it demonstrates good quality and performance.

Many solutions and hints for use of these functions can be found in Intel® IPP Samples. See *Intel IPP® Image Processing and Media Processing Samples* downloadable from <http://www.intel.com/cd/software/products/asmo-na/eng/220046.htm>.

## ROI Processing in Geometric Transforms

All the transform functions described in this chapter operate in rectangular regions of interest (ROIs) that are defined in both the source and destination images. The procedures for handling ROIs in geometric transform functions differ from those used in other functions (see [Regions of Interest in Intel IPP](#) in chapter 2). The main difference is that operations take place in the intersection of the *transformed* source ROI and the destination ROI. More specifically, all geometric transform functions (except those which perform inverse warping operations) handle ROIs with the following sequence of operations (see figure below):

- transform the rectangular ROI of the source image to quadrangle in the destination image;
- find the intersection of this quadrangle and the rectangular ROI of the destination image;
- update the destination image in the intersection area.



The coordinates in the source and destination images must have the same origin.

Using functions with ROI enables every scan line of a source image to be padded with zeroes for alignment, so that actual size of a scan line in bytes can be greater than it may be assumed from the image width. In that case the size of each row of image data in bytes is determined by the value of `srcStep` parameter, which gives distance in bytes between the starts of consecutive lines of an image.

To fully describe a rectangular ROI, both its origin (coordinates of top left corner) and size must be referenced. For geometrical transform functions, the source image ROI is specified by `srcRoi` parameter of `IppiRect` type, meaning that all four values describing the rectangular ROI are given explicitly.

The destination image ROI for different functions can be specified either by `dstRoi` parameter of `IppiRect` type, or `dstRoiSize` parameter of `IppiSize` type. In the latter case, only the destination ROI size is passed, while its origin is referenced by `pDst` pointer.

## Geometric Transform Functions

### Resize

**THIS FUNCTION IS DEPRECATED.** Changes an image size.

---

#### Syntax

##### Case 1: Operation on pixel-order data

```
IppStatus ippiResize_<mod>(const Ipp<datatype>* pSrc, IppiSize srcSize, int srcStep,
IppiRect srcRoi, Ipp<datatype>* pDst, int dstStep, IppiSize dstRoiSize, double xFactor,
double yFactor, int interpolation);
```

Supported values for mod:

8u_C1R	16u_C1R	32f_C1R
8u_C3R	16u_C3R	32f_C3R
8u_C4R	16u_C4R	32f_C4R
8u_AC4R	16u_AC4R	32f_AC4R

##### Case 2: Operation on planar-order data

```
IppStatus ippiResize_<mod>(const Ipp<datatype>* const pSrc[3], IppiSize srcSize, int
srcStep, IppiRect srcRoi, Ipp<datatype>* const pDst[3], int dstStep, IppiSize dstRoiSize,
double xFactor, double yFactor, int interpolation);
```

Supported values for mod:

8u_P3R	16u_P3R	32f_P3R
--------	---------	---------

```
IppStatus ippiResize_<mod>(const Ipp<datatype>* const pSrc[4], IppiSize srcSize, int
srcStep, IppiRect srcRoi, Ipp<datatype>* const pDst[4], int dstStep, IppiSize dstRoiSize,
double xFactor, double yFactor, int interpolation);
```

Supported values for mod:

8u_P4R	16u_P4R	32f_P4R
--------	---------	---------

#### Parameters

<i>pSrc</i>	Pointer to the source image origin. An array of separate pointers to each plane in case of data in planar format.
<i>srcSize</i>	Size in pixels of the source image.

<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image buffer.										
<i>srcRoi</i>	Region of interest in the source image (of the <code>IppiRect</code> type).										
<i>pDst</i>	Pointer to the destination image ROI. An array of pointers to separate ROI in each planes for planar image.										
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image buffer.										
<i>dstRoiSize</i>	Size of the destination ROI in pixels.										
<i>xFactor, yFactor</i>	Factors by which the <i>x</i> and <i>y</i> dimensions of the source ROI are changed. The factor value greater than 1 corresponds to increasing the image size in that dimension.										
<i>interpolation</i>	Specifies the <a href="#">interpolation</a> mode. Use one of the following values: <table> <tr> <td><code>IPPI_INTER_NN</code></td><td>nearest neighbor interpolation</td></tr> <tr> <td><code>IPPI_INTER_LINEAR</code></td><td>linear interpolation</td></tr> <tr> <td><code>IPPI_INTER_CUBIC</code></td><td>cubic interpolation</td></tr> <tr> <td><code>IPPI_INTER_SUPER</code></td><td>supersampling interpolation, cannot be applied for image enlarging</td></tr> <tr> <td><code>IPPI_INTER_LANCZOS</code></td><td>interpolation with Lanczos window.</td></tr> </table>	<code>IPPI_INTER_NN</code>	nearest neighbor interpolation	<code>IPPI_INTER_LINEAR</code>	linear interpolation	<code>IPPI_INTER_CUBIC</code>	cubic interpolation	<code>IPPI_INTER_SUPER</code>	supersampling interpolation, cannot be applied for image enlarging	<code>IPPI_INTER_LANCZOS</code>	interpolation with Lanczos window.
<code>IPPI_INTER_NN</code>	nearest neighbor interpolation										
<code>IPPI_INTER_LINEAR</code>	linear interpolation										
<code>IPPI_INTER_CUBIC</code>	cubic interpolation										
<code>IPPI_INTER_SUPER</code>	supersampling interpolation, cannot be applied for image enlarging										
<code>IPPI_INTER_LANCZOS</code>	interpolation with Lanczos window.										

## Description




---

**CAUTION.** THIS FUNCTION IS DEPRECATED. Please use the function [ippiResizeSqrPixel](#) instead.

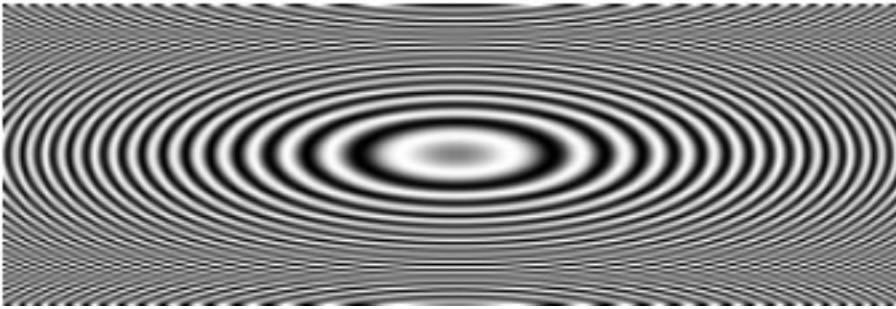
---

The function `ippiResize` is declared in the `ippi.h` file. It operates with ROI (see [ROI Processing in Geometric Transforms](#)).

This function resizes the source image ROI by *xFactor* in the *x* direction and *yFactor* in the *y* direction. The image size can be either reduced or increased in each direction, depending on the values of *xFactor*, *yFactor*. The result is resampled using the [interpolation mode](#) specified by the *interpolation* parameter, and written to the destination image ROI. Supersampling interpolation can be applied if both *xFactor* and *yFactor* are less

than or equal to 1. Figure “Sample Image Changed by the `ippiResize` Function” shows the result of applying the `ippiResize` function to the [sample image](#) of size 256 by 256 pixels. The destination image is of 300 by 100 pixels size.

### Sample Image Changed by the `ippiResize` Function



This function is used in the H.264 encoder included into Intel® IPP Samples downloadable from <http://www.intel.com/cd/software/products/asmo-na/eng/perflib/ipp/index.htm>.

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <code>pSrc</code> or <code>pDst</code> pointer is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <code>srcSize</code> or <code>dstRoiSize</code> has a field with zero or negative value.
<code>ippStsWrongIntersectROIErr</code>	Indicates an error condition if <code>srcRoi</code> has no intersection with the source image.
<code>ippStsResizeNoOperationErr</code>	Indicates an error condition if one of the destination image dimensions is less than 1 pixel.
<code>ippStsResizeFactorErr</code>	Indicates an error condition if <code>xFactor</code> or <code>yFactor</code> is less than or equal to zero.
<code>ippStsInterpolationErr</code>	Indicates an error condition if <code>interpolation</code> has an illegal value.

## Example

The code example below shows how to use the `ippiResize` function.

```
IppStatus resize( void ) {
    Ipp8u x[8*8], y[8*8];
    IppiSize srcsz={8,8}, dstroi={6,6};
    IppiRect srcroi={1,1,6,6};
    IppiSize roi = {8,8};
    int i;
    for( i=0; i<8; ++i ) {
        ippiSet_8u_C1R( (Ipp8u)i, x+8*i+i, 8, roi );
        --roi.width;
        --roi.height;    }
    return ippiResize_8u_C1R( x, srcsz, 8, srcroi, y, 8, dstroi, 2.9, 2, IPPI_INTER_NN );
}
```

The image has the following contents after resizing:

```
01 01 01 01 01 01 CC CC
01 01 01 01 01 01 CC CC
01 01 02 02 02 02 CC CC
01 01 02 02 02 02 CC CC
01 01 02 02 02 03 CC CC
01 01 02 02 02 03 CC CC
CC CC CC CC CC CC CC
CC CC CC CC CC CC CC
```



## ResizeCenter

**THIS FUNCTION IS DEPRECATED.** Changes an image size and shifts the destination image relative to the given point.

### Syntax

#### Case 1: Operation on pixel-order data

```
IppStatus ippiResizeCenter_<mod>(const Ipp<datatype>* pSrc, IppiSize srcSize, int srcStep,
IppiRect srcRoi, Ipp<datatype>* pDst, int dstStep, IppiSize dstRoiSize, double xFactor,
double yFactor, double xCenter, double yCenter, int interpolation);
```

Supported values for mod:

8u_C1R	16u_C1R	32f_C1R
8u_C3R	16u_C3R	32f_C3R
8u_C4R	16u_C4R	32f_C4R
8u_AC4R	16u_AC4R	32f_AC4R

#### Case 2: Operation on planar-order data

```
IppStatus ippiResizeCenter_<mod>(const Ipp<datatype>* const pSrc[3], IppiSize srcSize,
int srcStep, IppiRect srcRoi, Ipp<datatype>* const pDst[3], int dstStep, IppiSize
dstRoiSize, double xFactor, double yFactor, double xCenter, double yCenter, int
interpolation);
```

Supported values for mod:

8u_P3R	16u_P3R	32f_P3R
--------	---------	---------

```
IppStatus ippiResizeCenter_<mod>(const Ipp<datatype>* const pSrc[4], IppiSize srcSize,
int srcStep, IppiRect srcRoi, Ipp<datatype>* const pDst[4], int dstStep, IppiSize
dstRoiSize, double xFactor, double yFactor, double xCenter, double yCenter, int
interpolation);
```

Supported values for mod:

8u_P4R	16u_P4R	32f_P4R
--------	---------	---------

### Parameters

<i>pSrc</i>	Pointer to the source image origin. An array of 3 or 4 pointers to different color planes in case of data in planar format.
<i>srcSize</i>	Size in pixels of the source image.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image buffer.

<i>srcRoi</i>	Region of interest in the source image (of the <code>IppiRect</code> type).										
<i>pDst</i>	Pointer to the destination image ROI. An array of 3 or 4 pointers to destination ROI in different color planes for planar image.										
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image buffer.										
<i>dstRoiSize</i>	Size of the destination ROI in pixels.										
<i>xFactor, yFactor</i>	Factors by which the <i>x</i> and <i>y</i> dimensions of the source ROI are changed. The factor value greater than 1 corresponds to increasing the image size in that dimension.										
<i>xCenter, yCenter</i>	Coordinates of the preset center.										
<i>interpolation</i>	Specifies the <a href="#">interpolation</a> mode. Use one of the following values: <table> <tr> <td><code>IPPI_INTER_NN</code></td><td>nearest neighbor interpolation</td></tr> <tr> <td><code>IPPI_INTER_LINEAR</code></td><td>linear interpolation</td></tr> <tr> <td><code>IPPI_INTER_CUBIC</code></td><td>cubic interpolation</td></tr> <tr> <td><code>IPPI_INTER_SUPER</code></td><td>supersampling interpolation, cannot be applied for image enlarging</td></tr> <tr> <td><code>IPPI_INTER_LANCZOS</code></td><td>interpolation with Lanczos window.</td></tr> </table>	<code>IPPI_INTER_NN</code>	nearest neighbor interpolation	<code>IPPI_INTER_LINEAR</code>	linear interpolation	<code>IPPI_INTER_CUBIC</code>	cubic interpolation	<code>IPPI_INTER_SUPER</code>	supersampling interpolation, cannot be applied for image enlarging	<code>IPPI_INTER_LANCZOS</code>	interpolation with Lanczos window.
<code>IPPI_INTER_NN</code>	nearest neighbor interpolation										
<code>IPPI_INTER_LINEAR</code>	linear interpolation										
<code>IPPI_INTER_CUBIC</code>	cubic interpolation										
<code>IPPI_INTER_SUPER</code>	supersampling interpolation, cannot be applied for image enlarging										
<code>IPPI_INTER_LANCZOS</code>	interpolation with Lanczos window.										

## Description




---

**CAUTION.** THIS FUNCTION IS DEPRECATED. Please use the function [ippiResizeSqrPixel](#) instead.

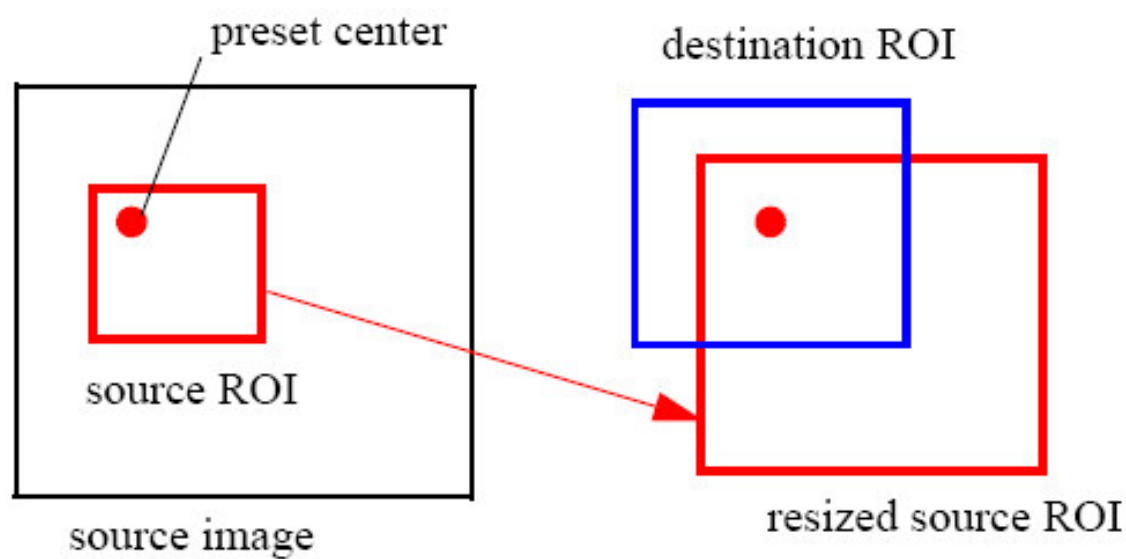
---

The function `ippiResizeCenter` is declared in the `ippi.h` file. It operates with ROI (see [ROI Processing in Geometric Transforms](#)).

This function resizes the source image ROI by *xFactor* in the *x* direction and *yFactor* in the *y* direction. The image size can be either reduced or increased in each direction, depending on the values of *xFactor*, *yFactor*. Unlike [ippiResize](#), this function also shifts the destination image in such a way that a given point with coordinate *xCenter*, *yCenter* - preset center - remains stationary after resizing, which creates an effect of changing the image size relative to that point. The preset center is always positioned in the center of the destination image

ROI with coordinates  $dstROI.width/2; dstROI.height/2$  (see Figure “Resizing Image with the Function `ippiResizeCenter`”). Note that the preset center may be specified in an arbitrary point including points outside the source image..

### Resizing Image with the Function `ippiResizeCenter`



The result is resampled using the [interpolation mode](#) specified by the `interpolation` parameter, and written to the destination image ROI. Supersampling interpolation can be applied if both `xFactor` and `yFactor` are less than or equal to 1.

## Example

The code example below shows how to use the `ippiResizeCenter` function.

```
IppStatus resizeCenter( void ) {
    Ipp8u x[8*8], y[8*8];
    IppiSize sz = {8,8}, roi = {8,8};
    IppiRect rect = {0,0,8,8};
    int i;
    for( i=0; i<8; ++i ) {
        ippiSet_8u_C1R( (Ipp8u)i, x+8*i+i, 8, roi );
        --roi.width;
        --roi.height;
    }
    return ippiResizeCenter_8u_C1R( x, sz, 8, rect, y, 8, sz,
        2.9, 2.0, 4, 4, IPPI_INTER_NN );
}
```

The image has the following contents after applying the `ippiResizeCenter` function:

```
02 02 02 02 02 02 02 02
02 02 02 02 02 02 02 02
02 02 03 03 03 03 03 03
02 02 03 03 03 03 03 03
02 02 03 03 03 04 04 04
02 02 03 03 03 04 04 04
02 02 03 03 03 04 04 04
02 02 03 03 03 04 04 04
```

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <code>pSrc</code> or <code>pDst</code> pointer is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <code>srcSize</code> or <code>dstRoiSize</code> has a field with zero or negative value.
<code>ippStsWrongIntersectROI</code>	Indicates a warning if <code>srcRoi</code> has no intersection with the source image.
<code>ippStsResizeNoOperationErr</code>	Indicates an error condition if one of the destination image dimensions is less than 1 pixel.

<code>ippStsResizeFactorErr</code>	Indicates an error condition if <i>xFactor</i> or <i>yFactor</i> is less than or equal to zero.
<code>ippStsInterpolationErr</code>	Indicates an error condition if <i>interpolation</i> has an illegal value.

## ResizeSqrPixel

*Changes an image size using different interpolation algorithm.*

---

### Syntax

#### Case 1: Operation on pixel-order data

```
IppStatus ippResizeSqrPixel_<mod>(const Ipp<datatype>* pSrc, IppiSize srcSize, int srcStep,
IppiRect srcRoi, Ipp<datatype>* pDst, int dstStep, IppiRect dstRoi, double xFactor, double
yFactor, double xShift, double yShift, int interpolation, Ipp8u* pBuffer);
```

Supported values for `mod`:

```
8u_C1R  16u_C1R  16s_C1R  32f_C1R  64f_C1R
8u_C3R  16u_C3R  16s_C3R  32f_C3R  64f_C3R
8u_C4R  16u_C4R  16s_C4R  32f_C4R  64f_C4R
8u_AC4R 16u_AC4R 16s_AC4R 32f_AC4R 64f_AC4R
```

#### Case 2: Operation on planar-order data

```
IppStatus ippResizeSqrPixel_<mod>(const Ipp<datatype>* const pSrc[3], IppiSize srcSize,
int srcStep, IppiRect srcRoi, Ipp<datatype>* const pDst[3], int dstStep, IppiRect dstRoi,
double xFactor, double yFactor, double xShift, double yShift, int interpolation, Ipp8u*
pBuffer);
```

Supported values for `mod`:

```
8u_P3R  16u_P3R  16s_P3R  32f_P3R  64f_P3R
```

```
IppStatus ippResizeSqrPixel_<mod>(const Ipp<datatype>* const pSrc[4], IppiSize srcSize,
int srcStep, IppiRect srcRoi, Ipp<datatype>* const pDst[4], int dstStep, IppiRect dstRoi,
double xFactor, double yFactor, double xShift, double yShift, int interpolation, Ipp8u*
pBuffer);
```

Supported values for `mod`:

```
8u_P4R  16u_P4R  16s_P4R  32f_P4R  64f_P4R
```

### Parameters

*pSrc*                      Pointer to the source image origin. An array of 3 or 4 pointers to the planes for planar image.

<i>srcSize</i>	Size in pixels of the source image.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image buffer.
<i>srcRoi</i>	Region of interest in the source image (of the <code>IppiRect</code> type).
<i>pDst</i>	Pointer to the destination image. An array of 3 or 4 pointers to the planes for planar image.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image buffer.
<i>dstRoi</i>	Region of interest in the destination image (of the <code>IppiRect</code> type).
<i>xFactor, yFactor</i>	Factors by which the <i>x</i> and <i>y</i> dimensions of the source ROI are changed. The factor value greater than 1 means increasing the image size in that dimension.
<i>xShift, yShift</i>	Shift values in the <i>x</i> and <i>y</i> directions respectively.
<i>interpolation</i>	Specifies the <a href="#">interpolation</a> mode. Possible values are: IPPI_INTER_NN - nearest neighbor interpolation IPPI_INTER_LINEAR - linear interpolation IPPI_INTER_CUBIC - cubic interpolation IPPI_INTER_LANCZOS - interpolation with Lanczos window IPPI_INTER_CUBIC2P_BSPLINE - B-spline IPPI_INTER_CUBIC2P_CATMULLROM - Catmull-Rom spline IPPI_INTER_CUBIC2P_B05C03 - special two-parameter cubic filter IPPI_INTER_SUPER - supersampling interpolation the following flags are used additionally to the above modes: IPPI_SMOOTH_EDGE, or IPPI_SUBPIXEL_EDGE - edge smoothing IPPI_ANTIALIASING - antialiasing method, is not applicable with IPPI_INTER_NN or IPPI_INTER_SUPER
<i>pBuffer</i>	Pointer to the external buffer.

## Description

The function `ippiResizeSqrPixel` is declared in the `ippi.h` file. It operates with ROI (see [ROI Processing in Geometric Transforms](#)).

This function works with different interpolation modes and can be used for general resize operations in image processing, in different mathematic and geometric tasks.

This function resizes the source image ROI by *xFactor* in the *x* direction and *yFactor* in the *y* direction. The image size can be either reduced or increased in each direction, depending on the values of *xFactor*, *yFactor*. The result is resampled using the [interpolation mode](#) specified by the *interpolation* parameter, and written to the destination image ROI. Interpolation modes with two-parameter cubic filters perform additional filtering to improve quality of the destination image.

Pixel coordinates *x'* and *y'* in the resized image are obtained from the following equations:

$$x' = xFactor * x + xShift$$

$$y' = yFactor * y + yShift$$

where  $x$  and  $y$  denote the pixel coordinates in the source image.

The interpolation algorithms may skip edge pixels in the source image out of the ROI. The function `ippiResizeSqrPixel` uses in calculation the weighted values of these outer pixels. If a source edge pixel and outer pixel have intersection  $a < 1$  in the corresponding direction, that pixels' value is taken with the weight  $a$ . If no edge smoothing flag is set, the function does not process the outer pixels with the weight less than 0.5.

If the flag `IPPI_SMOOTH_EDGE` is set, then the function processes outer pixels with the weight of original destination background:

$$DstRes = SrcSampled * (1 - a) + DstExist * a;$$

If the flag `IPPI_SUBPIXEL_EDGE` is set, then the function proceeds outer pixels without the weight of original destination background:

$$DstRes = SrcSampled * (1 - a);$$

where *SrcSampled* - intensity if the source pixel after resizing, *DstExist* - intensity of the destination pixel before resizing, *DstRes* - intensity of the result destination pixel,  $a$  - weight of the outer pixel; it is set by the function.



**NOTE.** The size of the destination image can vary depending on the edge smoothing settings.

The edge smoothing does not work on images with the high contrast borders.

If an image is compressed, use antialiasing method to limit the impact of aliasing (artifacts). In this case the function modifies (stretches out) the interpolation kernel and reduces amplitude; resampling is performed on the  $x$  and  $y$  directions independently. This method can be performed by setting *interpolation* to `IPPI_ANTIALIASING`. The factor values must be less than 1, and *interpolation* can not be set to `IPPI_INTER_NN` or `IPPI_INTER_SUPER`.



**CAUTION.** Only one of the flags `IPPI_SMOOTH_EDGE`, `IPPI_SUBPIXEL_EDGE`, `IPPI_ANTIALIASING` can be used.

The function requires the external buffer *pBuffer*, its size must be previously computed by calling the function `ippiResizeGetBufSize_64f` for 64f flavors, and `ippiResizeGetBufSize` for all other flavors.

## Application Notes

The source and destination image ROI are specified in independent systems of coordinates. The destination pixels are mapped back to the source image (see figure in [ROI Processing in Geometric Transforms](#)) and interpolated if they hit it. The following pseudocode approximately describes the process:

```
For (j = dstROI.y; j < dstROI.y + dstROI.height; j++) {
    For (i = dstROI.x; i < dstROI.x + dstROI.width; i++) {
        xCoordSrc = 1 / xFactor * (j + 0.5) - xShift / xFactor;
        yCoordSrc = 1 / yFactor * (i + 0.5) - yShift / yFactor;
        if (pixel (xCoordSrc, xCoordSrc) is in source image) {
            interpolate;
        }
    }
}
```

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if width or height of images or source image ROI has zero or negative value, or if one of the specified coordinates of the image ROI origin is negative.
<code>ippStsResizeFactorErr</code>	Indicates an error condition if <i>xFactor</i> or <i>yFactor</i> is less than or equal to zero.
<code>ippStsInterpolationErr</code>	Indicates an error condition if <i>interpolation</i> has an illegal value.
<code>ippStsWrongIntersectROI</code>	Indicates a warning if <i>srcRoi</i> has no intersection with the source image.
<code>ippStsNoAntialiasing</code>	Indicates a warning if the selected interpolation mode does not support antialiasing.

## Example

The code example below shows how the Intel IPP functions can be used to resample the source image of size 12x12 to the destination image of size 6x6 .



Please note that this code is developed for demonstration purposes only.

```
IppStatus ResizeSqrPixel( void )
{
    Ipp32f src[12*12], dst[12*12];
    IppiSize size = {12,12};
    IppiRect srect = {0,0,12,12};
    IppiRect drect = {0,0,6,6};
    Ipp8u *buf;
    int bufsize , I;
    IppStatus status = ippStsNoErr;
```

```

/* source image */
{
    IppiSize roi = {12,12};
    for( I=0; i<12; ++I ) {
        ippiSet_32f_C1R( (Ipp32f)I, src+12*i+i, 12*sizeof(Ipp32f), roi );
        --roi.width;
        --roi.height;
    }
}

/* calculation of work buffer size */
ippiResizeGetBufSize( srect, direct, 1, IPPI_INTER_SUPER, &bufsize );
/* memory allocate */
buf = ippsMalloc_8u( bufsize );
/* function call */
if( NULL != buf )
    status = ippiResizeSqrPixel_32f_C1R(
        src, size, 12*sizeof(Ipp32f), srect,
        dst, 6*sizeof(Ipp32f), direct,
        0.5, 0.5, 0, 0, IPPI_INTER_SUPER, buf );
/* memory free */
if( NULL != buf ) ippsFree( buf );
return status;
}

```

The source image

```

0 0 0 0 0 0 0 0 0 0 0 0
0 1 1 1 1 1 1 1 1 1 1 1
0 1 2 2 2 2 2 2 2 2 2 2
0 1 2 3 3 3 3 3 3 3 3 3
0 1 2 3 4 4 4 4 4 4 4 4
0 1 2 3 4 5 5 5 5 5 5 5
0 1 2 3 4 5 6 6 6 6 6 6
0 1 2 3 4 5 6 7 7 7 7 7
0 1 2 3 4 5 6 7 8 8 8 8

```

```
0 1 2 3 4 5 6 7 8 9 9 9
0 1 2 3 4 5 6 7 8 9 10 10
0 1 2 3 4 5 6 7 8 9 10 11
```

The image has the following contents after resizing with NEAREST NEIGHBOR interpolation

```
1.0000000  1.0000000  1.0000000  1.0000000  1.0000000  1.0000000
1.0000000  3.0000000  3.0000000  3.0000000  3.0000000  3.0000000
1.0000000  3.0000000  5.0000000  5.0000000  5.0000000  5.0000000
1.0000000  3.0000000  5.0000000  7.0000000  7.0000000  7.0000000
1.0000000  3.0000000  5.0000000  7.0000000  9.0000000  9.0000000
1.0000000  3.0000000  5.0000000  7.0000000  9.0000000  11.0000000
```

The image has the following contents after resizing with LINEAR interpolation

```
0.25000000  0.5000000  0.5000000  0.5000000  0.5000000  0.5000000
0.50000000  2.2500000  2.5000000  2.5000000  2.5000000  2.5000000
0.50000000  2.5000000  4.2500000  4.5000000  4.5000000  4.5000000
0.50000000  2.5000000  4.5000000  6.2500000  6.5000000  6.5000000
0.50000000  2.5000000  4.5000000  6.5000000  8.2500000  8.5000000
0.50000000  2.5000000  4.5000000  6.5000000  8.5000000  11.2500000
```

The image has the following contents after resizing with CUBIC interpolation

```
0.25390625  0.4335938  0.4375000  0.4375000  0.4375000  0.4375000
0.43359375  2.3828125  2.4960938  2.5000000  2.5000000  2.5000000
0.43750000  2.4960938  4.3828125  4.4960938  4.5000000  4.5000000
0.43750000  2.5000000  4.4960938  6.3828125  6.4960938  6.5000000
0.43750000  2.5000000  4.5000000  6.4960938  8.3828125  8.4960938
0.43750000  2.5000000  4.5000000  6.5000000  8.4960938  10.3789060
```

The image has the following contents after resizing with LANCZOS interpolation

```
0.26301101  0.3761742  0.4124454  0.4130435  0.4130435  0.4130435
0.37617415  2.4499352  2.4631310  2.4994020  2.5000002  2.5000002
```

```
0.41244543  2.4631310  4.4499354  4.4631310  4.4994025  4.5000005
0.41304356  2.4994023  4.4631310  6.4499359  6.4631314  6.4994025
0.41304356  2.5000005  4.4994020  6.4631314  8.4499359  8.4631310
0.41304356  2.5000005  4.5000000  6.4994025  8.4631310 10.4369250
```

The image has the following contents after resizing with SUPERSAMPLING interpolation

```
0.25000000  0.5000000  0.5000000  0.5000000  0.5000000  0.5000000
0.50000000  2.2500000  2.5000000  2.5000000  2.5000000  2.5000000
0.50000000  2.5000000  4.2500000  4.5000000  4.5000000  4.5000000
0.50000000  2.5000000  4.5000000  6.2500000  6.5000000  6.5000000
0.50000000  2.5000000  4.5000000  6.5000000  8.2500000  8.5000000
0.50000000  2.5000000  4.5000000  6.5000000  8.5000000 10.2500000
```

## ResizeGetBufSize

*Computes the size of the external buffer for image resizing.*

---

### Syntax

```
IppStatus ippiResizeGetBufSize(IppiRect srcRoi, IppiRect dstRoi, int nChannel, int
interpolation, int* pBufferSize);
```

```
IppStatus ippiResizeGetBufSize_64f(IppiRect srcRoi, IppiRect dstRoi, int nChannel, int
interpolation, int* pBufferSize);
```

### Parameters

<i>srcRoi</i>	Region of interest in the source image (of the <code>IppiRect</code> type).
<i>dstRoi</i>	Region of interest in the destination image (of the <code>IppiRect</code> type).
<i>nChannel</i>	Number of channels or planes, possible values are 1, 3, 4.
<i>interpolation</i>	Specifies the <a href="#">interpolation</a> mode. Use one of the following values: IPPI_INTER_NN - nearest neighbor interpolation IPPI_INTER_LINEAR - linear interpolation IPPI_INTER_CUBIC - cubic interpolation IPPI_INTER_LANCZOS - interpolation with Lanczos window IPPI_INTER_CUBIC2P_BSPLINE - B-spline IPPI_INTER_CUBIC2P_CATMULLROM - Catmull-Rom spline IPPI_INTER_CUBIC2P_B05C03 - special two-parameters cubic filter

IPPI\_INTER\_SUPER - supersampling interpolation  
 IPPI\_SMOOTH\_EDGE, or IPPI\_SUBPIXEL\_EDGE - edge smoothing in addition to one of the above modes  
 IPPI\_ANTIALIASING - antialiasing method in addition to the linear, cubic, or Lanczos modes.  
*pBuffer* Pointer to the external buffer.

## Description

The function `ippiResizeGetBufSize` is declared in the `ippi.h` file. This function computes the size (in bytes) of the external buffer that is required for the function `ippiResizeSqrPixel`.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pBufferSize</i> pointer is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if width or height of the <i>srcRoi</i> or <i>dstRoi</i> is less than or equal to 0.
<code>ippStsNumChannelErr</code>	Indicates an error condition if <i>nChannel</i> has an illegal value.
<code>ippStsInterpolationErr</code>	Indicates an error condition if <i>interpolation</i> has an illegal value.
<code>ippStsNoAntialiasing</code>	Indicates a warning if the selected interpolation mode does not support antialiasing.

## GetResizeRect

*Compute coordinates of the resized image.*

## Syntax

```
IppStatus ippiGetResizeRect(IppiRect srcRoi, IppiRect* pDstRect, double xFactor, double yFactor, double xShift, double yShift, int interpolation);
```

## Parameters

<i>srcRoi</i>	Region of interest in the source image (of the <code>IppiRect</code> type).
<i>pDstRect</i>	Pointer to the resultant rectangle (of the <code>IppiRect</code> type).
<i>xFactor</i> , <i>yFactor</i>	Factors by which the <i>x</i> and <i>y</i> dimensions of the source ROI are changed. The factor value greater than 1 means increasing the image size in that dimension.
<i>xShift</i> , <i>yShift</i>	Shift values in the <i>x</i> and <i>y</i> directions respectively.
<i>interpolation</i>	Specifies the <a href="#">interpolation</a> mode. Use one of the following values: IPPI_INTER_NN - nearest neighbor interpolation IPPI_INTER_LINEAR - linear interpolation IPPI_INTER_CUBIC - cubic interpolation IPPI_INTER_LANCZOS - interpolation with Lanczos window IPPI_INTER_CUBIC2P_BSPLINE - B-spline

`IPPI_INTER_CUBIC2P_CATMULLROM` - Catmull-Rom spline  
`IPPI_INTER_CUBIC2P_B05C03` - special two-parameters cubic filter  
`IPPI_INTER_SUPER` - supersampling interpolation  
`IPPI_SMOOTH_EDGE`, or `IPPI_SUBPIXEL_EDGE` - edge smoothing in addition to one of the above modes  
`IPPI_ANTIALIASING` - antialiasing method in addition to the linear, cubic, or Lanczos modes.

## Description

The function `ippiGetResizeRect` is declared in the `ippi.h` file. It operates with ROI (see [ROI Processing in Geometric Transforms](#)).

This function returns the pointer to the resized image ROI `pDstRect` of the `IppiRect` type, that is it calculates the coordinates and size of the rectangular destination image ROI, which is obtained if the source image ROI `srcRoi` is resized with the specified parameters. The resize operation is not performed.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <code>pDstRect</code> pointer is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if width or height of the source image ROI has zero or negative value.
<code>ippStsResizeFactorErr</code>	Indicates an error condition if <code>xFactor</code> or <code>yFactor</code> is less than or equal to zero.
<code>ippStsInterpolationErr</code>	Indicates an error condition if <code>interpolation</code> has an illegal value.
<code>ippStsNoOperation</code>	Indicates a warning if processed area is disappeared, no operation is performed.

## ResizeSqrPixelGetBufSize

***THIS FUNCTION IS DEPRECATED.*** Computes the size of the external buffer for image resizing.

---

## Syntax

```
IppStatus ippiResizeSqrPixelGetBufSize(IppiSize dstSize, int nChannel, int interpolation,
int* pBufferSize);
```

## Parameters

<code>dstSize</code>	Size in pixels of the destination image.
<code>nChannel</code>	Number of channels or planes, possible values are 1, 3, 4.
<code>interpolation</code>	Specifies the <a href="#">interpolation</a> mode. Use one of the following values: <code>IPPI_INTER_NN</code> - nearest neighbor interpolation <code>IPPI_INTER_LINEAR</code> - linear interpolation <code>IPPI_INTER_CUBIC</code> - cubic interpolation

IPPI\_INTER\_LANCZOS - interpolation with Lanczos window  
 IPPI\_INTER\_CUBIC2P\_BSPLINE - B-spline  
 IPPI\_INTER\_CUBIC2P\_CATMULLROM - Catmull-Rom spline  
 IPPI\_INTER\_CUBIC2P\_B05C03 - special two-parameters cubic filter  
 Pointer to the size of the external buffer.

*pBufferSize*

## Description



**CAUTION.** THIS FUNCTION IS DEPRECATED. Please use the function [ippiResizeGetBufSize](#) instead.

The function `ippiResizeSqrPixelGetBufSize` is declared in the `ippi.h` file. This function computes the size of the external buffer that is required for the function [ippiResizeSqrPixel](#).

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pBufferSize</i> pointer is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <i>dstSize</i> has a field with zero or negative value.
<code>ippStsNumChannelErr</code>	Indicates an error condition if <i>nChannel</i> has an illegal value.
<code>ippStsInterpolationErr</code>	Indicates an error condition if <i>interpolation</i> has an illegal value.

## GetResizeFract

**THIS FUNCTION IS DEPRECATED.** Recalculates resize factors for a tiled image.

## Syntax

```
IppStatus ippiGetResizeFract(IppiSize srcSize, IppiRect srcRoi, double xFactor, double yFactor, double* xFr, double* yFr, int interpolation);
```

## Parameters

<i>srcSize</i>	Size of the source image in pixels.
<i>srcRoi</i>	Region of interest in the source image (of the <code>IppiRect</code> type).
<i>xFactor</i> , <i>yFactor</i>	Factors by which the <i>x</i> and <i>y</i> dimensions of the source ROI are changed. The factor value greater than 1 corresponds to increasing the image size in that dimension.
<i>xFr</i> , <i>yFr</i>	Pointers to the recalculated resize factors. These values should be passed to <a href="#">ippiResizeShift</a> function to bring about the desired resizing of image tiles. The factor value greater than 1 corresponds to increasing the image size in that dimension.

<i>interpolation</i>	Specifies the <a href="#">interpolation</a> mode. Use one of the following values:
IPPI_INTER_NN	nearest neighbor interpolation
IPPI_INTER_LINEAR	linear interpolation
IPPI_INTER_CUBIC	cubic interpolation
IPPI_INTER_LANCZOS	interpolation with Lanczos window.

## Description



**CAUTION.** THIS FUNCTION IS DEPRECATED. Please use the function [ippiResizeSqrPixel](#) for tile processing.

The function `ippiGetResizeFract` is declared in the `ippi.h` file. It operates with ROI (see [ROI Processing in Geometric Transforms](#)).

Use this function if you need to resize the tiled source image by *xFactor* in the *x* direction and *yFactor* in the *y* direction. It calculates the new values of resize factors *xFr* and *yFr* that should be passed to the [ippiResizeShift](#) function, which performs resize operation on the individual tiles.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>srcSize</i> has a field with zero or negative value.
<code>ippStsResizeFactorErr</code>	Indicates an error condition if <i>xFactor</i> or <i>yFactor</i> is less than or equal to zero.
<code>ippStsInterpolationErr</code>	Indicates an error condition if <i>interpolation</i> has an illegal value.
<code>ippStsWrongIntersectROIErr</code>	Indicates an error condition if <i>srcRoi</i> has no intersection with the source image.

## ResizeShift

**THIS FUNCTION IS DEPRECATED.** *Changes the size of an image tile.*

## Syntax

### Case 1: Operation on pixel-order data

```
ippStatus ippiResizeShift_<mod>(const Ipp<datatype>* pSrc, IppiSize srcSize, int srcStep,
IppiRect srcRoi, Ipp<datatype>* pDst, int dstStep, IppiSize dstRoiSize, double xFr, double
yFr, double xShift, double yShift, int interpolation);
```

Supported values for *mod*:

8u_C1R	16u_C1R	32f_C1R
8u_C3R	16u_C3R	32f_C3R



8u_C4R	16u_C4R	32f_C4R
8u_AC4R	16u_AC4R	32f_AC4R

### Case 2: Operation on planar-order data

```
IppStatus ippiResizeShift_<mod>(const Ipp<datatype>* const pSrc[3], IppiSize srcSize, int
srcStep, IppiRect srcRoi, Ipp<datatype>* const pDst[3], int dstStep, IppiSize dstRoiSize,
double xFr, double yFr, double xShift, double yShift, int interpolation);
```

Supported values for mod:

8u_P3R	16u_P3R	32f_P3R
--------	---------	---------

```
IppStatus ippiResizeShift_<mod>(const Ipp<datatype>* const pSrc[4], IppiSize srcSize, int
srcStep, IppiRect srcRoi, Ipp<datatype>* const pDst[4], int dstStep, IppiSize dstRoiSize,
double xFr, double yFr, double xShift, double yShift, int interpolation);
```

Supported values for mod:

8u_P4R	16u_P4R	32f_P4R
--------	---------	---------

### Parameters

<i>pSrc</i>	Pointer to the source image. An array of 3 or 4 pointers to different color planes in case of data in planar format.								
<i>srcSize</i>	Size of the source image in pixels.								
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image buffer.								
<i>srcRoi</i>	Region of interest in the source image (of the <code>IppiRect</code> type).								
<i>pDst</i>	Pointer to the destination image ROI. An array of 3 or 4 pointers to destination ROI in different color planes for planar image.								
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image buffer.								
<i>dstRoiSize</i>	Size of the destination ROI in pixels.								
<i>xFr, yFr</i>	Factors, that are used to obtain the desired resizing of the source image ROI by the <i>xFactor</i> , <i>yFactor</i> .								
<i>xShift, yShift</i>	Corrections of the position of the processing area ( <i>srcRect</i> ).								
<i>interpolation</i>	Specifies the <a href="#">interpolation</a> mode. Use one of the following values: <table> <tbody> <tr> <td>IPPI_INTER_NN</td><td>nearest neighbor interpolation</td></tr> <tr> <td>IPPI_INTER_LINEAR</td><td>linear interpolation</td></tr> <tr> <td>IPPI_INTER_CUBIC</td><td>cubic interpolation</td></tr> <tr> <td>IPPI_INTER_LANCZOS</td><td>interpolation with Lanczos window.</td></tr> </tbody> </table>	IPPI_INTER_NN	nearest neighbor interpolation	IPPI_INTER_LINEAR	linear interpolation	IPPI_INTER_CUBIC	cubic interpolation	IPPI_INTER_LANCZOS	interpolation with Lanczos window.
IPPI_INTER_NN	nearest neighbor interpolation								
IPPI_INTER_LINEAR	linear interpolation								
IPPI_INTER_CUBIC	cubic interpolation								
IPPI_INTER_LANCZOS	interpolation with Lanczos window.								

## Description



**CAUTION.** THIS FUNCTION IS DEPRECATED. Please use the function [ippiResizeSqrPixel](#) for tile processing.

The function `ippiResizeShift` is declared in the `ippi.h` file. It operates with ROI (see [ROI Processing in Geometric Transforms](#)).

This function is used for resizing the tiled image by *xFactor* in the *x* direction and *yFactor* in the *y* direction. Prior to using it, the [ippiGetResizeFract](#) function should be called to calculate the values of the *xFr* and *yFr* factors. The image size can be either reduced or increased in each direction, depending on the values of *xFactor*, *yFactor*. The value of *xFr* or *yFr* factor greater than 1 corresponds to reducing the size, and the value less than 1 corresponds to increasing the image size in that dimension. The result is resampled using the [interpolation mode](#) specified by the *interpolation* parameter, and written to the destination image ROI.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <i>srcSize</i> or <i>dstRoiSize</i> has a field with zero or negative value.
<code>ippStsResizeFactorErr</code>	Indicates an error condition if <i>xFr</i> or <i>yFr</i> is less than or equal to zero.
<code>ippStsInterpolationErr</code>	Indicates an error condition if <i>interpolation</i> has an illegal value.
<code>ippStsWrongIntersectROIErr</code>	Indicates an error condition if <i>srcRoi</i> has no intersection with the source image.

## SuperSampling

Reduces the image size using the super sampling method.

### Syntax

#### Case 1: Operation on pixel-order data

```
IppStatus ippiSuperSampling_<mod>(const Ipp<datatype>* pSrc, int srcStep, IppiSize srcRoiSize, Ipp<datatype>* pDst, int dstStep, IppiSize dstRoiSize, Ipp8u* pBuffer);
```

Supported values for *mod*:

8u_C1R	16u_C1R	32f_C1R
8u_C3R	16u_C3R	32f_C3R
8u_C4R	16u_C4R	32f_C4R
8u_AC4R	16u_AC4R	32f_AC4R

**Case 2: Operation on planar-order data**

```
IppStatus ippiSuperSampling_<mod>(const Ipp<datatype>* const pSrc[3], int srcStep, IppiSize srcRoiSize, Ipp<datatype>* const pDst[3], int dstStep, IppiSize dstRoiSize, Ipp8u* pBuffer);
```

Supported values for mod:

8u\_P3R            16u\_P3R            32f\_P3R

```
IppStatus ippiSuperSampling_<mod>(const Ipp<datatype>* const pSrc[4], int srcStep, IppiSize srcRoiSize, Ipp<datatype>* const pDst[4], int dstStep, IppiSize dstRoiSize, Ipp8u* pBuffer);
```

Supported values for mod:

8u\_P4R            16u\_P4R            32f\_P4R

**Parameters**

<i>pSrc</i>	Pointer to the source image ROI. An array of pointers to separate ROIs in each plane for planar image.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image buffer.
<i>srcRoiSize</i>	Size of the source image ROI in pixels.
<i>pDst</i>	Pointer to the destination image ROI. An array of pointers to separate ROIs in each plane for planar image.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image buffer.
<i>dstRoiSize</i>	Size of the destination image ROI in pixels.
<i>pBuffer</i>	Pointer to the external buffer.

**Description**

The function `ippiSuperSampling` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function reduces the source image *pSrc* using the [super sampling interpolation](#). The size of the external buffer must be calculated using the functions [ippiSuperSamplingGetBufSize](#) beforehand.

Use this function for video processing, in which it demonstrates good quality and performance.



**NOTE.** Performance is better if you use the following scaling factors along the x and y axes: 1/2, 2/3, 3/4, 4/5, 5/6, 8/9, 1/3, 2/5, 3/5, 3/7, 4/9, 7/10, 1/4, 2/7, 3/8, 1/8.

**Return Values**

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointer is <code>NULL</code> .

<code>ippStsSizeErr</code>	Indicates an error condition if <i>srcRoiSize</i> or <i>dstRoiSize</i> has a field with zero or negative value, or if <i>dstRoiSize</i> has a field that is greater than the appropriate field of the <i>srcRoiSize</i> .
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> or <i>dstStep</i> is less than or equal to 0.

## SuperSamplingGetBufSize

*Computes the size of the external buffer for image reducing with the super sampling method.*

---

### Syntax

```
ippStatus ippSuperSamplingGetBufSize(IppiSize srcRoiSize, IppiSize dstRoiSize, int
nChannel, int* pBufferSize);
```

### Parameters

<i>srcRoiSize</i>	Size of the source image ROI in pixels.
<i>dstRoiSize</i>	Size of the destination image ROI in pixels.
<i>nChannel</i>	Number of channels or planes, possible values are 1, 3, 4.
<i>pBufferSize</i>	Pointer to the size of the external buffer.

### Description

The function `ippSuperSamplingGetBufSize` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function computes the size *pBufferSize* of the external buffer that is required for the function `ippSuperSampling`.

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pBufferSize</i> pointer is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>srcRoiSize</i> or <i>dstRoiSize</i> has a field with zero or negative value, or if <i>dstRoiSize</i> has a field that is greater than the appropriate field of the <i>srcRoiSize</i> .
<code>ippStsNumChannelErr</code>	Indicates an error condition if <i>nChannel</i> has an illegal value.

## ResizeYUV422

*Changes a size of 4:2:2 two-channel image.*

---

### Syntax

```
ippStatus ippResizeYUV422_8u_C2R(const Ipp8u* pSrc, IppiSize srcSize, int srcStep, IppiRect
srcRoi, Ipp8u* pDst, int dstStep, IppiSize dstRoiSize, double xFactor, double yFactor,
int interpolation);
```

## Parameters

<i>pSrc</i>	Pointer to the source image data.								
<i>srcSize</i>	Size in pixels of the source image.								
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image buffer.								
<i>srcRoi</i>	Region of interest in the source image (of the <code>IppiRect</code> type).								
<i>pDst</i>	Pointer to the destination image ROI.								
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image buffer.								
<i>dstRoiSize</i>	Size of the destination ROI in pixels.								
<i>xFactor, yFactor</i>	Factors by which the <i>x</i> and <i>y</i> dimensions of the source ROI are changed. The factor value greater than 1 corresponds to increasing the image size in that dimension.								
<i>interpolation</i>	Specifies the <a href="#">interpolation</a> mode. Use one of the following values: <table data-bbox="505 856 1346 1041"> <tr> <td><code>IPPI_INTER_NN</code></td><td>nearest neighbor interpolation</td></tr> <tr> <td><code>IPPI_INTER_LINEAR</code></td><td>linear interpolation</td></tr> <tr> <td><code>IPPI_INTER_CUBIC</code></td><td>cubic interpolation</td></tr> <tr> <td><code>IPPI_INTER_SUPER</code></td><td>supersampling interpolation, cannot be applied for image enlarging</td></tr> </table>	<code>IPPI_INTER_NN</code>	nearest neighbor interpolation	<code>IPPI_INTER_LINEAR</code>	linear interpolation	<code>IPPI_INTER_CUBIC</code>	cubic interpolation	<code>IPPI_INTER_SUPER</code>	supersampling interpolation, cannot be applied for image enlarging
<code>IPPI_INTER_NN</code>	nearest neighbor interpolation								
<code>IPPI_INTER_LINEAR</code>	linear interpolation								
<code>IPPI_INTER_CUBIC</code>	cubic interpolation								
<code>IPPI_INTER_SUPER</code>	supersampling interpolation, cannot be applied for image enlarging								

## Description

The function `ippiResizeYUV422` is declared in the `ippi.h` file. It operates with ROI (see [ROI Processing in Geometric Transforms](#)).

This function resizes the source image ROI by *xFactor* in the *x* direction and *yFactor* in the *y* direction. The image size can be either reduced or increased in each direction, depending on the values of *xFactor*, *yFactor*. The result is resampled using the interpolation mode specified by the *interpolation* parameter, and written to the destination image ROI. Supersampling interpolation can be applied if both *xFactor* and *yFactor* are less than or equal to 1.

The source image is a two-channel image in 4:2:2 sampling format in color spaces with decoupled luminance and chrominance components, for example, YUV422 or YCrCb422.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>srcSize</i> or <i>dstRoiSize</i> has a field with zero or negative value; or if <i>dstRoiSize.width</i> is less than 2; or if width of the source image or the source ROI is odd; or if horizontal offset of the source ROI is odd.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pSrc</i> or <i>pDst</i> pointer is <code>NULL</code> .
<code>ippStsResizeNoOperationErr</code>	Indicates an error condition if one of the destination image dimensions is less than 1 pixel.

<code>ippStsResizeFactorErr</code>	Indicates an error condition if <i>xFactor</i> or <i>yFactor</i> is less than or equal to zero.
<code>ippStsInterpolationErr</code>	Indicates an error condition if <i>interpolation</i> has an illegal value.

## ResizeYUV420

*Changes a size of 4:2:0 two-plane image.*

---

### Syntax

```
ippStatus ippiResizeYUV420_8u_P2R(const Ipp8u* pSrcY, int srcYStep, const Ipp8u* pSrcUV,
int srcUVStep, IppiSize srcRoiSize, Ipp8u* pDstY, int dstYStep, Ipp8u* pDstUV, int
dstUVStep, IppiSize dstRoiSize, int interpolation, Ipp8u* pBuffer);
```

### Parameters

<i>pSrcY</i>	Pointer to the ROI in the luminance plane for a source image.
<i>srcYStep</i>	Distance in bytes between starts of consecutive lines in the luminance plane of the source image.
<i>pSrcUV</i>	Pointer to the ROI in the interleaved chrominance plane for a source image.
<i>srcUVStep</i>	Distance in bytes between starts of consecutive lines in the interleaved chrominance plane of the source image.
<i>srcRoiSize</i>	Size of the source image ROI in pixels.
<i>pDstY</i>	Pointer to the ROI in the luminance plane for the destination image.
<i>dstYStep</i>	Distance in bytes between starts of consecutive lines in the luminance plane of the destination image.
<i>pDstUV</i>	Pointer to the ROI in the interleaved chrominance plane for the destination image.
<i>dstUVStep</i>	Distance in bytes between starts of consecutive lines in the chrominance plane of a destination image.
<i>dstRoiSize</i>	Size of the destination ROI in pixels.
<i>interpolation</i>	Specifies the <a href="#">interpolation</a> mode. Use one of the following values: <div style="margin-left: 20px;"> <code>IPPI_INTER_LANCZOS</code> interpolation with Lanczos window  <code>IPPI_INTER_SUPER</code> supersampling interpolation, cannot be applied for image enlarging </div>
<i>pBuffer</i>	Pointer to the work buffer.

### Description

The function `ippiResizeYUV400` is declared in the `ippi.h` file. It operates with ROI (see [ROI Processing in Geometric Transforms](#)).

This function resizes the source image ROI to the destination image ROI using specified interpolation mode *interpolation*. The function requires the external buffer *pBuffer*, its size must be previously computed by calling the function `ippiResizeYUV420GetBufSize`.

The source image is a two-plane image in 4:2:0 sampling format in color spaces with decoupled luminance and chrominance components, for example, YUV420.

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if width or height of <code>srcRoiSize</code> or <code>dstRoiSize</code> is less than 2; or if width or height of the <code>srcRoiSize</code> is less than corresponding field of <code>dstRoiSize</code> for supersampling.
<code>ippStsInterpolationErr</code>	Indicates an error condition if <code>interpolation</code> has an illegal value.
<code>ippStsDoubleSize</code>	Indicates a warning if width or height of the <code>srcRoiSize</code> is odd.

## ResizeYUV420GetBufSize

*Computes the size of the external buffer for resizing a 4:2:0 two-plane image.*

### Syntax

```
ippStatus ippResizeYUV420GetBufSize(IppiSize srcRoiSize, IppiSize dstRoiSize, int
interpolation, int* pSize);
```

### Parameters

<code>srcRoiSize</code>	Size of the source image ROI in pixels.
<code>dstRoiSize</code>	Size of the destination ROI in pixels.
<code>interpolation</code>	Specifies the <a href="#">interpolation</a> mode. Use one of the following values: <code>IPPI_INTER_LANCZOS</code> interpolation with Lanczos window <code>IPPI_INTER_SUPER</code> supersampling interpolation, cannot be applied for image enlarging
<code>pSize</code>	Pointer to the size of the work buffer.

### Description

The function `ippResizeYUV400GetBufSize` is declared in the `ippi.h` file. It operates with ROI (see [ROI Processing in Geometric Transforms](#)).

This function computes the size of the external buffer that is required for the function `ippResizeYUV420`.

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if the <code>pSize</code> pointer is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if width or height of <code>srcRoiSize</code> or <code>dstRoiSize</code> is less than 2; or if width or height of the <code>srcRoiSize</code> is less than corresponding field of <code>dstRoiSize</code> for supersampling.
<code>ippStsInterpolationErr</code>	Indicates an error condition if <code>interpolation</code> has an illegal value.

`ippStsDoubleSize` Indicates a warning if width or height of the `srcRoiSize` is odd.

## ResizeFilterGetSize

*Calculates the size of the state structure for resizing filter.*

---

### Syntax

```
ippStatus ippiResizeFilterGetSize_8u_C1R(IppiSize srcRoiSize, IppiSize dstRoiSize,
ippiResizeFilterType filter, Ipp32u* pSize);
```

### Parameters

<code>srcRoiSize</code>	Size of the source image ROI in pixels.
<code>dstRoiSize</code>	Size of the destination image ROI in pixels.
<code>filter</code>	Type of filter used in resizing; possible values: <code>ippResizeFilterHann</code> , <code>ippResizeFilterLanczos</code> .
<code>pSize</code>	Pointer to the size (in bytes) of the state structure.

### Description

The function `ippiResizeFilterGetSize` is declared in the `ippi.h` file. It operates with ROI (see [ROI Processing in Geometric Transforms](#)).

This function calculates the size `pSize` of the state structure required for the function `ippiResizeFilter`. The type of filter is specified by the parameter `filter`.

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if the <code>pSize</code> pointer is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <code>srcRoiSize</code> or <code>dstRoiSize</code> has a field with zero or negative value.
<code>ippStsNotSupportedModeErr</code>	Indicates an error condition if <code>filter</code> has an invalid value.

## ResizeFilterInit

*Initializes the state structure for the resize filter.*

---

### Syntax

```
ippStatus ippiResizeFilterInit_8u_C1R(IppiResizeFilterState* pState, IppiSize srcRoiSize,
ippiSize dstRoiSize, IppiResizeFilterType filter);
```

### Parameters

<code>pState</code>	Pointer to the state structure for resize filter.
<code>srcRoiSize</code>	Size of the source image ROI in pixels.
<code>dstRoiSize</code>	Size of the destination image ROI in pixels.



*filter*                      Type of filter used in resizing; possible values:  
                                  `ippResizeFilterHann`, `ippResizeFilterLanczos`

## Description

The function `ippiResizeFilterInit` is declared in the `ippi.h` file. It operates with ROI (see [ROI Processing in Geometric Transforms](#)).

This function initializes the state structure *pState* for resizing filter used by the function `ippiResizeFilter`. The size of the structure must be computed by the function `ippiResizeFilterGetSize` beforehand. The type of filter is specified by the parameter *filter* and it must be the same as in the function `ippiResizeFilterGetSize`.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if the <i>pState</i> pointer is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <i>srcRoiSize</i> or <i>dstRoiSize</i> has a field with zero or negative value.
<code>ippStsNotSupportedModeErr</code>	Indicates an error condition if <i>filter</i> has an invalid value.

## ResizeFilter

Changes the size of an image using a generic filter.

## Syntax

```
ippiStatus ippResizeFilter_8u_C1R(const Ipp8u* pSrc, int srcStep, IppiSize srcRoiSize,
Ipp8u* pDst, int dstStep, IppiSize dstRoiSize, IppiResizeFilterState* pState);
```

## Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image buffer.
<i>srcRoiSize</i>	Size of the source image ROI in pixels.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image buffer.
<i>dstRoiSize</i>	Size of the destination image ROI in pixels.
<i>pState</i>	Pointer to the state structure for the resize filter.

## Description

The function `ippiResizeFilter` is declared in the `ippi.h` file. It operates with ROI (see [ROI Processing in Geometric Transforms](#)).

This function resizes the source image *pSrc* using the special generic filters. The state structure *pState* contains the parameters of filtering and must be initialized by the function `ippiResizeFilterInit` beforehand.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointers is <code>NULL</code> .
<code>ippStsStepErr</code>	Indicates an error condition if <code>srcStep</code> or <code>dstStep</code> has a zero or negative value.
<code>ippStsSizeErr</code>	Indicates an error condition if <code>srcRoiSize</code> or <code>dstRoiSize</code> has a field with zero or negative value.
<code>ippStsContextMatchErr</code>	Indicates an error condition if a pointer to an invalid state structure is passed.

## Mirror

*Mirrors an image about a horizontal or vertical axis, or both.*

---

### Syntax

#### Case 1: Not-in-place operation

```
IppStatus ippiMirror_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pDst,
int dstStep, IppiSize roiSize, IppiAxis flip);
```

Supported values for mod:

8u_C1R	16u_C1R	16s_C1R	32s_C1R	32f_C1R
8u_C3R	16u_C3R	16s_C3R	32s_C3R	32f_C3R
8u_C4R	16u_C4R	16s_C4R	32s_C4R	32f_C4R
8u_AC4R	16u_AC4R	16s_AC4R	32s_AC4R	32f_AC4R

#### Case 2: In-place operation

```
IppStatus ippiMirror_<mod>(const Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize,
IppiAxis flip);
```

Supported values for mod:

8u_C1IR	16u_C1IR	16s_C1IR	32s_C1IR	32f_C1IR
8u_C3IR	16u_C3IR	16s_C3IR	32s_C3IR	32f_C3IR
8u_C4IR	16u_C4IR	16s_C4IR	32s_C4IR	32f_C4IR
8u_AC4IR	16u_AC4IR	16s_AC4IR	32s_AC4IR	32f_AC4IR

## Parameters

*pSrc*                      Pointer to the source buffer.

<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image buffer.						
<i>pDst</i>	Pointer to the destination buffer.						
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image buffer.						
<i>pSrcDst</i>	Pointer to the source and destination buffer for the in-place operation.						
<i>srcDstStep</i>	Distance in bytes between starts of consecutive lines in the source and destination image buffer for the in-place operation.						
<i>roiSize</i>	Size of the source and destination ROI in pixels.						
<i>flip</i>	Specifies the axis to mirror the image about. Use the following values to specify the axes: <table data-bbox="505 751 1292 863"> <tr> <td><code>ippAxsHorizontal</code></td><td>for the horizontal axis</td></tr> <tr> <td><code>ippAxsVertical</code></td><td>for the vertical axis</td></tr> <tr> <td><code>ippAxsBoth</code></td><td>for both horizontal and vertical axes.</td></tr> </table>	<code>ippAxsHorizontal</code>	for the horizontal axis	<code>ippAxsVertical</code>	for the vertical axis	<code>ippAxsBoth</code>	for both horizontal and vertical axes.
<code>ippAxsHorizontal</code>	for the horizontal axis						
<code>ippAxsVertical</code>	for the vertical axis						
<code>ippAxsBoth</code>	for both horizontal and vertical axes.						

## Description

The function `ippiMirror` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)). This function mirrors the source image *pSrc* about a horizontal or vertical axis or both, depending on the *flip* value, and writes it to the destination image *pDst*.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value, or when one of the dimensions is equal to 1.
<code>ippStsMirrorFlipErr</code>	Indicates an error condition if <i>flip</i> has an illegal value.

## Example

The code example below shows how to use the function `ippiMirror_8u_C1R`.

```
Ipp8u src[8*4] = {1, 2, 3, 4, 8, 8, 8, 8,
```

```

1, 2, 3, 4, 8, 8, 8, 8,
1, 2, 3, 4, 8, 8, 8, 8,
1, 2, 3, 4, 8, 8, 8, 8};

```

```
Ipp8u dst[4*4];
```

```
IppiSize srcRoi = { 4, 4 };
```

```
ippiMirror_8u_C1R(src, 8, dst, 4, srcRoi, ippAxsBoth);
```

Result:

```

1 2 3 4 8 8 8 8
1 2 3 4 8 8 8 8    src
1 2 3 4 8 8 8 8
1 2 3 4 8 8 8 8

4 3 2 1
4 3 2 1
4 3 2 1    dst
4 3 2 1

```

## Remap

*Performs the look-up coordinate mapping of pixels of the source image.*

---

### Syntax

#### Case 1: Operation on pixel-order data

```

IppStatus ippiRemap_<mod>(const Ipp<datatype>* pSrc, IppiSize srcSize, int srcStep, IppiRect
srcRoi, const Ipp32f* pxMap, int xMapStep, const Ipp32f* pyMap, int yMapStep, Ipp<datatype>*
pDst, int dstStep, IppiSize dstRoiSize, int interpolation);

```

Supported values for mod:

```

8u_C1R   16u_C1R   16s_C1R   32f_C1R   64f_C1R

8u_C3R   16u_C3R   16s_C3R   32f_C3R   64f_C3R

8u_C4R   16u_C4R   16s_C4R   32f_C4R   64f_C4R

8u_AC4R  16u_AC4R  16s_AC4R  32f_AC4R  64f_AC4R

```

**Case 2: Operation on planar-order data**

```
IppStatus ippRemap_<mod>(const Ipp<datatype>* const pSrc[3], IppiSize srcSize, int srcStep,
IppiRect srcRoi, const Ipp32f* pxMap, int xMapStep, const Ipp32f* pyMap, int yMapStep,
Ipp<datatype>* const pDst[3], int dstStep, IppiSize dstRoiSize, int interpolation);
```

Supported values for mod:

8u\_P3R 16u\_P3R 16s\_P3R 32f\_P3R 64f\_P3R

```
IppStatus ippRemap_<mod>(const Ipp<datatype>* const pSrc[4], IppiSize srcSize, int srcStep,
IppiRect srcRoi, const Ipp32f* pxMap, int xMapStep, const Ipp32f* pyMap, int yMapStep,
Ipp<datatype>* const pDst[4], int dstStep, IppiSize dstRoiSize, int interpolation);
```

Supported values for mod:

8u\_P4R 16u\_P4R 16s\_P4R 32f\_P4R 64f\_P4R

**Parameters**

<i>pSrc</i>	Pointer to the source image origin. An array of separate pointers to each plane in case of data in planar format.
<i>srcSize</i>	Size in pixels of the source image.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image buffer.
<i>srcRoi</i>	Region of interest in the source image (of the <code>IppiRect</code> type).
<i>pxMap, pyMap</i>	Pointers to the starts of 2D buffers, containing tables of the <i>x</i> - and <i>y</i> -coordinates.
<i>xMapStep, yMapStep</i>	Steps in bytes through the buffers containing tables of the <i>x</i> - and <i>y</i> -coordinates.
<i>pDst</i>	Pointer to the destination image ROI. An array of separate pointers to ROI in each plane for planar image.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image buffer.
<i>dstRoiSize</i>	Size of the destination ROI in pixels.
<i>interpolation</i>	Specifies the <a href="#">interpolation</a> mode. Possible values are: IPPI_INTER_NN - nearest neighbor interpolation IPPI_INTER_LINEAR - linear interpolation IPPI_INTER_CUBIC - cubic interpolation IPPI_INTER_LANCZOS - interpolation with Lanczos window IPPI_INTER_CUBIC2P_CATMULLROM - Catmull-Rom spline the following flag is used additionally to the above modes: IPPI_SMOOTH_EDGE - edge smoothing

## Description

The function `ippiRemap` is declared in the `ippi.h` file. It operates with ROI (see [ROI Processing in Geometric Transforms](#)).

This function transforms the source image by remapping its pixels. Pixel remapping is performed using `pxMap` and `pyMap` buffers to look-up the coordinates of the source image pixel that is written to the target destination image pixel. The application has to supply these look-up tables. The remapping of the source pixels to the destination pixels is made according to the following formula:

$$dst\_pixel[i, j] = src\_pixel[pxMap[i, j], pyMap[i, j]]$$

where  $i, j$  are the  $x$ - and  $y$ -coordinates of the target destination image pixel `dst_pixel`;

`pxMap[i, j]` contains the  $x$ - coordinates of the source image pixels `src_pixel` that are written to `dst_pixel`;

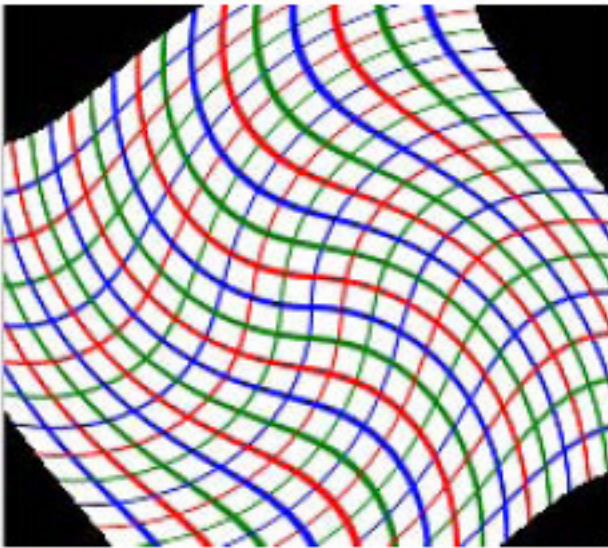
`pyMap[i, j]` contains the  $y$ - coordinates of the source image pixels `src_pixel` that are written to `dst_pixel`.

If the referenced coordinates correspond to a pixel outside of the source ROI, and the flag `IPPI_SMOOTH_EDGE` is not set, then no mapping of the source pixel is performed.

[Figure "Remapping the Sample Image"](#) gives an example of applying the function `ippiRemap` to a sample image that is a square grid of alternating blue, red, and green lines.

## Remapping the Sample Image

---



The transformed part of the image is resampled using the [interpolation method](#) specified by the `interpolation` parameter, and is written to the destination image ROI. The function can be used with or without edge smoothing. The pseudo code below shows how it works.

The function works without edge smoothing - the flag `IPPI_SMOOTH_EDGE` is not set:

```
if ( xMap < srcRoi . x || xMap > srcRoi . x + srcRoi . width -1 || yMap < srcRoi . y ||
yMap > srcRoi . y + srcRoi . height -1)
    not fill dst    /* do not remap */
```

```
else
```

```
    fill dst with Interpolate(Src, xMap, yMap)    /* remap */
```

The function works with edge smoothing - the flag `IPPI_SMOOTH_EDGE` is set:

```
if (xMap < srcRoi.x - 1 || xMap > srcRoi.x+srcRoi.width || yMap < srcRoi.y - 1 || yMap >
srcRoi.y+srcRoi.height)
```

```
    not fill dst    /* do not remap */
```

```
else if (xMap < srcRoi.x || xMap > srcRoi.x+srcRoi.width-1 || yMap < srcRoi.y || yMap >
srcRoi.y+srcRoi.height-1)
```

```
    fill dst with Interpolate(Src, fillvalue, xMap, yMap)    /* smoothing */
```

```
else
```

```
    fill dst with Interpolate(Src, xMap, yMap)    /* remap */
```

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <i>srcSize</i> or <i>dstRoiSize</i> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if one of the <i>srcStep</i> , <i>dstStep</i> , <i>xMapStep</i> , or <i>yMapStep</i> has a zero or negative value.
<code>ippStsInterpolationErr</code>	Indicates an error condition if <i>interpolation</i> has an illegal value.
<code>ippStsWrongIntersectROIErr</code>	Indicates an error condition if <i>srcRoi</i> has no intersection with the source image.

## Example

The code example below shows how to use the function `ippiRemap_8u_C1R`.

```
IppiSize size = { 4, 4 };
IppiRect srcRect = { 0, 0, 4, 4 };
Ipp8u src[4*4] = {0, 1, 2, 3,
```

```
    4, 5, 6, 7,  
    8, 9, 0, 1,  
    2, 3, 4, 5};  
Ipp8u dst[4*4];  
Ipp32f pxMap[16] = {0, 1, 2, 3,  
    0, 1, 2, 0,  
    0, 1, 0, 0,  
    0, 1, 2, 3};  
  
Ipp32f pyMap[16] = {0, 0, 0, 0,  
    1, 1, 1, 0,  
    2, 2, 0, 0,  
    3, 3, 3, 3}; // some order of pixels coordinates to write  
  
ippiRemap_8u_C1R ( src, size, 4, srcRect, pxMap, 16, pyMap, 16, dst, 4,  
    size, IPPI_INTER_NN);
```

Result:

```
0 1 2 3  
4 5 6 7  
8 9 0 1    src  
2 3 4 5
```

```
0 1 2 3  
4 5 6 0  
8 9 0 0    dst  
2 3 4 5
```



## Rotate

*Rotates an image around the origin (0,0) and then shifts it.*

---

### Syntax

#### Case 1: Operation on pixel-order data

```
IppStatus ippiRotate_<mod>(const Ipp<datatype>* pSrc, IppiSize srcSize, int srcStep,
IppiRect srcRoi, Ipp<datatype>* pDst, int dstStep, IppiRect dstRoi, double angle, double
xShift, double yShift, int interpolation);
```

Supported values for mod:

8u_C1R	16u_C1R	32f_C1R	64f_C1R
8u_C3R	16u_C3R	32f_C3R	64f_C3R
8u_C4R	16u_C4R	32f_C4R	64f_C4R
8u_AC4R	16u_AC4R	32f_AC4R	64f_AC4R

#### Case 2: Operation on planar-order data

```
IppStatus ippiRotate_<mod>(const Ipp<datatype>* const pSrc[3], IppiSize srcSize, int
srcStep, IppiRect srcRoi, Ipp<datatype>* const pDst[3], int dstStep, IppiRect dstRoi,
double angle, double xShift, double yShift, int interpolation);
```

Supported values for mod:

8u_P3R	16u_P3R	32f_P3R	64f_P3R
--------	---------	---------	---------

```
IppStatus ippiRotate_<mod>(const Ipp<datatype>* const pSrc[4], IppiSize srcSize, int
srcStep, IppiRect srcRoi, Ipp<datatype>* const pDst[4], int dstStep, IppiRect dstRoi,
double angle, double xShift, double yShift, int interpolation);
```

Supported values for mod:

8u_P4R	16u_P4R	32f_P4R	64f_P4R
--------	---------	---------	---------

### Parameters

<i>pSrc</i>	Pointer to the source image origin. An array of separate pointers to each plane for data in planar format.
<i>srcSize</i>	Size in pixels of the source image.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image buffer.
<i>srcRoi</i>	Region of interest in the source image (of the <code>IppiRect</code> type).

<i>pDst</i>	Pointer to the destination image origin. An array of separate pointers to each plane for data in planar format.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image buffer.
<i>dstRoi</i>	Region of interest in the destination image (of the <code>IppiRect</code> type).
<i>angle</i>	The angle of rotation in degrees. The source image is rotated counterclockwise around the origin (0,0).
<i>xShift, yShift</i>	The shifts along horizontal and vertical axes to perform after the rotation.
<i>interpolation</i>	Specifies the <a href="#">interpolation</a> mode. Possible values are: IPPI_INTER_NN - nearest neighbor interpolation IPPI_INTER_LINEAR - linear interpolation IPPI_INTER_CUBIC - cubic interpolation IPPI_INTER_CUBIC2P_CATMULLROM - Catmull-Rom spline the following flag is used additionally to the above modes: IPPI_SMOOTH_EDGE - edge smoothing

## Description

The function `ippiRotate` is declared in the `ippi.h` file. It operates with ROI (see [ROI Processing in Geometric Transforms](#)).

This function rotates the ROI of the source image by *angle* degrees (counterclockwise for positive *angle* values) around the origin (0,0) that is implied to be in the top left corner, and shifts it by *xShift* and *yShift* values along the *x*- and *y*- axes, respectively. The result is resampled using the [interpolation mode](#) specified by the *interpolation* parameter, and written to the destination image ROI.

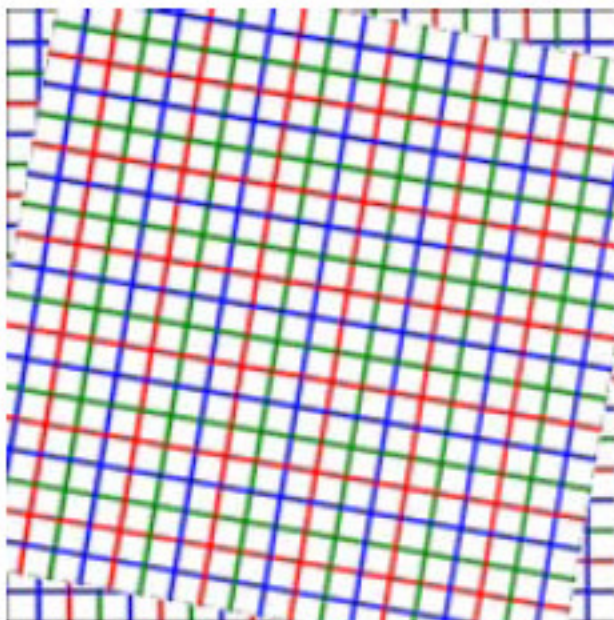
If you need to rotate an image about an arbitrary center (*xCenter*, *yCenter*), use the function `ippiGetRotateShift` to compute the appropriate *xShift*, *yShift* values, and then call `ippiRotate` with these values as input parameters.

Alternatively, you can use the `ippiRotateCenter` function instead.

Figure “Rotation of an Image” shows the result of rotating the sample image by 10 degrees.

### Rotation of an Image

---



The figures below illustrate how the `SMOOTH_EDGE` option affects the resulting destination image appearance. Both images are computed as a result of rotating the same source image by 9 degrees using the same [interpolation method](#), with the `SMOOTH_EDGE` option turned off and on, respectively. The use of this option has an antialiasing effect seen on the right border of image b) in [Figure “The Effect of Edge Smoothing in the Destination Image”](#).

### The Effect of Edge Smoothing in the Destination Image

---



a) `SMOOTH_EDGE` option turned off

b) `SMOOTH_EDGE` option turned on

[Example](#) shows how to use the function `ippiRotate_8u_C1R`.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if any image dimension has zero or negative value.
<code>ippStsInterpolationErr</code>	Indicates an error condition if <i>interpolation</i> has an illegal value.
<code>ippStsRectErr</code>	Indicates an error condition if width or height of the intersection of the <i>srcRoi</i> and source image is less than or equal to 1.
<code>ippStsWrongIntersectROIErr</code>	Indicates an error condition if <i>srcRoi</i> has no intersection with the source image.
<code>ippStsWrongIntersectQuad</code>	Indicates a warning that no operation is performed if the transformed source ROI has no intersection with the destination ROI.

## GetRotateShift

*Computes shift values for rotation of an image around the specified center.*

---

### Syntax

```
ippStatus ippGetRotateShift (double xCenter, double yCenter, double angle, double* xShift, double* yShift);
```

### Parameters

<code>xCenter, yCenter</code>	Coordinates of the required center of rotation.
<code>angle</code>	The angle in degrees to rotate the image clockwise around the point with coordinates ( <i>xCenter</i> , <i>yCenter</i> ).
<code>xShift, yShift</code>	Pointers to computed shift values along horizontal and vertical axes. These shift values should be passed to <code>ippiRotate</code> function to bring about the desired rotation around ( <i>xCenter</i> , <i>yCenter</i> ).

### Description

The function `ippiGetRotateShift` is declared in the `ippi.h` file. Use this function if you need to rotate an image about an arbitrary center (*xCenter*, *yCenter*) rather than the origin (0,0). The function helps compute shift values *xShift*, *yShift* that should be passed to `ippiRotate` function for the rotation around (*xCenter*, *yCenter*) to take place.

[Example](#) shows how to use the function `ippiGetRotateShift`.

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>xShift</i> or <i>yShift</i> pointer is <code>NULL</code> .

## AddRotateShift

*Computes shift values for rotating an image around the specified center with specified shifts.*

---

### Syntax

```
IppStatus ippiAddRotateShift(double xCenter, double yCenter, double angle, double* xShift, double* yShift);
```

### Parameters

<code>xCenter, yCenter</code>	Coordinates of the required center of rotation.
<code>angle</code>	The angle in degrees to rotate the image clockwise around the point with coordinates ( <code>xCenter, yCenter</code> ).
<code>xShift, yShift</code>	Pointers to modified shift values along horizontal and vertical axes. These new shift values should be passed to <code>ippiRotate</code> function to bring about the desired rotation around ( <code>xCenter, yCenter</code> ) and shifting.

### Description

The function `ippiAddRotateShift` is declared in the `ippi.h` file. Use this function if you need to rotate an image about an arbitrary center (`xCenter, yCenter`) rather than the origin (0,0) with required shifts. The function helps compute shift values `xShift, yShift` that should be passed to the `ippiRotate` function to perform the rotation around (`xCenter, yCenter`) and desired shifting. The shift values should be initialized. For example, to rotate an image around a point (`xCenter, yCenter`) by the `angle` with shift values (30.3, 26.2) the following code must be written:

```
xShift = 30.3 yShift = 26.2 ippiAddRotateShift(xCenter,yCenter,angle,&xShift,&yShift);
ippiRotate(angle,xShift,yShift);
```

**Example** shows how to use the function `ippiAddRotateShift`.

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <code>xShift</code> or <code>yShift</code> pointer is <code>NULL</code> .

## GetRotateQuad

*Computes vertex coordinates of the quadrangle, to which the source ROI is mapped by the `ippiRotate` function.*

---

### Syntax

```
IppStatus ippiGetRotateQuad(IppiRect srcRoi, double quad[4][2], double angle, double xShift, double yShift);
```

### Parameters

<code>srcRoi</code>	Source image ROI.
---------------------	-------------------

<i>quad</i>	Output array. Contains vertex coordinates of the quadrangle, to which the source ROI is mapped by <code>ippiRotate</code> function.
<i>angle</i>	The angle of rotation in degrees.
<i>xShift, yShift</i>	The shifts along horizontal and vertical axes to perform after the rotation.

## Description

The function `ippiGetRotateQuad` is declared in the `ippi.h` file. It operates with ROI (see [ROI Processing in Geometric Transforms](#)).

This function is used as a support function for `ippiRotate`. It computes vertex coordinates of the quadrangle, to which the source rectangular ROI is mapped by the `ippiRotate` function that rotates an image by *angle* degrees and shifts it by *xShift, yShift*.

The first dimension [4] of the array *quad* [4][2] is equal to the number of vertices, and the second dimension [2] means *x* and *y* coordinates of the vertex. Quadrangle vertices have the following meaning:

*quad*[0] corresponds to the transformed top-left corner of the source ROI,  
*quad*[1] corresponds to the transformed top-right corner of the source ROI,  
*quad*[2] corresponds to the transformed bottom-right corner of the source ROI,  
*quad*[3] corresponds to the transformed bottom-left corner of the source ROI.

[Example](#) shows how to use the function `ippiGetRotateQuad`.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>srcRoi</i> has a size field with zero or negative value.

## GetRotateBound

*Computes the bounding rectangle for the source ROI transformed by the `ippiRotate` function.*

---

### Syntax

```
IppStatus ippiGetRotateBound(IppiRect srcRoi, double bound[2][2], double angle, double xShift, double yShift);
```

### Parameters

<i>srcRoi</i>	Source image ROI.
<i>bound</i>	Output array. Contains vertex coordinates of the bounding rectangle for the transformed source ROI.
<i>angle</i>	The angle of rotation in degrees.
<i>xShift, yShift</i>	The shifts along horizontal and vertical axes to perform after the rotation.

## Description

The function `ippiGetRotateBound` is declared in the `ippi.h` file. It operates with ROI (see [ROI Processing in Geometric Transforms](#)).

This function is used as a support function for `ippiRotate`. It computes vertex coordinates of the smallest bounding rectangle for the quadrangle *quad*, to which the source ROI is mapped by the `ippiRotate` function, which rotates an image by *angle* degrees and shifts it by *xShift*, *yShift*. *bound[0]* specifies *x*, *y* coordinates of the top-left corner, *bound[1]* specifies *x*, *y* coordinates of the bottom-right corner.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>srcRoi</i> has a size field with zero or negative value.

## Example

The code example below shows how to use the function `ippiGetRotateBound`.

```
Ipp8u src[4*4] = {4, 4, 4, 4,
                  3, 3, 3, 3,
                  2, 2, 2, 2,
                  1, 1, 1, 1};

Ipp8u dst[6*6];

IppiSize srcSize = { 4, 4 };
IppiSize dstSize = { 6, 6 };
IppiRect srcRect = { 0, 0, 4, 4 };

double xShift;
double yShift;

double xCenterSrc = 2.0;
double yCenterSrc = 2.0;
double xCenterDst = 3.5;
double yCenterDst = 2.5;

double quad[4][2];
double bound[2][2];

double angle = 35.0;

ippiGetRotateShift ( xCenterSrc, yCenterSrc, angle, &xShift, &yShift );
// xShift = -0.7 , yShift = 1.5

ippiAddRotateShift ( 1.0, 1.0, angle, &xShift, &yShift);
// xShift = -1.1, yShift = 2.2

xShift += xCenterDst - xCenterSrc;
yShift += yCenterDst - yCenterSrc;
// xShift = 0.3, yShift = 2.7

ippiGetRotateQuad ( srcRect, quad, angle, xShift, yShift);
// quad[0] = { 0.3, 2.7 }
// quad[1] = { 2.7, 1.0 }
// quad[2] = { 4.5, 3.5 }
// quad[3] = { 2.0, 5.2 }
```



```

ippiGetRotateBound ( srcRect, bound, angle, xShift, yShift);
// bound[0] = { 0.3, 1.0 }
// bound[1] = { 4.5, 5.2 }

IppiRect dstRect = { (int)quad[1][0], (int)quad[1][1], (int)quad[3][0], (int)quad[3][1] };

ippiSet_8u_C1R ( 0, dst, 6, dstSize );
ippiRotate_8u_C1R ( src, srcSize, 4, srcRect, dst, 6, dstRect, angle, xShift, yShift, IPPI_INTER_NN);

```

Result:

```

4 4 4 4
3 3 3 3
2 2 2 2      src
1 1 1 1

0 0 0 0 0 0
0 0 0 0 0 0
0 0 4 3 0 0
0 0 3 2 0 0      dst
0 0 2 1 0 0
0 0 1 0 0 0

```

## RotateCenter

*Rotates an image about an arbitrary center.*

### Syntax

#### Case 1: Operation on pixel-order data

```

IppStatus ippiRotateCenter_<mod>(const Ipp<datatype>* pSrc, IppiSize srcSize, int srcStep,
IppiRect srcRoi, Ipp<datatype>* pDst, int dstStep, IppiRect dstRoi, double angle, double
xCenter, double yCenter, int interpolation);

```

Supported values for mod:

8u_C1R	16u_C1R	32f_C1R	64f_C1R
8u_C3R	16u_C3R	32f_C3R	64f_C3R

```
8u_C4R      16u_C4R      32f_C4R      64f_C4R
8u_AC4R     16u_AC4R     32f_AC4R     64f_AC4R
```

**Case 2: Operation on planar-order data**

```
IppStatus ippiRotateCenter_<mod>(const Ipp<datatype>* const pSrc[3], IppiSize srcSize,
int srcStep, IppiRect srcRoi, Ipp<datatype>* const pDst[3], int dstStep, IppiRect dstRoi,
double angle, double xCenter, double yCenter, int interpolation);
```

Supported values for mod:

```
8u_P3R      16u_P3R      32f_P3R      64f_P3R
```

```
IppStatus ippiRotateCenter_<mod>(const Ipp<datatype>* const pSrc[4], IppiSize srcSize,
int srcStep, IppiRect srcRoi, Ipp<datatype>* const pDst[4], int dstStep, IppiRect dstRoi,
double angle, double xCenter, double yCenter, int interpolation);
```

Supported values for mod:

```
8u_P4R      16u_P4R      32f_P4R      64f_P4R
```

**Parameters**

<i>pSrc</i>	Pointer to the source image origin. An array of separate pointers to each plane for data in planar format.
<i>srcSize</i>	Size in pixels of the source image.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image buffer.
<i>srcRoi</i>	Region of interest in the source image (of the <code>IppiRect</code> type).
<i>pDst</i>	Pointer to the destination image origin. An array of separate pointers to each plane for data in planar format.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image buffer.
<i>dstRoi</i>	Region of interest in the destination image (of the <code>IppiRect</code> type).
<i>angle</i>	The angle of rotation in degrees.
<i>xCenter, yCenter</i>	Center of rotation coordinates.
<i>interpolation</i>	Specifies the <a href="#">interpolation</a> mode. Possible values are: IPPI_INTER_NN - nearest neighbor interpolation IPPI_INTER_LINEAR - linear interpolation IPPI_INTER_CUBIC - cubic interpolation IPPI_INTER_CUBIC2P_CATMULLROM - Catmull-Rom spline the following flag is used additionally to the above modes: IPPI_SMOOTH_EDGE - edge smoothing

## Description

The function `ippiRotateCenter` is declared in the `ippi.h` file. It operates with ROI (see [ROI Processing in Geometric Transforms](#)).

This function rotates the ROI of the source image by *angle* degrees (counterclockwise for positive *angle* values) around the point with coordinates *xCenter*, *yCenter*. The origin of the source image is implied to be in the top left corner. The result is resampled using the [interpolation mode](#) specified by the *interpolation* parameter, and written to the destination image ROI.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if any image dimension has zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> or <i>dstStep</i> has a zero or negative value.
<code>ippStsInterpolationErr</code>	Indicates an error condition if <i>interpolation</i> has an illegal value.
<code>ippStsRectErr</code>	Indicates an error condition if width or height of the intersection of the <i>srcRoi</i> and source image is less than or equal to 1.
<code>ippStsWrongIntersectQuad</code>	Indicates a warning that no operation is performed if the transformed source ROI has no intersection with the destination ROI.

### Example

The code example below shows how to use the function `ippiRotateCenter_8u_C1R`.

```

IppiSize srcSize = { 4, 4 }, dstSize = { 6, 6 };
IppiRect srcRect = { 0, 0, 4, 4 };
IppiRect dstRect = { 0, 0, 6, 6 };
Ipp8u src[4*4] = {4, 4, 4, 4,
                  3, 3, 3, 3,
                  2, 2, 2, 2,
                  1, 1, 1, 1};
Ipp8u dst[6*6];
double angle = -45.0;
double xCenter = 1.0;
double yCenter = 4.0;

ippiSet_8u_C1R ( 0, dst, 6, dstSize);

ippiRotateCenter_8u_C1R (src, srcSize, 4, srcRect, dst, 6, dstRect, angle,
                        xCenter, yCenter, IPPI_INTER_NN);

```

Result:

```

4 4 4 4
3 3 3 3
2 2 2 2   src
1 1 1 1

0 0 0 0 0 0
0 0 0 4 0 0
0 0 2 3 4 0
0 0 1 2 3 0   dst
0 0 0 1 0 0
0 0 0 0 0 0

```

## Shear

*Performs shearing transform of the source image.*

### Syntax

#### Case 1: Operation on pixel-order data

```
IppStatus ippiShear_<mod>(const Ipp<datatype>* pSrc, IppiSize srcSize, int srcStep, IppiRect srcRoi, Ipp<datatype>* pDst, int dstStep, IppiRect dstRoi, double xShear, double yShear, double xShift, double yShift, int interpolation);
```

Supported values for mod:

8u_C1R	16u_C1R	32f_C1R	64f_C1R
8u_C3R	16u_C3R	32f_C3R	64f_C3R
8u_C4R	16u_C4R	32f_C4R	64f_C4R
8u_AC4R	16u_AC4R	32f_AC4R	64f_AC4R

#### Case 2: Operation on planar-order data

```
IppStatus ippiShear_<mod>(const Ipp<datatype>* const pSrc[3], IppiSize srcSize, int srcStep, IppiRect srcRoi, Ipp<datatype>* const pDst[3], int dstStep, IppiRect dstRoi, double xShear, double yShear, double xShift, double yShift, int interpolation);
```

Supported values for mod:

8u_P3R	16u_P3R	32f_P3R	64f_P3R
--------	---------	---------	---------

```
IppStatus ippiShear_<mod>(const Ipp<datatype>* const pSrc[4], IppiSize srcSize, int srcStep, IppiRect srcRoi, Ipp<datatype>* const pDst[4], int dstStep, IppiRect dstRoi, double xShear, double yShear, double xShift, double yShift, int interpolation);
```

Supported values for mod:

8u_P4R	16u_P4R	32f_P4R	64f_P4R
--------	---------	---------	---------

### Parameters

<i>pSrc</i>	Pointer to the source image origin. An array of separate pointers to each plane for data in planar format.
<i>srcSize</i>	Size in pixels of the source image.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image buffer.
<i>srcRoi</i>	Region of interest in the source image (of the <code>IppiRect</code> type).
<i>pDst</i>	Pointer to the destination image origin. An array of separate pointers to each plane for data in planar format.

<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image buffer.
<i>dstRoi</i>	Region of interest in the destination image (of the <code>IppiRect</code> type).
<i>xShear, yShear</i>	Coefficients of the shearing transform.
<i>xShift, yShift</i>	Additional shift values along the horizontal and vertical axes.
<i>interpolation</i>	Specifies the <a href="#">interpolation</a> mode. Possible values are: IPPI_INTER_NN - nearest neighbor interpolation IPPI_INTER_LINEAR - linear interpolation IPPI_INTER_CUBIC - cubic interpolation IPPI_INTER_CUBIC2P_CATMULLROM - Catmull-Rom spline the following flag is used additionally to the above modes: IPPI_SMOOTH_EDGE - edge smoothing

## Description

The function `ippiShear` is declared in the `ippi.h` file. It operates with ROI (see [ROI Processing in Geometric Transforms](#)).

This function performs shearing transform - it maps the source image pixel coordinates  $(x, y)$  according to the following formulas:

$$x' = x + xShear * y + xShift$$

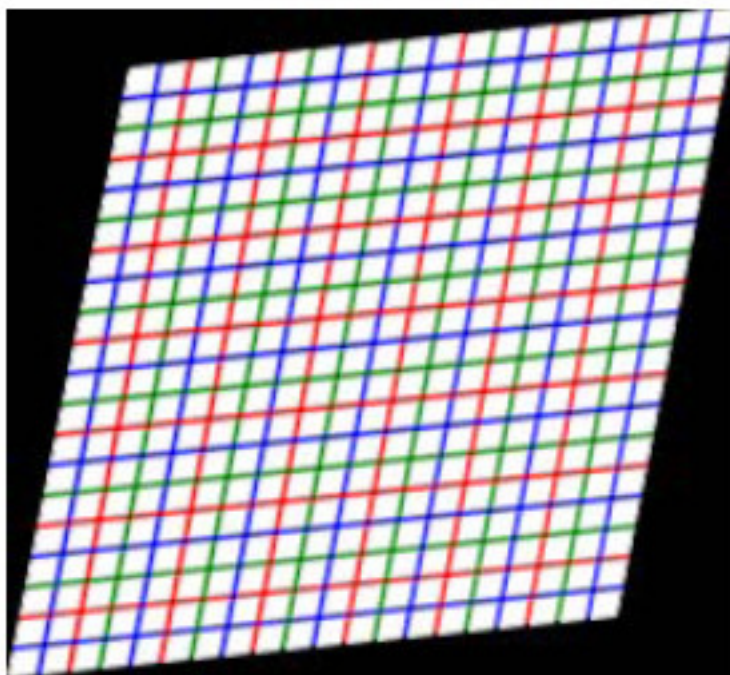
$$y' = yShear * x + y + yShift ,$$

where  $x'$  and  $y'$  denote the pixel coordinates in the sheared image. The shearing transform is a special case of an affine transform, performed by the [ippiWarpAffine](#) function.

The transformed part of the image is resampled using the [interpolation mode](#) specified by the *interpolation* parameter, and written to the destination image ROI.

Figure “Shearing a Sample Image” gives an example of applying the shearing transform function `ippiShear` to a sample image.

### Shearing a Sample Image



### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if any image dimension has zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <code>srcStep</code> or <code>dstStep</code> has a zero or negative value.
<code>ippStsInterpolationErr</code>	Indicates an error condition if <code>interpolation</code> has an illegal value.
<code>ippStsCoeffErr</code>	Indicates an error condition if $xShear * yShear = 1$ .
<code>ippStsRectErr</code>	Indicates an error condition if width or height of the intersection of the <code>srcRoi</code> and source image is less than or equal to 1.
<code>ippStsWrongIntersectROIErr</code>	Indicates an error condition if <code>srcRoi</code> has no intersection with the source image.
<code>ippStsWrongIntersectQuad</code>	Indicates a warning that no operation is performed if the transformed source ROI has no intersection with the destination ROI.

**Example**

The code example below shows how to use the function `ippiShear_8u_C1R`.

```

IppiSize srcSize = { 4, 4 };
IppiSize dstSize = { 7, 4 };
IppiRect srcRect = { 0, 0, 4, 4 };
IppiRect dstRect = { 0, 0, 7, 4 };
Ipp8u    src[4*4] = { 4, 4, 4, 4,
                      3, 3, 3, 3,
                      2, 2, 2, 2,
                      1, 1, 1, 1 };
Ipp8u    dst[7*4];
double xShear = -1.0;
double yShear = 0.0;
double xShift = 3.0;
double yShift = 0.0;
ippiSet_8u_C1R (0, dst, 7, dstSize);
ippiShear_8u_C1R (src,srcSize, 4, srcRect, dst, 7, dstRect, xShear,
                  yShear, xShift, yShift, IPPI_INTER_NN );

```

result:

```

4 4 4 4
3 3 3 3   src
2 2 2 2
1 1 1 1

0 0 0 4 4 4 4
0 0 3 3 3 3 0
0 2 2 2 2 0 0   dst
1 1 1 1 0 0 0

```



## GetShearQuad

*Computes vertex coordinates of the quadrangle, to which the source ROI rectangle is mapped by the shearing transform.*

---

### Syntax

```
IppStatus ippiGetShearQuad(IppiRect srcRoi, double quad[4][2], double xShear, double yShear, double xShift, double yShift);
```

### Parameters

<code>srcRoi</code>	Region of interest in the source image (of the <code>IppiRect</code> type).
<code>quad</code>	Output array. Contains vertex coordinates of the quadrangle, to which the source ROI is mapped by the shearing transform function.
<code>xShear, yShear</code>	Shearing transform coefficients.
<code>xShift, yShift</code>	Additional shift values along the horizontal and vertical axes.

### Description

The function `ippiGetShearQuad` is declared in the `ippi.h` file. It operates with ROI (see [ROI Processing in Geometric Transforms](#)).

This function is used as a support function for `ippiShear`. It computes vertex coordinates of the quadrangle, to which the source rectangular ROI is mapped by the shearing transform function `ippiShear` using coefficients `xShear, yShear` and shift values `xShift, yShift`.

The first dimension [4] of the array `quad[4][2]` is equal to the number of vertices, and the second dimension [2] means *x* and *y* coordinates of the vertex. Quadrangle vertices have the following meaning:

- `quad[0]` corresponds to the transformed top-left corner of the source ROI,
- `quad[1]` corresponds to the transformed top-right corner of the source ROI,
- `quad[2]` corresponds to the transformed bottom-right corner of the source ROI,
- `quad[3]` corresponds to the transformed bottom-left corner of the source ROI.

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
<code>ippStsCoeffErr</code>	Indicates an error condition if $xShear * yShear = 1$ .
<code>ippStsSizeErr</code>	Indicates an error condition if <code>srcRoi</code> has a size field with zero or negative value.

## GetShearBound

*Computes the bounding rectangle for the source ROI transformed by the `ippiShear` function.*

---

### Syntax

```
IppStatus ippiGetShearBound(IppiRect srcRoi, double bound[2][2], double xShear, double yShear, double xShift, double yShift);
```

### Parameters

<code>srcRoi</code>	Region of interest in the source image (of the <code>IppiRect</code> type).
<code>bound</code>	Output array. Contains vertex coordinates of the bounding rectangle for the transformed source ROI.
<code>xShear, yShear</code>	Shearing transform coefficients.
<code>xShift, yShift</code>	Additional shift values along the horizontal and vertical axes.

### Description

The function `ippiGetShearBound` is declared in the `ippi.h` file. It operates with ROI (see [ROI Processing in Geometric Transforms](#)).

This function is used as a support function for `ippiShear`. It computes vertex coordinates of the smallest bounding rectangle for the quadrangle *quad*, to which the source ROI is mapped by the shearing transform function `ippiShear` using coefficients `xShear, yShear` and shift values `xShift, yShift`.

`bound[0]` specifies *x, y* coordinates of the top-left corner, `bound[1]` specifies *x, y* coordinates of the bottom-right corner.

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsCoeffErr</code>	Indicates an error condition if $xShear * yShear = 1$ .
<code>ippStsSizeErr</code>	Indicates an error condition if <code>srcRoi</code> has a size field with zero or negative value.

## WarpAffine

Performs general affine transform of the source image.

### Syntax

#### Case 1: Operation on pixel-order data

```
IppStatus ippiWarpAffine_<mod>(const Ipp<datatype>* pSrc, IppiSize srcSize, int srcStep,
IppiRect srcRoi, Ipp<datatype>* pDst, int dstStep, IppiRect dstRoi, const double
coeffs[2][3], int interpolation);
```

Supported values for mod:

8u_C1R	16u_C1R	32f_C1R	64f_C1R
8u_C3R	16u_C3R	32f_C3R	64f_C3R
8u_C4R	16u_C4R	32f_C4R	64f_C4R
8u_AC4R	16u_AC4R	32f_AC4R	64f_AC4R

#### Case 2: Operation on planar-order data

```
IppStatus ippiWarpAffine_<mod>(const Ipp<datatype>* const pSrc[3], IppiSize srcSize, int
srcStep, IppiRect srcRoi, Ipp<datatype>* const pDst[3], int dstStep, IppiRect dstRoi,
const double coeffs[2][3], int interpolation);
```

Supported values for mod:

8u_P3R	16u_P3R	32f_P3R	64f_P3R
--------	---------	---------	---------

```
IppStatus ippiWarpAffine_<mod>(const Ipp<datatype>* const pSrc[4], IppiSize srcSize, int
srcStep, IppiRect srcRoi, Ipp<datatype>* const pDst[4], int dstStep, IppiRect dstRoi,
const double coeffs[2][3], int interpolation);
```

Supported values for mod:

8u_P4R	16u_P4R	32f_P4R	64f_P4R
--------	---------	---------	---------

### Parameters

<i>pSrc</i>	Pointer to the source image origin. An array of separate pointers to each plane in case of data in planar format.
<i>srcSize</i>	Size in pixels of the source image.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image buffer.
<i>srcRoi</i>	Region of interest in the source image (of the <code>IppiRect</code> type).
<i>pDst</i>	Pointer to the destination image origin. An array of separate pointers to each plane in case of data in planar format.

<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image buffer.
<i>dstRoi</i>	Region of interest in the destination image (of the <code>IppiRect</code> type).
<i>coeffs</i>	The affine transform coefficients.
<i>interpolation</i>	Specifies the <a href="#">interpolation</a> mode. Possible values are: <code>IPPI_INTER_NN</code> - nearest neighbor interpolation <code>IPPI_INTER_LINEAR</code> - linear interpolation <code>IPPI_INTER_CUBIC</code> - cubic interpolation <code>IPPI_INTER_CUBIC2P_CATMULLROM</code> - Catmull-Rom spline the following flag is used additionally to the above modes: <code>IPPI_SMOOTH_EDGE</code> - edge smoothing

## Description

The function `ippiWarpAffine` is declared in the `ippi.h` file. It operates with ROI (see [ROI Processing in Geometric Transforms](#)).

This affine warp function transforms the source image pixel coordinates  $(x, y)$  according to the following formulas:

$$x' = c_{00} * x + c_{01} * y + c_{02}$$

$$y' = c_{10} * x + c_{11} * y + c_{12}$$

where  $x'$  and  $y'$  denote the pixel coordinates in the transformed image, and  $c_{ij}$  are the affine transform coefficients passed in the array *coeffs*.

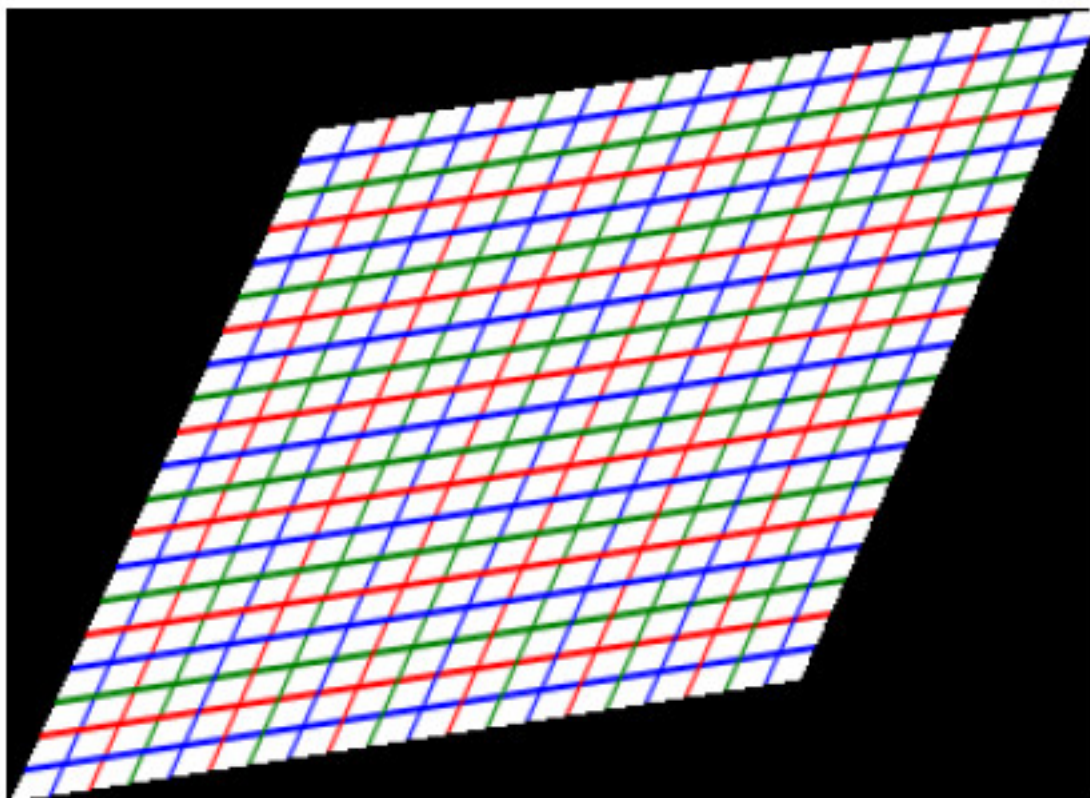
The affine warping is a general linear transform which incorporates such elementary transformations as scaling, rotation, translation, stretching, and shearing. It always transforms parallel lines into parallel lines and preserves equal distances between points on a line.

The transformed part of the image is resampled using the [interpolation mode](#) specified by the *interpolation* parameter, and written to the destination image ROI.

[Figure "Affine Transform of an Image"](#) gives an example of applying the affine warping function `ippiWarpAffine` to a sample image.

To compute the affine transform parameters, use the `ippiGetAffineQuad`, `ippiGetAffineBound`, and `ippiGetAffineTransform` functions.

### Affine Transform of an Image



**Example** shows how to use the function `ippiWarpAffine_32f_C1R`.

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if any image dimension has zero or negative value.
<code>ippStsInterpolationErr</code>	Indicates an error condition if <i>interpolation</i> has an illegal value.
<code>ippStsCoeffErr</code>	Indicates an error condition if $c_{00} * c_{11} - c_{01} * c_{10} = 0$ .
<code>ippStsRectErr</code>	Indicates an error condition if width or height of the intersection of the <i>srcRoi</i> and source image is less than or equal to 1.
<code>ippStsWrongIntersectROIErr</code>	Indicates an error condition if <i>srcRoi</i> has no intersection with the source image.

`ippStsWrongIntersectQuad` Indicates a warning that no operation is performed if the transformed source ROI has no intersection with the destination ROI.

## WarpAffineBack

*Performs an inverse affine transform of an image.*

---

### Syntax

#### Case 1: Operation on pixel-order data

```
IppStatus ippWarpAffineBack_<mod>(const Ipp<datatype>* pSrc, IppiSize srcSize, int srcStep,
IppiRect srcRoi, Ipp<datatype>* pDst, int dstStep, IppiRect dstRoi, const double
coeffs[2][3], int interpolation);
```

Supported values for mod:

8u_C1R	16u_C1R	32f_C1R
8u_C3R	16u_C3R	32f_C3R
8u_C4R	16u_C4R	32f_C4R
8u_AC4R	16u_AC4R	32f_AC4R

#### Case 2: Operation on planar-order data

```
IppStatus ippWarpAffineBack_<mod>(const Ipp<datatype>* const pSrc[3], IppiSize srcSize,
int srcStep, IppiRect srcRoi, Ipp<datatype>* const pDst[3], int dstStep, IppiRect dstRoi,
const double coeffs[2][3], int interpolation);
```

Supported values for mod:

8u_P3R	16u_P3R	32f_P3R
--------	---------	---------

```
IppStatus ippWarpAffineBack_<mod>(const Ipp<datatype>* const pSrc[4], IppiSize srcSize,
int srcStep, IppiRect srcRoi, Ipp<datatype>* const pDst[4], int dstStep, IppiRect dstRoi,
const double coeffs[2][3], int interpolation);
```

Supported values for mod:

8u_P4R	16u_P4R	32f_P4R
--------	---------	---------

### Parameters

<code>pSrc</code>	Pointer to the source image origin. An array of separate pointers to each plane in case of data in planar format.
<code>srcSize</code>	Size in pixels of the source image.

<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image buffer.						
<i>srcRoi</i>	Region of interest in the source image (of the <code>IppiRect</code> type).						
<i>pDst</i>	Pointer to the destination image origin. An array of separate pointers to each plane in case of data in planar format.						
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image buffer.						
<i>dstRoi</i>	Region of interest in the destination image (of the <code>IppiRect</code> type).						
<i>coeffs</i>	The affine transform coefficients.						
<i>interpolation</i>	Specifies the <a href="#">interpolation</a> mode. Use one of the following values: <table> <tr> <td><code>IPPI_INTER_NN</code></td><td>nearest neighbor interpolation</td></tr> <tr> <td><code>IPPI_INTER_LINEAR</code></td><td>linear interpolation</td></tr> <tr> <td><code>IPPI_INTER_CUBIC</code></td><td>cubic interpolation</td></tr> </table>	<code>IPPI_INTER_NN</code>	nearest neighbor interpolation	<code>IPPI_INTER_LINEAR</code>	linear interpolation	<code>IPPI_INTER_CUBIC</code>	cubic interpolation
<code>IPPI_INTER_NN</code>	nearest neighbor interpolation						
<code>IPPI_INTER_LINEAR</code>	linear interpolation						
<code>IPPI_INTER_CUBIC</code>	cubic interpolation						

## Description

The function `ippiWarpAffineBack` is declared in the `ippi.h` file. It operates with ROI (see [ROI Processing in Geometric Transforms](#)).

This function performs the inverse transform to that defined by `ippiWarpAffine` function. Pixel coordinates  $x'$  and  $y'$  in the transformed image are obtained from the following equations:

$$c_{00} * x' + c_{01} * y' + c_{02} = x$$

$$c_{10} * x' + c_{11} * y' + c_{12} = y$$

where  $x$  and  $y$  denote the pixel coordinates in the source image, and coefficients  $c_{ij}$  are given in the array `coeffs`. Thus, you do not need to invert transform coefficients in your application program before calling `ippiWarpAffineBack`.

Note that inverse transform functions handle source and destination ROI in a different way than other geometric transform functions. Their implementation include the following logic:

- backward transform is applied to coordinates of each pixel in the destination ROI, which gives as a result coordinates of some pixel in the source image
- if the obtained source pixel is inside source ROI, the corresponding pixel in destination ROI is modified accordingly; otherwise no change is made.

The above algorithm can be represented in pseudocode as follows:

```
for (j = dstRoi.y; j < dstRoi.y + dstRoi.height; j++) {
  for (i = dstRoi.x; i < dstRoi.x + dstRoi.width; i++) {
    sx = TransformX(i, j); sy = TransformY(i, j);
    if (sx >= srcRoi.x && sx < srcRoi.x + srcRoi.width && sy >= srcRoi.y && sy < srcRoi.y +
        srcRoi.height) {
      dst[i, j] = Interpolate(sx, sy); } } }
```

[Example](#) shows how to use the function `ippiWarpAffineBack_32f_C1R`.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if any image dimension has zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <code>srcStep</code> or <code>dstStep</code> has a zero or negative value.
<code>ippStsInterpolationErr</code>	Indicates an error condition if <code>interpolation</code> has an illegal value.
<code>ippStsCoeffErr</code>	Indicates an error condition if $c_{00} * c_{11} - c_{01} * c_{10} = 0$ .
<code>ippStsWrongIntersectROIErr</code>	Indicates an error condition if <code>srcRoi</code> has no intersection with the source image.
<code>ippStsWrongIntersectQuad</code>	Indicates a warning that no operation is performed if the transformed source ROI has no intersection with the destination ROI.

## WarpAffineQuad

*Performs image affine warping that transforms the given source quadrangle to the specified destination quadrangle.*

---

### Syntax

#### Case 1: Operation on pixel-order data

```
ippStatus ippkWarpAffineQuad_<mod>(const Ipp<datatype>* pSrc, IppiSize srcSize, int srcStep,
IppiRect srcRoi, const double srcQuad[4][2], Ipp<datatype>* pDst, int dstStep, IppiRect
dstRoi, const double dstQuad[4][2], int interpolation);
```

Supported values for `mod`:

8u_C1R	16u_C1R	32f_C1R
8u_C3R	16u_C3R	32f_C3R
8u_C4R	16u_C4R	32f_C4R
8u_AC4R	16u_AC4R	32f_AC4R

#### Case 2: Operation on planar-order data

```
ippStatus ippkWarpAffineQuad_<mod>(const Ipp<datatype>* const pSrc[3], IppiSize srcSize,
int srcStep, IppiRect srcRoi, const double srcQuad[4][2], Ipp<datatype>* const pDst[3],
int dstStep, IppiRect dstRoi, const double dstQuad[4][2], int interpolation);
```

Supported values for `mod`:

8u_P3R	16u_P3R	32f_P3R
--------	---------	---------



```
IppStatus ippiWarpAffineQuad_<mod>(const Ipp<datatype>* const pSrc[4], IppiSize srcSize,
int srcStep, IppiRect srcRoi, const double srcQuad[4][2], Ipp<datatype>* const pDst[4],
int dstStep, IppiRect dstRoi, const double dstQuad[4][2], int interpolation);
```

Supported values for mod:

8u\_P4R            16u\_P4R            32f\_P4R

## Parameters

<i>pSrc</i>	Pointer to the source image origin. An array of separate pointers to each plane in case of data in planar format.								
<i>srcSize</i>	Size in pixels of the source image.								
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image buffer.								
<i>srcRoi</i>	Region of interest in the source image (of the <code>IppiRect</code> type).								
<i>srcQuad</i>	A given quadrangle in the source image.								
<i>pDst</i>	Pointer to the destination image origin. An array of separate pointers to each plane in case of data in planar format.								
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image buffer.								
<i>dstRoi</i>	Region of interest in the destination image (of the <code>IppiRect</code> type).								
<i>dstQuad</i>	A given quadrangle in the destination image.								
<i>interpolation</i>	Specifies the <a href="#">interpolation</a> mode. Use one of the following: <table data-bbox="503 1176 1347 1362"> <tr> <td><code>IPPI_INTER_NN</code></td><td>nearest neighbor interpolation</td></tr> <tr> <td><code>IPPI_INTER_LINEAR</code></td><td>linear interpolation</td></tr> <tr> <td><code>IPPI_INTER_CUBIC</code></td><td>cubic interpolation</td></tr> <tr> <td><code>IPPI_SMOOTH_EDGE</code></td><td>use edge smoothing in addition to one of the above modes.</td></tr> </table>	<code>IPPI_INTER_NN</code>	nearest neighbor interpolation	<code>IPPI_INTER_LINEAR</code>	linear interpolation	<code>IPPI_INTER_CUBIC</code>	cubic interpolation	<code>IPPI_SMOOTH_EDGE</code>	use edge smoothing in addition to one of the above modes.
<code>IPPI_INTER_NN</code>	nearest neighbor interpolation								
<code>IPPI_INTER_LINEAR</code>	linear interpolation								
<code>IPPI_INTER_CUBIC</code>	cubic interpolation								
<code>IPPI_SMOOTH_EDGE</code>	use edge smoothing in addition to one of the above modes.								

## Description

The function `ippiWarpAffineQuad` is declared in the `ippi.h` file. It operates with ROI (see [ROI Processing in Geometric Transforms](#)).

This function applies the affine transform to an arbitrary quadrangle *srcQuad* in the source image *pSrc*. The operations take place only inside the intersection of the source image ROI *srcRoi* and the source quadrangle. The function `ippiWarpAffineQuad` uses the same formulas for pixel mapping as in the case of the [ippiWarpAffine](#) function. Transform coefficients are computed internally, based on the mapping of the source quadrangle to the quadrangle *dstQuad* specified in the destination image *pDst*. The *dstQuad* should have a non-empty intersection with the destination image ROI *dstRoi*. The function computes the coordinates of the 4th vertex of the destination quadrangle that uniquely depends on the three other vertices. If the computed coordinates are not equal to the ones specified in *dstQuad*, the function returns the warning message and continues operation with the computed values.

The first dimension [4] of the array specifying the quadrangle *srcQuad*[4][2] or *dstQuad*[4][2] is equal to the number of vertices, and the second dimension [2] holds x and y coordinates of the vertex.

Edge smoothing interpolation is applicable only if the source quadrangle lies in the source image ROI.

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if any image dimension has zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> or <i>dstStep</i> has a zero or negative value.
<code>ippStsInterpolationErr</code>	Indicates an error condition if <i>interpolation</i> has an illegal value.
<code>ippStsQuadErr</code>	Indicates an error condition if <i>srcQuad</i> or <i>dstQuad</i> degenerates into triangle, line, or point, or if <i>dstQuad</i> has wrong vertex coordinates.
<code>ippStsWrongIntersectROIErr</code>	Indicates an error condition if <i>srcRoi</i> has no intersection with the source image.
<code>ippStsWrongIntersectQuad</code>	Indicates a warning that no operation is performed if the <i>srcRoi</i> has no intersection with the <i>srcQuad</i> , or <i>dstRoi</i> has no intersection with the <i>dstQuad</i> .
<code>ippStsAffineQuadChanged</code>	Indicates a warning that coordinates of the 4th vertex of the destination quadrangle <i>dstQuad</i> are corrected by the function.

### GetAffineQuad

*Computes vertex coordinates of the quadrangle, to which the source ROI rectangle is mapped by the affine transform.*

---

#### Syntax

```
IppStatus ippiGetAffineQuad (IppiRect srcRoi , double quad [4][2], const double
coeffs[2][3]);
```

#### Parameters

<i>srcRoi</i>	Region of interest in the source image (of the <code>IppiRect</code> type).
<i>quad</i>	Output array. Contains vertex coordinates of the quadrangle, to which the source ROI is mapped by the affine transform function.
<i>coeffs</i>	The given affine transform coefficients.

#### Description

The function `ippiGetAffineQuad` is declared in the `ippi.h` file. It operates with ROI (see [ROI Processing in Geometric Transforms](#)).

This function is used as a support function for [ippiWarpAffine](#). It computes vertex coordinates of the quadrangle, to which the source rectangular ROI is mapped by the affine transform function [ippiWarpAffine](#) using the given coefficients *coeffs*.

The first dimension [4] of the array *quad*[4][2] is equal to the number of vertices, and the second dimension [2] means x and y coordinates of the vertex. Quadrangle vertices have the following meaning:

*quad*[0] corresponds to the transformed top-left corner of the source ROI,

*quad*[1] corresponds to the transformed top-right corner of the source ROI,

*quad*[2] corresponds to the transformed bottom-right corner of the source ROI,

*quad*[3] corresponds to the transformed bottom-left corner of the source ROI.

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
<code>ippStsCoeffErr</code>	Indicates an error condition if $c_{00} * c_{11} - c_{01} * c_{10} = 0$ .
<code>ippStsSizeErr</code>	Indicates an error condition if <i>srcRoi</i> has a size field with zero or negative value.

## GetAffineBound

*Computes the bounding rectangle for the source ROI transformed by the [ippiWarpAffine](#) function.*

### Syntax

```
ippiStatus ippiGetAffineBound (IppiRect srcRoi , double bound[2][2], const double coeffs[2][3]);
```

### Parameters

<i>srcRoi</i>	Region of interest in the source image (of the <code>IppiRect</code> type).
<i>bound</i>	Output array. Contains vertex coordinates of the bounding rectangle for the transformed source ROI.
<i>coeffs</i>	The given affine transform coefficients.

### Description

The function `ippiGetAffineBound` is declared in the `ippi.h` file. This function is used as a support function for [ippiWarpAffine](#). It computes vertex coordinates of the smallest bounding rectangle for the quadrangle *quad*, to which the source ROI is mapped by the affine transform function [ippiWarpAffine](#) using coefficients *coeffs*.

*bound*[0] specifies x, y coordinates of the top-left corner, *bound*[1] specifies x, y coordinates of the bottom-right corner.

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsCoeffErr</code>	Indicates an error condition if $c_{00} * c_{11} - c_{01} * c_{10} = 0$ .

<code>ippStsSizeErr</code>	Indicates an error condition if <code>srcRoi</code> has a size field with zero or negative value.
----------------------------	---

## GetAffineTransform

*Computes affine transform coefficients to map the source ROI to the quadrangle with the specified vertex coordinates*

---

### Syntax

```
IppStatus ippGetAffineTransform (IppiRect srcRoi , const double quad [4][2], double
coeffs[2][3]);
```

### Parameters

<code>srcRoi</code>	Region of interest in the source image (of the <code>IppiRect</code> type).
<code>quad</code>	Vertex coordinates of the quadrangle, to which the source ROI is mapped by the affine transform function.
<code>coeffs</code>	Output array. Contains the target affine transform coefficients.

### Description

The function `ippGetAffineTransform` is declared in the `ippi.h` file. It operates with ROI (see [ROI Processing in Geometric Transforms](#)).

This function is used as a support function for `ippiWarpAffine`. It computes the coefficients `coeffs` of the affine transform that must be used by the function `ippiWarpAffine` to map the source rectangular ROI to the quadrangle with the specified vertex coordinates `quad`. The first dimension [4] of the array `quad[4][2]` is equal to the number of vertices, and the second dimension [2] means x and y coordinates of the vertex. Quadrangle vertices have the following meaning:

`quad[0]` corresponds to the transformed top-left corner of the source ROI, `quad[1]` corresponds to the transformed top-right corner of the source ROI, `quad[2]` corresponds to the transformed bottom-right corner of the source ROI, `quad[3]` corresponds to the transformed bottom-left corner of the source ROI.

The function computes the coordinates of the 4th vertex of the destination quadrangle that uniquely depends on the three other vertices. If the computed coordinates are not equal to the ones specified in `quad`, the function returns the warning message and continues operation with the computed values.

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or warning.
<code>ippStsRectErr</code>	Indicates an error condition if width or height of the <code>srcRoi</code> is less than or equal to 1.
<code>ippStsCoeffErr</code>	Indicates an error condition if $c_{00} * c_{11} - c_{01} * c_{10} = 0$ .
<code>ippStsSizeErr</code>	Indicates an error condition if <code>srcRoi</code> has a size field with zero or negative value.
<code>ippStsAffineQuadChanged</code>	Indicates a warning that coordinates of the 4th vertex of the specified quadrangle <code>quad</code> are not correct.

## Example

The code example below shows how to use the function `ippiGetAffineTransform`.

```
void func_warp_affine()
{
    int step;

    Ipp32f pSrc[8*8] = {1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0,
                       1.0, 1.0, 1.0, 5.0, 5.0, 1.0, 1.0, 1.0,
                       1.0, 1.0, 5.0, 5.0, 5.0, 5.0, 1.0, 1.0,
                       1.0, 5.0, 5.0, 5.0, 5.0, 5.0, 5.0, 1.0,
                       1.0, 5.0, 5.0, 5.0, 5.0, 5.0, 5.0, 1.0,
                       1.0, 1.0, 5.0, 5.0, 5.0, 5.0, 1.0, 1.0,
                       1.0, 1.0, 1.0, 5.0, 5.0, 1.0, 1.0, 1.0,
                       1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0};

    int srcStep = 8*sizeof(Ipp32f);

    IppiSize srcSize = { 8, 8};
    IppiRect srcRoi = {0, 0, 8, 8};

    Ipp32f pDst[8*8];
    Ipp32f pDstB[8*8];
    int dstStep = 8*sizeof(ipp32f);
    int dstbStep = 8*sizeof(ipp32f);

    IppiRect dstRoi = {0, 0, 8, 8};
    double quad[4][2] = {{0.0, 0.0},
                        {7.0, 1.0},
                        {8.0, 8.0},
                        {1.0, 7.0}};

    double coeffs[2][3]; // affine transform coefficients

    ippiGetAffineTransform(srcRoi, quad, coeffs); //computes the affine transform coefficients

    ippiSet_32f_C1R(2,pDst,srcStep,srcSize);
    ippiWarpAffine_32f_C1R(pSrc, srcSize, srcStep, srcRoi, pDst, dstStep,
                           dstRoi, coeffs, IPPI_INTER_NN);
}
```

```
ippiSet_32f_C1R(3,pDstB,srcStep,srcSize);
ippiWarpAffineBack_32f_C1R( pDst, srcSize, dstStep, dstRoi, pDstB,  dstbStep,
                           dstRoi, coeffs, IPPI_INTER_NN);
}
```

Result:

pSrc

1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
1.0	1.0	1.0	5.0	5.0	1.0	1.0	1.0
1.0	1.0	5.0	5.0	5.0	5.0	1.0	1.0
1.0	5.0	5.0	5.0	5.0	5.0	5.0	1.0
1.0	5.0	5.0	5.0	5.0	5.0	5.0	1.0
1.0	1.0	5.0	5.0	5.0	5.0	1.0	1.0
1.0	1.0	1.0	5.0	5.0	1.0	1.0	1.0
1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0

pDst

1.0	2.0	2.0	2.0	2.0	2.0	2.0	2.0
2.0	1.0	1.0	5.0	1.0	1.0	1.0	1.0
2.0	1.0	5.0	5.0	5.0	1.0	1.0	1.0
2.0	5.0	5.0	5.0	5.0	5.0	1.0	1.0
2.0	1.0	5.0	5.0	5.0	5.0	5.0	1.0
2.0	1.0	1.0	5.0	5.0	5.0	5.0	5.0
2.0	1.0	1.0	1.0	5.0	5.0	5.0	1.0
2.0	1.0	1.0	1.0	1.0	5.0	1.0	1.0

pDstB

1.0	2.0	2.0	2.0	1.0	1.0	1.0	1.0
2.0	1.0	1.0	5.0	5.0	1.0	1.0	3.0
2.0	1.0	5.0	5.0	5.0	5.0	1.0	3.0
2.0	5.0	5.0	5.0	5.0	5.0	5.0	3.0
1.0	5.0	5.0	5.0	5.0	5.0	5.0	3.0
1.0	1.0	5.0	5.0	5.0	5.0	1.0	3.0

```
1.0  1.0  1.0  5.0  5.0  1.0  1.0  3.0
1.0  3.0  3.0  3.0  3.0  3.0  3.0  3.0
```

## WarpPerspective

*Performs perspective warping of the source image using the given transform coefficients.*

### Syntax

#### Case 1: Operation on pixel-order data

```
IppStatus ippiWarpPerspective_<mod>(const Ipp<datatype>* pSrc, IppiSize srcSize, int
srcStep, IppiRect srcRoi, Ipp<datatype>* pDst, int dstStep, IppiRect dstRoi, const double
coeffs[3][3], int interpolation);
```

Supported values for mod:

8u_C1R	16u_C1R	32f_C1R
8u_C3R	16u_C3R	32f_C3R
8u_C4R	16u_C4R	32f_C4R
8u_AC4R	16u_AC4R	32f_AC4R

#### Case 2: Operation on planar-order data

```
IppStatus ippiWarpPerspective_<mod>(const Ipp<datatype>* const pSrc[3], IppiSize srcSize,
int srcStep, IppiRect srcRoi, Ipp<datatype>* const pDst[3], int dstStep, IppiRect dstRoi,
const double coeffs[3][3], int interpolation);
```

Supported values for mod:

8u_P3R	16u_P3R	32f_P3R
--------	---------	---------

```
IppStatus ippiWarpPerspective_<mod>(const Ipp<datatype>* const pSrc[4], IppiSize srcSize,
int srcStep, IppiRect srcRoi, Ipp<datatype>* const pDst[4], int dstStep, IppiRect dstRoi,
const double coeffs[3][3], int interpolation);
```

Supported values for mod:

8u_P4R	16u_P4R	32f_P4R
--------	---------	---------

### Parameters

<i>pSrc</i>	Pointer to the source image origin. An array of separate pointers to each plane in case of data in planar format.
<i>srcSize</i>	Size in pixels of the source image.

<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image buffer.								
<i>srcRoi</i>	Region of interest in the source image (of the <code>IppiRect</code> type).								
<i>pDst</i>	Pointer to the destination image origin. An array of separate pointers to each plane in case of data in planar format.								
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image buffer.								
<i>dstRoi</i>	Region of interest in the destination image (of the <code>IppiRect</code> type).								
<i>coeffs</i>	The perspective transform coefficients.								
<i>interpolation</i>	Specifies the <a href="#">interpolation</a> mode. Use one of the following values: <table data-bbox="505 722 1346 905"> <tr> <td><code>IPPI_INTER_NN</code></td><td>nearest neighbor interpolation</td></tr> <tr> <td><code>IPPI_INTER_LINEAR</code></td><td>linear interpolation</td></tr> <tr> <td><code>IPPI_INTER_CUBIC</code></td><td>cubic interpolation</td></tr> <tr> <td><code>IPPI_SMOOTH_EDGE</code></td><td>use edge smoothing in addition to one of the above modes.</td></tr> </table>	<code>IPPI_INTER_NN</code>	nearest neighbor interpolation	<code>IPPI_INTER_LINEAR</code>	linear interpolation	<code>IPPI_INTER_CUBIC</code>	cubic interpolation	<code>IPPI_SMOOTH_EDGE</code>	use edge smoothing in addition to one of the above modes.
<code>IPPI_INTER_NN</code>	nearest neighbor interpolation								
<code>IPPI_INTER_LINEAR</code>	linear interpolation								
<code>IPPI_INTER_CUBIC</code>	cubic interpolation								
<code>IPPI_SMOOTH_EDGE</code>	use edge smoothing in addition to one of the above modes.								

## Description

The function `ippiWarpPerspective` is declared in the `ippi.h` file. It operates with ROI (see [ROI Processing in Geometric Transforms](#)).

This perspective warp function transforms the source image pixel coordinates  $(x, y)$  according to the following formulas:

$$x' = (c_{00} * x + c_{01} * y + c_{02}) / (c_{20} * x + c_{21} * y + c_{22})$$

$$y' = (c_{10} * x + c_{11} * y + c_{12}) / (c_{20} * x + c_{21} * y + c_{22})$$

where  $x'$  and  $y'$  denote the pixel coordinates in the transformed image, and  $c_{ij}$  are the perspective transform coefficients passed in the array *coeffs*.

The transformed part of the image is resampled using the [interpolation mode](#) specified by the *interpolation* parameter, and written to the destination image ROI.




---

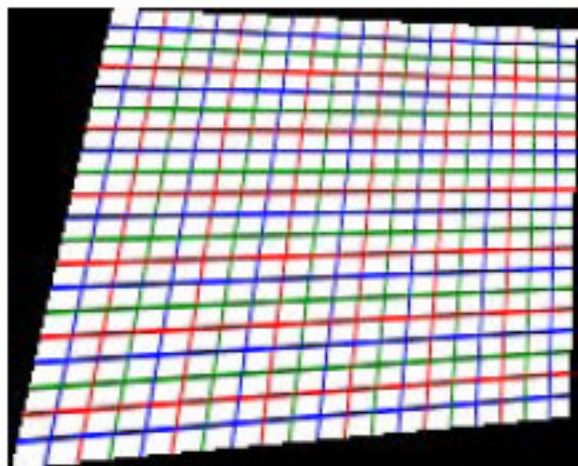
**NOTE.** `IPPI_SMOOTH_EDGE` is the artificial method. It is not universal: in some cases it can improve the image, but in some other cases it is not effective. For example, the smoothing does not improve the images with the high contrast borders. In such cases don't use edge smoothing.

---



Figure “Perspective Transform of an Image” gives an example of applying the perspective transform function `ippiWarpPerspective` to a sample image.

### Perspective Transform of an Image



To estimate how the source image ROI will be transformed by the `ippiWarpPerspective` function, use functions `ippiWarpPerspectiveQuad` and `ippiGetPerspectiveBound`. To calculate coefficients of the perspective transform that maps source ROI to a given quadrangle, use `ippiGetPerspectiveTransform` function.

**Example** shows how to use the function `ippiWarpPerspective_32f_C1R`.

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointer is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if any image dimension has zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <code>srcStep</code> or <code>dstStep</code> has a zero or negative value.
<code>ippStsRectErr</code>	Indicates an error condition if width or height of the intersection of the <code>srcRoi</code> and source image is less than or equal to 1.
<code>ippStsInterpolationErr</code>	Indicates an error condition if <code>interpolation</code> has an illegal value.
<code>ippStsCoeffErr</code>	Indicates an error condition if coefficient values are invalid.
<code>ippStsWrongIntersectROIErr</code>	Indicates an error condition if <code>srcRoi</code> has no intersection with the source image.
<code>ippStsWrongIntersectQuad</code>	Indicates a warning that no operation is performed if the transformed source ROI has no intersection with the destination ROI.

## WarpPerspectiveBack

*Performs an inverse perspective warping of the source image.*

---

### Syntax

#### Case 1: Operation on pixel-order data

```
IppStatus ippiWarpPerspectiveBack_<mod>(const Ipp<datatype>* pSrc, IppiSize srcSize, int srcStep, IppiRect srcRoi, Ipp<datatype>* pDst, int dstStep, IppiRect dstRoi, const double coeffs[3][3], int interpolation);
```

Supported values for mod:

8u_C1R	16u_C1R	32f_C1R
8u_C3R	16u_C3R	32f_C3R
8u_C4R	16u_C4R	32f_C4R
8u_AC4R	16u_AC4R	32f_AC4R

#### Case 2: Operation on planar-order data

```
IppStatus ippiWarpPerspectiveBack_<mod>(const Ipp<datatype>* const pSrc[3], IppiSize srcSize, int srcStep, IppiRect srcRoi, Ipp<datatype>* const pDst[3], int dstStep, IppiRect dstRoi, const double coeffs[3][3], int interpolation);
```

Supported values for mod:

8u_P3R	16u_P3R	32f_P3R
--------	---------	---------

```
IppStatus ippiWarpPerspectiveBack_<mod>(const Ipp<datatype>* const pSrc[4], IppiSize srcSize, int srcStep, IppiRect srcRoi, Ipp<datatype>* const pDst[4], int dstStep, IppiRect dstRoi, const double coeffs[3][3], int interpolation);
```

Supported values for mod:

8u_P4R	16u_P4R	32f_P4R
--------	---------	---------

### Parameters

<i>pSrc</i>	Pointer to the source image origin. An array of separate pointers to each plane in case of data in planar format.
<i>srcSize</i>	Size in pixels of the source image (of the <code>IppiRect</code> type).
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image buffer.
<i>srcRoi</i>	Region of interest in the source image.

<i>pDst</i>	Pointer to the destination image origin. An array of separate pointers to each plane in case of data in planar format.						
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image buffer.						
<i>dstRoi</i>	Region of interest in the destination image (of the <code>IppiRect</code> type).						
<i>coeffs</i>	The perspective transform coefficients.						
<i>interpolation</i>	Specifies the <a href="#">interpolation</a> mode. Use one of the following values: <table> <tr> <td><code>IPPI_INTER_NN</code></td><td>nearest neighbor interpolation</td></tr> <tr> <td><code>IPPI_INTER_LINEAR</code></td><td>linear interpolation</td></tr> <tr> <td><code>IPPI_INTER_CUBIC</code></td><td>cubic interpolation.</td></tr> </table>	<code>IPPI_INTER_NN</code>	nearest neighbor interpolation	<code>IPPI_INTER_LINEAR</code>	linear interpolation	<code>IPPI_INTER_CUBIC</code>	cubic interpolation.
<code>IPPI_INTER_NN</code>	nearest neighbor interpolation						
<code>IPPI_INTER_LINEAR</code>	linear interpolation						
<code>IPPI_INTER_CUBIC</code>	cubic interpolation.						

## Description

The function `ippiWarpPerspectiveBack` is declared in the `ippi.h` file. It operates with ROI (see [ROI Processing in Geometric Transforms](#)).

This function performs the inverse transform to that defined by `ippiWarpPerspective` function. Pixel coordinates  $x'$  and  $y'$  in the transformed image are obtained from the following equations

$$(c_{00} * x' + c_{01} * y' + c_{02}) / (c_{20} * x' + c_{21} * y' + c_{22}) = x$$

$$(c_{10} * x' + c_{11} * y' + c_{12}) / (c_{20} * x' + c_{21} * y' + c_{22}) = y$$

where  $x$  and  $y$  denote the pixel coordinates in the source image, and coefficients  $c_{ij}$  are given in the array `coeffs`. Thus, you don't need to invert transform coefficients in your application program before calling `ippiWarpPerspectiveBack`.

Note that inverse transform functions handle source and destination ROI in a different way than other geometric transform functions. See implementation details in the description of `ippiWarpAffineBack` function.

[Example](#) shows how to use the function `ippiWarpPerspectiveBack_32f_C1R`.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if any image dimension has zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <code>srcStep</code> or <code>dstStep</code> has a zero or negative value.
<code>ippStsInterpolationErr</code>	Indicates an error condition if <code>interpolation</code> has an illegal value.
<code>ippStsCoeffErr</code>	Indicates an error condition if coefficient values are invalid.
<code>ippStsWrongIntersectROIErr</code>	Indicates an error condition if <code>srcRoi</code> has no intersection with the source image.
<code>ippStsWrongIntersectQuad</code>	Indicates a warning that no operation is performed if the transformed source ROI has no intersection with the destination ROI.

## WarpPerspectiveQuad

*Performs perspective warping of the given source quadrangle to the specified destination quadrangle.*

---

### Syntax

#### Case 1: Operation on pixel-order data

```
IppStatus ippiWarpPerspectiveQuad_<mod>(const Ipp<datatype>* pSrc, IppiSize srcSize, int srcStep, IppiRect srcRoi, const double srcQuad[4][2], Ipp<datatype>* pDst, int dstStep, IppiRect dstRoi, const double dstQuad[4][2], int interpolation);
```

Supported values for mod:

8u_C1R	16u_C1R	32f_C1R
8u_C3R	16u_C3R	32f_C3R
8u_C4R	16u_C4R	32f_C4R
8u_AC4R	16u_AC4R	32f_AC4R

#### Case 2: Operation on planar-order data

```
IppStatus ippiWarpPerspectiveQuad_<mod>(const Ipp<datatype>* const pSrc[3], IppiSize srcSize, int srcStep, IppiRect srcRoi, const double srcQuad[4][2], Ipp<datatype>* const pDst[3], int dstStep, IppiRect dstRoi, const double dstQuad[4][2], int interpolation);
```

Supported values for mod:

8u_P3R	16u_P3R	32f_P3R
--------	---------	---------

```
IppStatus ippiWarpPerspectiveQuad_<mod>(const Ipp<datatype>* const pSrc[4], IppiSize srcSize, int srcStep, IppiRect srcRoi, const double srcQuad[4][2], Ipp<datatype>* const pDst[4], int dstStep, IppiRect dstRoi, const double dstQuad[4][2], int interpolation);
```

Supported values for mod:

8u_P4R	16u_P4R	32f_P4R
--------	---------	---------

### Parameters

<i>pSrc</i>	Pointer to the source image origin. An array of separate pointers to each plane in case of data in planar format.
<i>srcSize</i>	Size in pixels of the source image.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image buffer.
<i>srcRoi</i>	Region of interest in the source image (of the <code>IppiRect</code> type).
<i>srcQuad</i>	A given quadrangle in the source image.

<i>pDst</i>	Pointer to the destination image origin. An array of separate pointers to each plane in case of data in planar format.						
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image buffer.						
<i>dstRoi</i>	Region of interest in the destination image (of the <code>IppiRect</code> type).						
<i>dstQuad</i>	A given quadrangle in the destination image.						
<i>interpolation</i>	Specifies the <a href="#">interpolation</a> mode. Use one of the following values: <table data-bbox="505 611 1206 722"> <tr> <td><code>IPPI_INTER_NN</code></td><td>nearest neighbor interpolation</td></tr> <tr> <td><code>IPPI_INTER_LINEAR</code></td><td>linear interpolation</td></tr> <tr> <td><code>IPPI_INTER_CUBIC</code></td><td>cubic interpolation.</td></tr> </table>	<code>IPPI_INTER_NN</code>	nearest neighbor interpolation	<code>IPPI_INTER_LINEAR</code>	linear interpolation	<code>IPPI_INTER_CUBIC</code>	cubic interpolation.
<code>IPPI_INTER_NN</code>	nearest neighbor interpolation						
<code>IPPI_INTER_LINEAR</code>	linear interpolation						
<code>IPPI_INTER_CUBIC</code>	cubic interpolation.						

## Description

The function `ippiWarpPerspectiveQuad` is declared in the `ippi.h` file. It operates with ROI (see [ROI Processing in Geometric Transforms](#)).

This function applies a perspective transform to an arbitrary quadrangle *srcQuad* in the source image *pSrc*. The operations take place only in the intersection of the source image ROI *srcRoi* and the source quadrangle. The function `ippiWarpPerspectiveQuad` uses the same formulas for pixel mapping as in the case of the [ippiWarpPerspective](#) function. Transform coefficients are computed internally, based on the mapping of the source quadrangle to the quadrangle *dstQuad* specified in the destination image *pDst*. The *dstQuad* should have a non-empty intersection with the destination image ROI *dstRoi*.

The first dimension [4] of the array specifying the quadrangle *srcQuad*[4][2] or *dstQuad*[4][2] is equal to the number of vertices, and the second dimension [2] holds *x* and *y* coordinates of the vertex.

Edge smoothing interpolation is applicable only if the source quadrangle lies in the source image ROI.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if any image dimension has zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> or <i>dstStep</i> has a zero or negative value.
<code>ippStsInterpolationErr</code>	Indicates an error condition if <i>interpolation</i> has an illegal value.
<code>ippStsQuadErr</code>	Indicates an error condition if <i>srcQuad</i> or <i>dstQuad</i> degenerates into triangle.
<code>ippStsWrongIntersectROIErr</code>	Indicates an error condition if <i>srcRoi</i> has no intersection with the source image.
<code>ippStsWrongIntersectQuad</code>	Indicates a warning that no operation is performed if the <i>srcRoi</i> has no intersection with the <i>srcQuad</i> , or <i>dstRoi</i> has no intersection with the <i>dstQuad</i> .

## GetPerspectiveQuad

*Computes vertex coordinates of the quadrangle, to which the source ROI rectangle is mapped by the perspective transform.*

---

### Syntax

```
IppStatus ippiGetPerspectiveQuad(IppiRect srcRoi, double quad[4][2], const double coeffs[3][3]);
```

### Parameters

<i>srcRoi</i>	Region of interest in the source image (of the <code>IppiRect</code> type).
<i>quad</i>	Output array. Contains vertex coordinates of the quadrangle, to which the source ROI is mapped by the perspective transform function.
<i>coeffs</i>	The given perspective transform coefficients.

### Description

The function `ippiGetPerspectiveQuad` is declared in the `ippi.h` file. It operates with ROI (see [ROI Processing in Geometric Transforms](#)).

This function is used as a support function for [ippiWarpPerspective](#). It computes vertex coordinates of the quadrangle, to which the source rectangular ROI is mapped by the perspective transform function [ippiWarpPerspective](#) using the given coefficients *coeffs*.

The first dimension [4] of the array *quad*[4][2] is equal to the number of vertices, and the second dimension [2] means *x* and *y* coordinates of the vertex. Quadrangle vertices have the following meaning:

- quad*[0] corresponds to the transformed top-left corner of the source ROI,
- quad*[1] corresponds to the transformed top-right corner of the source ROI,
- quad*[2] corresponds to the transformed bottom-right corner of the source ROI,
- quad*[3] corresponds to the transformed bottom-left corner of the source ROI.

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>srcRoi</i> has a size field with zero or negative value.
<code>ippStsCoeffErr</code>	Indicates an error condition if coefficient values are invalid.

## GetPerspectiveBound

*Computes the bounding rectangle for the source ROI transformed by the `ippiWarpPerspective` function.*

---

### Syntax

```
IppStatus ippiGetPerspectiveBound(IppiRect srcRoi, double bound[2][2], const double coeffs[3][3]);
```

### Parameters

<i>srcRoi</i>	Region of interest in the source image (of the <code>IppiRect</code> type).
<i>bound</i>	Output array. Contains vertex coordinates of the bounding rectangle for the transformed source ROI.
<i>coeffs</i>	The given perspective transform coefficients.

### Description

The function `ippiGetPerspectiveBound` is declared in the `ippi.h` file. It operates with ROI (see [ROI Processing in Geometric Transforms](#)).

This function is used as a support function for `ippiWarpPerspective`. It computes vertex coordinates of the smallest bounding rectangle for the quadrangle *quad*, to which the source ROI is mapped by the perspective transform function `ippiWarpPerspective` using the given coefficients *coeffs*.

*bound*[0] specifies x, y coordinates of the top-left corner, *bound*[1] specifies x, y coordinates of the bottom-right corner.

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>srcRoi</i> has a size field with zero or negative value.
<code>ippStsCoeffErr</code>	Indicates an error condition if coefficient values are invalid.

## GetPerspectiveTransform

*Computes the perspective transform coefficients to map the source ROI to the quadrangle with the specified vertex coordinates.*

---

### Syntax

```
IppStatus ippiGetPerspectiveTransform(IppiRect srcRoi, const double quad[4][2], double coeffs[3][3]);
```

### Parameters

<i>srcRoi</i>	Region of interest in the source image (of the <code>IppiRect</code> type).
---------------	---

<i>quad</i>	Vertex coordinates of the quadrangle, to which the source ROI is mapped by the perspective transform function.
<i>coeffs</i>	Output array. Contains the target perspective transform coefficients.

### Description

The function `ippiGetPerspectiveTransform` is declared in the `ippi.h` file. It operates with ROI (see ROI Processing in Geometric Transforms).

This function is used as a support function for `ippiWarpPerspective`. It computes the coefficients *coeffs* that should be used by the function `ippiWarpPerspective` to map the source rectangular ROI to the quadrangle with the given vertex coordinates *quad*.

The first dimension [4] of the array *quad*[4][2] is equal to the number of vertices, and the second dimension [2] means *x* and *y* coordinates of the vertex. Quadrangle vertices have the following meaning:

*quad*[0] corresponds to the transformed top-left corner of the source ROI,  
*quad*[1] corresponds to the transformed top-right corner of the source ROI,  
*quad*[2] corresponds to the transformed bottom-right corner of the source ROI,  
*quad*[3] corresponds to the transformed bottom-left corner of the source ROI.

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>srcRoi</i> has a size field with zero or negative value.
<code>ippStsCoeffErr</code>	Indicates an error condition if coefficient values are invalid.
<code>ippStsRectErr</code>	Indicates an error condition if width or height of the <i>srcRoi</i> is less than or equal to 1.



### Example

The code example below shows how to use the function `ippiGetPerspectiveTransform`.

```
void func_WarpPerspective()
{
    int step;
    Ipp32f pSrc[8*8] = {1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0,
                       1.0, 1.0, 1.0, 5.0, 5.0, 1.0, 1.0, 1.0,
                       1.0, 1.0, 5.0, 5.0, 5.0, 5.0, 1.0, 1.0,
                       1.0, 5.0, 5.0, 5.0, 5.0, 5.0, 5.0, 1.0,
                       1.0, 5.0, 5.0, 5.0, 5.0, 5.0, 5.0, 1.0,
                       1.0, 1.0, 5.0, 5.0, 5.0, 5.0, 1.0, 1.0,
                       1.0, 1.0, 1.0, 5.0, 5.0, 1.0, 1.0, 1.0,
                       1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0};

    int srcStep = 8*sizeof(Ipp32f);
    IppiSize srcSize = { 8, 8};
    IppiRect srcRoi = {0, 0, 8, 8};
```

```

Ipp32f pDst[8*8];
Ipp32f pDstB[8*8];
int dstStep = 8*sizeof(ipp32f);
IppiRect dstRoi = {0, 0, 8, 8};
IppiSize dstSize = {8, 8};
int dstbStep = 8*sizeof(ipp32f);
IppiRect dstbRoi = {0, 0, 8, 8};
double quad[4][2] = {{0.0, 0.0},
                     {8.0, 0.0},
                     {6.0, 8.0},
                     {2.0, 8.0}};

double coeffs[3][3];

ippiSet_32f_C1R(2,pDst,srcStep,srcSize);
ippiSet_32f_C1R(3,pDstB,srcStep,srcSize);
ippiGetPerspectiveTransform(srcRoi, quad, coeffs);
ippiWarpPerspective_32f_C1R(pSrc, srcSize, srcStep, srcRoi, pDst, dstStep,
                             dstRoi, coeffs, IPPI_INTER_NN);
ippiWarpPerspectiveBack_32f_C1R( pDst, dstSize, dstStep, dstRoi, pDstB,
                                 dstbStep, dstbRoi, coeffs,IPPI_INTER_NN);
}

```

Result:

pSrc

```

1.0  1.0  1.0  1.0  1.0  1.0  1.0  1.0
1.0  1.0  1.0  5.0  5.0  1.0  1.0  1.0
1.0  1.0  5.0  5.0  5.0  5.0  1.0  1.0
1.0  5.0  5.0  5.0  5.0  5.0  5.0  1.0
1.0  5.0  5.0  5.0  5.0  5.0  5.0  1.0
1.0  1.0  5.0  5.0  5.0  5.0  1.0  1.0
1.0  1.0  1.0  5.0  5.0  1.0  1.0  1.0
1.0  1.0  1.0  1.0  1.0  1.0  1.0  1.0

```

pDst

```

1.0  1.0  1.0  1.0  1.0  1.0  1.0  1.0
2.0  1.0  1.0  1.0  1.0  1.0  1.0  1.0
2.0  1.0  1.0  1.0  5.0  5.0  1.0  1.0
2.0  1.0  1.0  5.0  5.0  5.0  1.0  1.0
2.0  1.0  1.0  5.0  5.0  5.0  1.0  1.0
2.0  2.0  5.0  5.0  5.0  5.0  5.0  2.0
2.0  2.0  5.0  5.0  5.0  5.0  5.0  2.0
2.0  2.0  1.0  5.0  5.0  5.0  1.0  2.0

```

pDstB

```

1.0  1.0  1.0  1.0  1.0  1.0  1.0  3.0
2.0  1.0  1.0  5.0  5.0  1.0  1.0  3.0
1.0  1.0  5.0  5.0  5.0  5.0  1.0  3.0
2.0  5.0  5.0  5.0  5.0  5.0  5.0  2.0
2.0  5.0  5.0  5.0  5.0  5.0  5.0  2.0
1.0  1.0  5.0  5.0  5.0  5.0  1.0  1.0
3.0  3.0  3.0  3.0  3.0  3.0  3.0  3.0
3.0  3.0  3.0  3.0  3.0  3.0  3.0  3.0

```

## WarpBilinear

*Performs bilinear warping of the source image using the specified transform coefficients.*

### Syntax

#### Case 1: Operation on pixel-order data

```

IppStatus ippiWarpBilinear_<mod>(const Ipp<datatype>* pSrc, IppiSize srcSize, int srcStep,
IppiRect srcRoi, Ipp<datatype>* pDst, int dstStep, IppiRect dstRoi, const double
coeffs[2][4], int interpolation);

```

Supported values for mod:

8u_C1R	16u_C1R	32f_C1R
8u_C3R	16u_C3R	32f_C3R
8u_C4R	16u_C4R	32f_C4R
8u_AC4R	16u_AC4R	32f_AC4R

## Case 2: Operation on planar-order data

```
IppStatus ippiWarpBilinear_<mod>(const Ipp<datatype>* const pSrc[3], IppiSize srcSize,
int srcStep, IppiRect srcRoi, Ipp<datatype>* const pDst[3], int dstStep, IppiRect dstRoi,
const double coeffs[2][4], int interpolation);
```

Supported values for mod:

8u\_P3R            16u\_P3R            32f\_P3R

```
IppStatus ippiWarpBilinear_<mod>(const Ipp<datatype>* const pSrc[4], IppiSize srcSize,
int srcStep, IppiRect srcRoi, Ipp<datatype>* const pDst[4], int dstStep, IppiRect dstRoi,
const double coeffs[2][4], int interpolation);
```

Supported values for mod:

8u\_P4R            16u\_P4R            32f\_P4R

## Parameters

<i>pSrc</i>	Pointer to the source image origin. An array of separate pointers to each plane in case of data in planar format.								
<i>srcSize</i>	Size in pixels of the source image.								
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image buffer.								
<i>srcRoi</i>	Region of interest in the source image (of the <code>IppiRect</code> type).								
<i>pDst</i>	Pointer to the destination image origin. An array of separate pointers to each plane in case of data in planar format.								
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image buffer.								
<i>dstRoi</i>	Region of interest in the destination image (of the <code>IppiRect</code> type).								
<i>coeffs</i>	The bilinear transform coefficients.								
<i>interpolation</i>	Specifies the <a href="#">interpolation</a> mode. Use one of the following values: <table data-bbox="503 1438 1347 1619"> <tr> <td><code>IPPI_INTER_NN</code></td><td>nearest neighbor interpolation</td></tr> <tr> <td><code>IPPI_INTER_LINEAR</code></td><td>linear interpolation</td></tr> <tr> <td><code>IPPI_INTER_CUBIC</code></td><td>cubic interpolation</td></tr> <tr> <td><code>IPPI_SMOOTH_EDGE</code></td><td>use edge smoothing in addition to one of the above modes.</td></tr> </table>	<code>IPPI_INTER_NN</code>	nearest neighbor interpolation	<code>IPPI_INTER_LINEAR</code>	linear interpolation	<code>IPPI_INTER_CUBIC</code>	cubic interpolation	<code>IPPI_SMOOTH_EDGE</code>	use edge smoothing in addition to one of the above modes.
<code>IPPI_INTER_NN</code>	nearest neighbor interpolation								
<code>IPPI_INTER_LINEAR</code>	linear interpolation								
<code>IPPI_INTER_CUBIC</code>	cubic interpolation								
<code>IPPI_SMOOTH_EDGE</code>	use edge smoothing in addition to one of the above modes.								

## Description

The function `ippiWarpBilinear` is declared in the `ippi.h` file. It operates with ROI (see [ROI Processing in Geometric Transforms](#)).

This bilinear warp function transforms the source image pixel coordinates ( $x, y$ ) according to the following formulas:

$$x' = c_{00} * x * y + c_{01} * x + c_{02} * y + c_{03}$$

$$y' = c_{10} * x * y + c_{11} * x + c_{12} * y + c_{13}$$

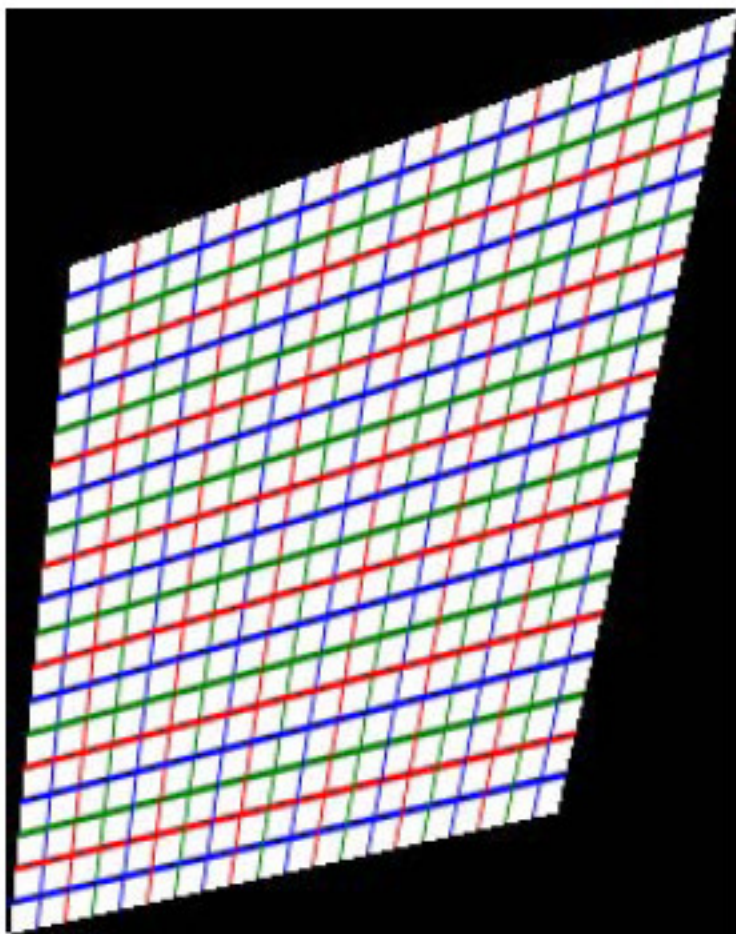
where  $x'$  and  $y'$  denote the pixel coordinates in the transformed image, and  $c_{ij}$  are the bilinear transform coefficients passed in the array *coeffs*.

The bilinear transform preserves equal distances between points on a line.

The transformed part of the source image is resampled using the [interpolation mode](#) specified by the *interpolation* parameter, and written to the destination image ROI.

[Figure "Bilinear Transform of an Image"](#) gives an example of applying the bilinear warping function `ippiWarpBilinear` to a sample image.

### Bilinear Transform of an Image



To estimate how the source image ROI will be transformed by the `ippiWarpBilinear` function, use functions `ippiWarpBilinearQuad` and `ippiGetBilinearBound`. To calculate coefficients of the bilinear transform which maps source ROI to a given quadrangle, use `ippiGetBilinearTransform` function.

**Example** shows how to use the function `ippiWarpBilinear_32f_C1R`.

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if any image dimension has zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> or <i>dstStep</i> has a zero or negative value.
<code>ippStsInterpolationErr</code>	Indicates an error condition if <i>interpolation</i> has an illegal value.
<code>ippStsRectErr</code>	Indicates an error condition if width or height of the intersection of the <i>srcRoi</i> and source image is less than or equal to 1.
<code>ippStsCoeffErr</code>	Indicates an error condition if coefficient values are invalid.
<code>ippStsWrongIntersectROIErr</code>	Indicates an error condition if <i>srcRoi</i> has no intersection with the source image.
<code>ippStsWrongIntersectQuad</code>	Indicates a warning that no operation is performed if the transformed source ROI has no intersection with the destination ROI.

## WarpBilinearBack

Performs an inverse bilinear warping of the source image.

### Syntax

#### Case 1: Operation on pixel-order data

```
IppStatus ippiWarpBilinearBack_<mod>(const Ipp<datatype>* pSrc, IppiSize srcSize, int
srcStep, IppiRect srcRoi, Ipp<datatype>* pDst, int dstStep, IppiRect dstRoi, const double
coeffs[2][4], int interpolation);
```

Supported values for `mod`:

<code>8u_C1R</code>	<code>16u_C1R</code>	<code>32f_C1R</code>
<code>8u_C3R</code>	<code>16u_C3R</code>	<code>32f_C3R</code>
<code>8u_C4R</code>	<code>16u_C4R</code>	<code>32f_C4R</code>
<code>8u_AC4R</code>	<code>16u_AC4R</code>	<code>32f_AC4R</code>

**Case 2: Operation on planar-order data**

```
IppStatus ippiWarpBilinearBack_<mod>(const Ipp<datatype>* const pSrc[3], IppiSize srcSize,
int srcStep, IppiRect srcRoi, Ipp<datatype>* const pDst[3], int dstStep, IppiRect dstRoi,
const double coeffs[2][4], int interpolation);
```

Supported values for mod:

8u\_P3R            16u\_P3R            32f\_P3R

```
IppStatus ippiWarpBilinearBack_<mod>(const Ipp<datatype>* const pSrc[4], IppiSize srcSize,
int srcStep, IppiRect srcRoi, Ipp<datatype>* const pDst[4], int dstStep, IppiRect dstRoi,
const double coeffs[2][4], int interpolation);
```

Supported values for mod:

8u\_P4R            16u\_P4R            32f\_P4R

**Parameters**

<i>pSrc</i>	Pointer to the source image origin. An array of separate pointers to each plane in case of data in planar format.						
<i>srcSize</i>	Size in pixels of the source image.						
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image buffer.						
<i>srcRoi</i>	Region of interest in the source image (of the <code>IppiRect</code> type).						
<i>pDst</i>	Pointer to the destination image origin. An array of separate pointers to each plane in case of data in planar format.						
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image buffer.						
<i>dstRoi</i>	Region of interest in the destination image (of the <code>IppiRect</code> type).						
<i>coeffs</i>	The bilinear transform coefficients.						
<i>interpolation</i>	Specifies the <a href="#">interpolation</a> mode. Use one of the following values: <table data-bbox="503 1438 1209 1549"> <tr> <td>IPPI_INTER_NN</td><td>nearest neighbor interpolation</td></tr> <tr> <td>IPPI_INTER_LINEAR</td><td>linear interpolation</td></tr> <tr> <td>IPPI_INTER_CUBIC</td><td>cubic interpolation.</td></tr> </table>	IPPI_INTER_NN	nearest neighbor interpolation	IPPI_INTER_LINEAR	linear interpolation	IPPI_INTER_CUBIC	cubic interpolation.
IPPI_INTER_NN	nearest neighbor interpolation						
IPPI_INTER_LINEAR	linear interpolation						
IPPI_INTER_CUBIC	cubic interpolation.						

**Description**

The function `ippiWarpBilinearBack` is declared in the `ippi.h` file. It operates with ROI (see [ROI Processing in Geometric Transforms](#)).

This function performs the inverse transform to that defined by `ippiWarpBilinear` function. Pixel coordinates  $x'$  and  $y'$  in the transformed image are obtained from the following equations

$$c_{00} * x' * y' + c_{01} * x' + c_{02} * y' + c_{03} = x$$

$$c_{10} * x' * y' + c_{11} * x' + c_{12} * y' + c_{13} = y$$

where  $x$  and  $y$  denote the pixel coordinates in the source image, and coefficients  $c_{ij}$  are given in the array *coeffs*. Thus, you don't need to invert transform coefficients in your application program before calling `ippiWarpBilinearBack`.

Note that inverse transform functions handle source and destination ROI in a different way than other geometric transform functions. See implementation details in the description of `ippiWarpAffineBack` function.

**Example** shows how to use the function `ippiWarpBilinearBack_32f_C1R`.

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if any image dimension has zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> or <i>dstStep</i> has a zero or negative value.
<code>ippStsInterpolationErr</code>	Indicates an error condition if <i>interpolation</i> has an illegal value.
<code>ippStsCoeffErr</code>	Indicates an error condition if coefficient values are invalid.
<code>ippStsWrongIntersectROIErr</code>	Indicates an error condition if <i>srcRoi</i> has no intersection with the source image.
<code>ippStsWrongIntersectQuad</code>	Indicates a warning that no operation is performed if the transformed source ROI has no intersection with the destination ROI.

## WarpBilinearQuad

*Performs bilinear warping of the source image that transforms the given source quadrangle to the specified destination quadrangle.*

---

### Syntax

#### Case 1: Operation on pixel-order data

```
IppStatus ippiWarpBilinearQuad_<mod>(const Ipp<datatype>* pSrc, IppiSize srcSize, int
srcStep, IppiRect srcRoi, const double srcQuad[4][2], Ipp<datatype>* pDst, int dstStep,
IppiRect dstRoi, const double dstQuad[4][2], int interpolation);
```

Supported values for *mod*:

8u_C1R	16u_C1R	32f_C1R
8u_C3R	16u_C3R	32f_C3R
8u_C4R	16u_C4R	32f_C4R
8u_AC4R	16u_AC4R	32f_AC4R



**Case 2: Operation on planar-order data**

```
IppStatus ippiWarpBilinearQuad_<mod>(const Ipp<datatype>* const pSrc[3], IppiSize srcSize,
int srcStep, IppiRect srcRoi, const double srcQuad[4][2], Ipp<datatype>* const pDst[3],
int dstStep, IppiRect dstRoi, const double dstQuad[4][2], int interpolation);
```

Supported values for mod:

8u\_P3R            16u\_P3R            32f\_P3R

```
IppStatus ippiWarpBilinearQuad_<mod>(const Ipp<datatype>* const pSrc[4], IppiSize srcSize,
int srcStep, IppiRect srcRoi, const double srcQuad[4][2], Ipp<datatype>* const pDst[4],
int dstStep, IppiRect dstRoi, const double dstQuad[4][2], int interpolation);
```

Supported values for mod:

8u\_P4R            16u\_P4R            32f\_P4R

**Parameters**

<i>pSrc</i>	Pointer to the source image origin. An array of separate pointers to each plane in case of data in planar format.
<i>srcSize</i>	Size in pixels of the source image.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image buffer.
<i>srcRoi</i>	Region of interest in the source image (of the <code>IppiRect</code> type).
<i>srcQuad</i>	A given quadrangle in the source image.
<i>pDst</i>	Pointer to the destination image origin. An array of separate pointers to each plane in case of data in planar format.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image buffer.
<i>dstRoi</i>	Region of interest in the destination image (of the <code>IppiRect</code> type).
<i>dstQuad</i>	A given quadrangle in the destination image.
<i>interpolation</i>	Specifies the <a href="#">interpolation</a> mode. Use one of the following values:
<code>IPPI_INTER_NN</code>	nearest neighbor interpolation
<code>IPPI_INTER_LINEAR</code>	linear interpolation
<code>IPPI_INTER_CUBIC</code>	cubic interpolation
<code>IPPI_SMOOTH_EDGE</code>	use edge smoothing in addition to one of the above modes.

**Description**

The function `ippiWarpBilinearQuad` is declared in the `ippi.h` file. It operates with ROI (see [ROI Processing in Geometric Transforms](#)).

This function applies a bilinear transform to an arbitrary quadrangle *srcQuad* in the source image *pSrc*. The operations take place only in the intersection of the source image ROI *srcRoi* and the source quadrangle *srcQuad*. The function `ippiWarpBilinearQuad` uses the same formulas for pixel mapping as in the case of the `ippiWarpBilinear` function. Transform coefficients are computed internally, based on the mapping of the source quadrangle to the quadrangle *dstQuad* specified in the destination image *pDst*. The *dstQuad* should have a non-empty intersection with the destination image ROI *dstRoi*.

The first dimension [4] of the array specifying the quadrangle *srcQuad*[4][2] or *dstQuad*[4][2] is equal to the number of vertices, and the second dimension [2] holds x and y coordinates of the vertex.

Edge smoothing interpolation is applicable only if the source quadrangle lies in the source image ROI.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if any image dimension has zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> or <i>dstStep</i> has a zero or negative value.
<code>ippStsInterpolationErr</code>	Indicates an error condition if <i>interpolation</i> has an illegal value.
<code>ippStsQuadErr</code>	Indicates an error condition if <i>srcQuad</i> or <i>dstQuad</i> degenerates into triangle.
<code>ippStsWrongIntersectROIErr</code>	Indicates an error condition if <i>srcRoi</i> has no intersection with the source image.
<code>ippStsWrongIntersectQuad</code>	Indicates a warning that no operation is performed if the <i>srcRoi</i> has no intersection with the <i>srcQuad</i> , or <i>dstRoi</i> has no intersection with the <i>dstQuad</i> .

## GetBilinearQuad

*Computes the vertex coordinates of the quadrangle, to which the source rectangular ROI is mapped by the bilinear transform.*

---

### Syntax

```
IppStatus ippiGetBilinearQuad(IppiRect srcRoi, double quad[4][2], const double coeffs[2][4]);
```

### Parameters

<i>srcRoi</i>	Region of interest in the source image (of the <code>IppiRect</code> type).
<i>quad</i>	Output array. Contains vertex coordinates of the quadrangle, to which the source ROI is mapped by the bilinear transform function.
<i>coeffs</i>	The given bilinear transform coefficients.

## Description

The function `ippiGetBilinearQuad` is declared in the `ippi.h` file. It operates with ROI (see [ROI Processing in Geometric Transforms](#)).

This function is used as a support function for `ippiWarpBilinear`. It computes vertex coordinates of the quadrangle, to which the source rectangular ROI is mapped by the bilinear transform function `ippiWarpBilinear` using coefficients `coeffs`.

The first dimension [4] of the array `quad[4][2]` is equal to the number of vertices, and the second dimension [2] means *x* and *y* coordinates of the vertex. Quadrangle vertices have the following meaning:

`quad[0]` corresponds to the transformed top-left corner of the source ROI,

`quad[1]` corresponds to the transformed top-right corner of the source ROI,

`quad[2]` corresponds to the transformed bottom-right corner of the source ROI,

`quad[3]` corresponds to the transformed bottom-left corner of the source ROI.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
<code>ippStsSizeErr</code>	Indicates an error condition if <code>srcRoi</code> has a size field with zero or negative value.
<code>ippStsCoeffErr</code>	Indicates an error condition if coefficient values are invalid.

## GetBilinearBound

*Computes the bounding rectangle for the source ROI transformed by the `ippiWarpBilinear` function.*

---

## Syntax

```
ippStatus ippiGetBilinearBound(IppiRect srcRoi, double bound[2][2], const double coeffs[2][4]);
```

## Parameters

<code>srcRoi</code>	Region of interest in the source image (of the <code>IppiRect</code> type).
<code>bound</code>	Output array. Contains vertex coordinates of the bounding rectangle for the transformed source ROI.
<code>coeffs</code>	The given bilinear transform coefficients.

## Description

The function `ippiGetBilinearBound` is declared in the `ippi.h` file. It operates with ROI (see [ROI Processing in Geometric Transforms](#)).

This function is used as a support function for `ippiWarpBilinear`. It computes vertex coordinates of the smallest bounding rectangle for the quadrangle `quad`, to which the source ROI is mapped by the bilinear transform function `ippiWarpBilinear` using coefficients `coeffs`.

`bound[0]` specifies *x*, *y* coordinates of the top-left corner, `bound[1]` specifies *x*, *y* coordinates of the bottom-right corner.

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsSizeErr</code>	Indicates an error condition if <code>srcRoi</code> has a size field with zero or negative value.
<code>ippStsCoeffErr</code>	Indicates an error condition if coefficient values are invalid.

## GetBilinearTransform

*Computes bilinear transform coefficients to map the source ROI to the quadrangle with the specified vertex coordinates.*

---

### Syntax

```
IppStatus ippGetBilinearTransform(IppiRect srcRoi, const double quad[4][2], double
coeffs[2][4]);
```

### Parameters

<code>srcRoi</code>	Region of interest in the source image (of the <code>IppiRect</code> type).
<code>quad</code>	Vertex coordinates of the quadrangle, to which the source ROI is mapped by the bilinear transform function.
<code>coeffs</code>	Output array. Contains the target bilinear transform coefficients.

### Description

The function `ippGetBilinearTransform` is declared in the `ippi.h` file. It operates with ROI (see [ROI Processing in Geometric Transforms](#)).

This function is used as a support function for `ippiWarpBilinear`. It computes the coefficients `coeffs` of the bilinear transform that maps the source rectangular ROI to the quadrangle with the specified vertex coordinates `quad`.

The first dimension [4] of the array `quad[4][2]` is equal to the number of vertices, and the second dimension [2] means *x* and *y* coordinates of the vertex. Quadrangle vertices have the following meaning:

- `quad[0]` corresponds to the transformed top-left corner of the source ROI,
- `quad[1]` corresponds to the transformed top-right corner of the source ROI,
- `quad[2]` corresponds to the transformed bottom-right corner of the source ROI,
- `quad[3]` corresponds to the transformed bottom-left corner of the source ROI.

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
<code>ippStsSizeErr</code>	Indicates an error condition if <code>srcRoi</code> has a size field with zero or negative value.

---

<code>ippStsCoeffErr</code>	Indicates an error condition if coefficient values are invalid.
<code>ippStsRectErr</code>	Indicates an error condition if width or height of the <code>srcRoi</code> is less than or equal to 1.

### Example

The code example below shows how to use the function `ippiGetBilinearTransform`.

```
void func_WarpBilinear()
{
    int step;
    Ipp32f pSrc[8*8] = {1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0,
                        1.0, 1.0, 1.0, 5.0, 5.0, 1.0, 1.0, 1.0,
                        1.0, 1.0, 5.0, 5.0, 5.0, 5.0, 1.0, 1.0,
                        1.0, 5.0, 5.0, 5.0, 5.0, 5.0, 5.0, 1.0,
                        1.0, 5.0, 5.0, 5.0, 5.0, 5.0, 5.0, 1.0,
                        1.0, 1.0, 5.0, 5.0, 5.0, 5.0, 1.0, 1.0,
                        1.0, 1.0, 1.0, 5.0, 5.0, 1.0, 1.0, 1.0,
                        1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0};

    int srcStep = 8*sizeof(Ipp32f);
    IppiSize srcSize = { 8, 8};
    IppiRect srcRoi = {0, 0, 8, 8};
```

```
Ipp32f pDst[8*8];
Ipp32f pDstB[8*8];
int dstStep = 8*sizeof(ipp32f);
IppiRect dstRoi = {0, 0, 8, 8};
IppiSize dstSize = {8, 8};
int dstbStep = 8*sizeof(ipp32f);
IppiRect dstbRoi = {0, 0, 8, 8};

double quad[4][2] = {{1.0, 1.0},
                    {8.0, 0.0},
                    {6.0, 8.0},
                    {2.0, 6.0}};

double coeffs[2][4];

ippiSet_32f_C1R(2,pDst,srcStep,srcSize);
ippiSet_32f_C1R(3,pDstB,srcStep,srcSize);

ippiGetBilinearTransform(srcRoi, quad, coeffs);
ippiWarpBilinear_32f_C1R( pSrc, srcSize, srcStep, srcRoi, pDst, dstStep,
                        dstRoi, coeffs, IPPI_INTER_NN);
```

```
ippiWarpBilinearBack_32f_C1R( pDst, dstSize, dstStep, dstRoi, pDstB,  
                               dstbStep, dstbRoi, coeffs, IPPI_INTER_NN);  
}
```

Result:

pSrc

1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
1.0	1.0	1.0	5.0	5.0	1.0	1.0	1.0
1.0	1.0	5.0	5.0	5.0	5.0	1.0	1.0
1.0	5.0	5.0	5.0	5.0	5.0	5.0	1.0
1.0	5.0	5.0	5.0	5.0	5.0	5.0	1.0
1.0	1.0	5.0	5.0	5.0	5.0	1.0	1.0
1.0	1.0	1.0	5.0	5.0	1.0	1.0	1.0
1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0

pDst

2.0	2.0	2.0	2.0	2.0	2.0	2.0	2.0
2.0	1.0	1.0	1.0	1.0	5.0	1.0	1.0
2.0	2.0	1.0	5.0	5.0	5.0	5.0	1.0
2.0	2.0	5.0	5.0	5.0	5.0	5.0	1.0
2.0	2.0	5.0	5.0	5.0	5.0	5.0	1.0
2.0	2.0	1.0	5.0	5.0	5.0	5.0	2.0
2.0	2.0	1.0	1.0	5.0	1.0	1.0	2.0
2.0	2.0	2.0	2.0	1.0	1.0	1.0	2.0

pDstB

1.0	1.0	1.0	1.0	2.0	2.0	2.0	3.0
2.0	1.0	5.0	1.0	5.0	1.0	1.0	3.0
2.0	1.0	5.0	5.0	5.0	5.0	1.0	3.0
2.0	5.0	5.0	5.0	5.0	5.0	5.0	3.0
5.0	5.0	5.0	5.0	5.0	5.0	5.0	2.0
1.0	1.0	5.0	5.0	5.0	5.0	1.0	2.0
1.0	1.0	1.0	5.0	5.0	1.0	1.0	1.0
1.0	1.0	2.0	1.0	3.0	3.0	3.0	3.0





# Wavelet Transforms

This chapter describes the Intel® IPP image processing functions that perform two-dimensional discrete wavelet transform (DWT).

In many applications the multiresolution analysis by discrete wavelet transforms is a better alternative to windowing and discrete Fourier analysis techniques. On the one hand, the forward two-dimensional wavelet transform may be considered as a decomposition of an image on the base of functions bounded or localized in space; and on the other, the wavelet transforms are related to subband filtering and resampling.

Intel IPP for image processing contains one-level discrete wavelet decomposition and reconstruction functions. It also provides the necessary interface for initialization and deallocation of the transform context structure. [Table “Image Wavelet Transform Functions”](#) lists all functions of this group:

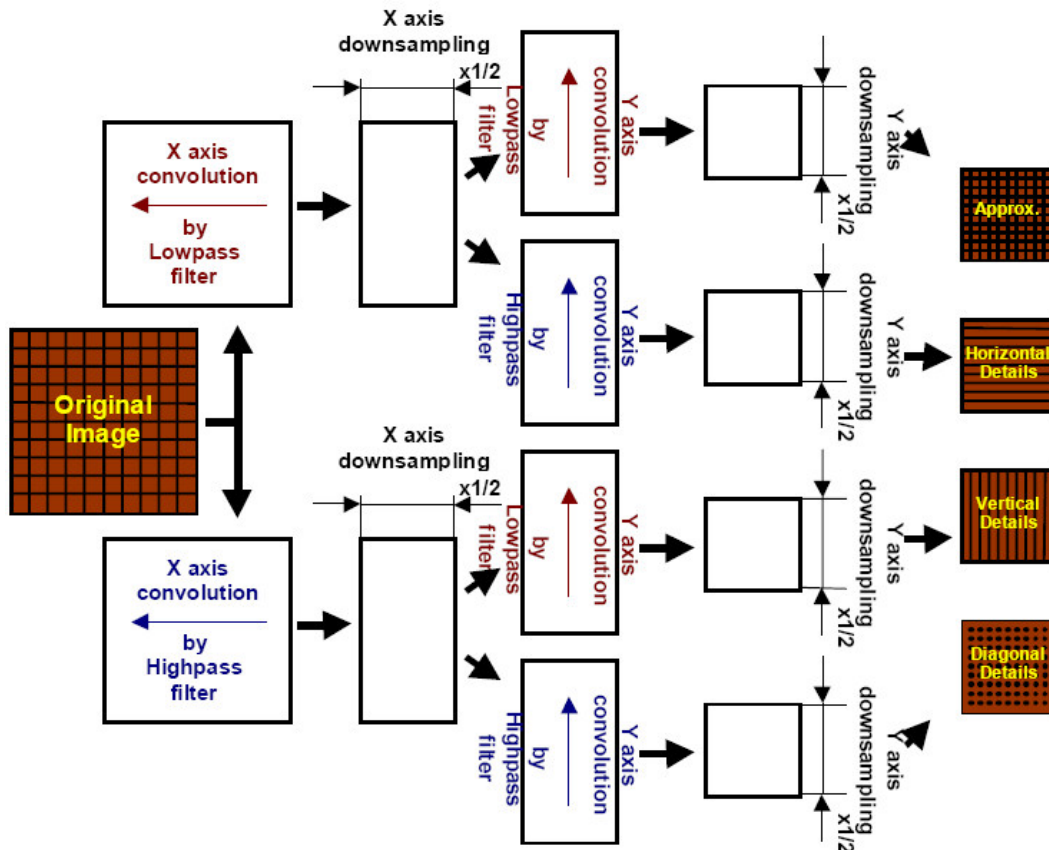
## Image Wavelet Transform Functions

Function Base Name	Operation
<a href="#">WTFwdInitAlloc</a>	Allocates memory and initializes the forward wavelet transform context structure
<a href="#">WTFwdFree</a>	Deallocates memory allocated for a forward wavelet transform context structure
<a href="#">WTFwdGetBufSize</a>	Determines the size of external work buffer for a forward wavelet transform
<a href="#">WTFwd</a>	Performs one-level wavelet decomposition of an image
<a href="#">WTInvInitAlloc</a>	Allocates memory and initializes the inverse wavelet transform context structure
<a href="#">WTInvFree</a>	Deallocates memory allocated for an inverse wavelet transform context structure
<a href="#">WTInvGetBufSize</a>	Determines the size of external work buffer for an inverse wavelet transform
<a href="#">WTInv</a>	Performs one-level wavelet reconstruction of an image

The wavelet transform type can be set by specifying the appropriate filter taps in the initialization function. Note that Intel IPP supports only one-dimensional finite impulse response filters for separable convolution.

The Intel IPP functions for wavelet decomposition and reconstruction use fast polyphase algorithm, which is equivalent to traditional application of separable convolution and dyadic resampling in different order. [Figure “Equivalent Scheme of Wavelet Decomposition Algorithm”](#) shows the equivalent algorithm of wavelet-based image decomposition:

### Equivalent Scheme of Wavelet Decomposition Algorithm



Decomposition operation applied to a source image produces four output images of equal size: approximation image, horizontal detail image, vertical detail image, and diagonal detail image.

These decomposition components have the following meaning:

- The 'approximation' image is obtained by vertical and horizontal lowpass filtering.
- The 'horizontal detail' image is obtained by vertical highpass and horizontal lowpass filtering.
- The 'vertical detail' image is obtained by vertical lowpass and horizontal highpass filtering.
- The 'diagonal detail' image is obtained by vertical and horizontal highpass filtering.

The above image names are used in this manual for identification convenience only.

The wavelet-based image reconstruction can be represented by a sequence of separate convolution and dyadic upsampling.

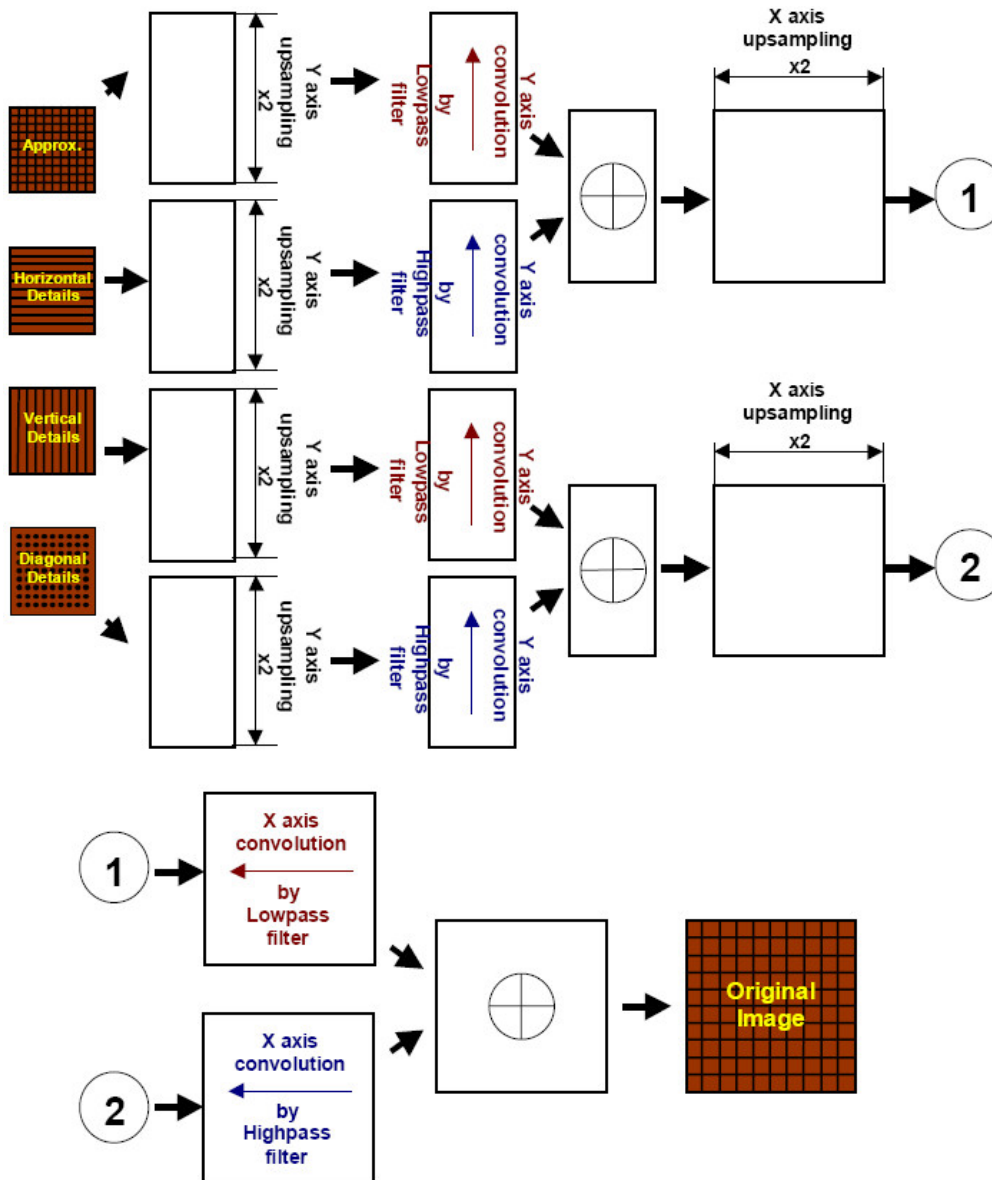
The reconstruction function uses four input images that are the same as those resulting from the decomposition operation.

Figure “Equivalent Scheme of Wavelet Reconstruction Algorithm” shows the equivalent algorithm of wavelet reconstruction of an image.

Wavelet transform functions support regions of interest (ROI, see [Regions of Interest in Intel IPP in chapter 2](#)) in the images. However, these functions do not perform internally any border extensions of image ROI data. It means that source images must already contain all border data that are necessary for convolution operations. See descriptions of the functions `ippiWTFwd` and `ippiWTInv` for detailed information on how to calculate extended image border sizes.

The use of Intel IPP for wavelet transform is demonstrated in the *Wavelet Transform Sample*. In addition to this sample many solutions and hints for use of Intel IPP in wavelet applications can be found in the *JPEG 2000 Encode-Decode Sample*. Both samples can be downloaded from <http://www.intel.com/cd/software/products/asmo-na/eng/220046.htm>.

### Equivalent Scheme of Wavelet Reconstruction Algorithm



## WTFwdInitAlloc

*Allocates memory and initializes the forward wavelet transform context structure.*

### Syntax

```
IppStatus ippiWTFwdInitAlloc_32f_C1R(IppiWTFwdSpec_32f_C1R** ppSpec, const Ipp32f* pTapsLow,
int lenLow, int anchorLow, const Ipp32f* pTapsHigh, int lenHigh, int anchorHigh);
```

```
IppStatus ippiWTFwdInitAlloc_32f_C3R(IppiWTFwdSpec_32f_C3R** ppSpec, const Ipp32f* pTapsLow,
int lenLow, int anchorLow, const Ipp32f* pTapsHigh, int lenHigh, int anchorHigh);
```

### Parameters

<i>ppSpec</i>	Double pointer to the forward DWT context structure.
<i>pTapsLow</i>	Pointer to lowpass filter taps.
<i>lenLow</i>	Length of the lowpass filter.
<i>anchorLow</i>	Anchor position of the lowpass filter.
<i>pTapsHigh</i>	Pointer to highpass filter taps.
<i>lenHigh</i>	Length of the highpass filter.
<i>anchorHigh</i>	Anchor position of the highpass filter.

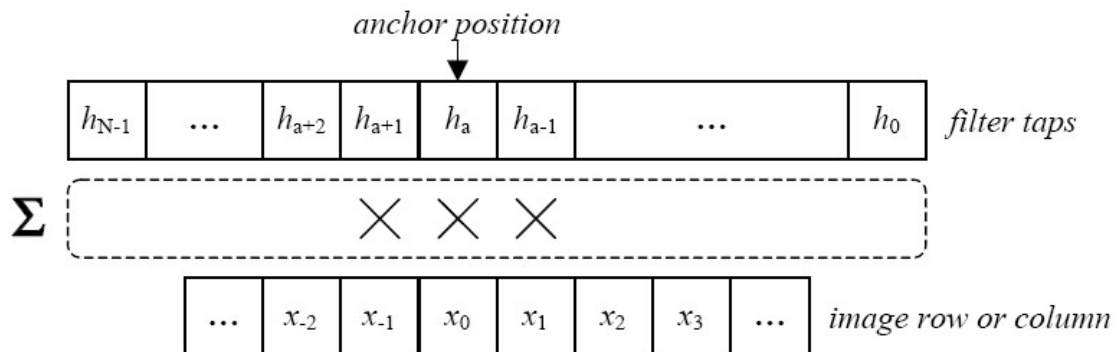
### Description

The function `ippiWTFwdInitAlloc` is declared in the `ippi.h` file. This function allocates memory for the context structure *pSpec* of a one-level wavelet decomposition and initializes this structure.

The forward wavelet transform context structure contains parameters of a wavelet filter bank used for image decomposition. The filter bank consists of two analysis filters and includes the lowpass decomposition filter (or *coarse* filter) and the highpass decomposition filter (or *detail* filter).

The parameters *pTapsLow* and *pTapsHigh* specify coefficients, and *anchorLow* and *anchorHigh* - anchor positions for two synthesis filters. The anchor value sets the initial leftmost filter position relative to image row or column as shown in the [Figure "Anchor Value and Initial Filter Position for Wavelet Decomposition"](#):

### Anchor Value and Initial Filter Position for Wavelet Decomposition



Here *a* stands for anchor value, *N* is filter length, *x0* is the starting pixel of the processed row or column, and *x-1*, *x-2*, ... are the additional border pixels that are needed for calculations. The anchor value and filter length completely determine right, left, top, and bottom border sizes for the source image used in decomposition. The corresponding C-language expressions to calculate border sizes are given in the description of `ippiWTFwd` function.

Once allocated and initialized the context structure can be used in several threads of calculations as well as in a number of wavelet decomposition levels.

[Example](#) shows how to use the function `ippiWTFwdInitAlloc_32f_C1R`.

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pTapsLow</i> , <i>pTapsHigh</i> , or <i>pSpec</i> pointer is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if filter length <i>lenLow</i> or <i>lenHigh</i> is less than 2.
<code>ippStsAnchorErr</code>	Indicates an error condition if anchor position <i>anchorLow</i> or <i>anchorHigh</i> is less than zero.
<code>ippStsMemAllocErr</code>	Indicates an error condition in case of memory allocation failure for the context structure.

## WTFwdFree

*Frees memory allocated for a forward wavelet transform context structure.*

---

### Syntax

```
ippStatus ippiWTFwdFree_32f_C1R (IppiWTFwdSpec_32f_C1R* pSpec);
ippStatus ippiWTFwdFree_32f_C3R (IppiWTFwdSpec_32f_C3R* pSpec);
```

### Parameters

*pSpec*                      Pointer to an allocated and initialized forward DWT context structure.

### Description

The function `ippiWTFwdFree` is declared in the `ippi.h` file. This function frees memory previously allocated by the function `ippiWTFwdFree` for a forward wavelet transform context structure *pSpec*.

[Example](#) shows how to use the function `ippiWTFwdFree_32f_C1R`.

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pSpec</i> pointer is <code>NULL</code> .
<code>ippStsContextMatchErr</code>	Indicates an error condition if a pointer to an invalid context structure is passed.

## WTFwdGetBufSize

*Calculates the size of external work buffer for a forward wavelet transform.*

---

### Syntax

```
IppStatus ippiWTFwdGetBufSize_C1R(const IppiWTFwdSpec_32f_C1R* pSpec, int* pBufferSize);
IppStatus ippiWTFwdGetBufSize_C3R(const IppiWTFwdSpec_32f_C3R* pSpec, int* pBufferSize);
```

### Parameters

<i>pSpec</i>	Pointer to an allocated and initialized forward DWT context structure.
<i>pBufferSize</i>	Pointer to a variable that will receive the size of work buffer required for forward wavelet transform.

### Description

The function `ippiWTFwdGetBufSize` is declared in the `ippi.h` file. This function computes the size in bytes of the work buffer required for the forward wavelet transform function `ippiWTFwd`.

[Example](#) shows how to use the function `ippiWTInvInitAlloc_32f_C1R`.

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if any of the specified pointers is <code>NULL</code> .
<code>ippStsContextMatchErr</code>	Indicates an error condition if a pointer to an invalid context structure is passed.

## WTFwd

*Performs one-level wavelet decomposition of an image.*

---

### Syntax

```
IppStatus ippiWTFwd_32f_C1R (const Ipp32f* pSrc, int srcStep, Ipp32f* pApproxDst, int approxStep, Ipp32f* pDetailXDst, int detailXStep, Ipp32f* pDetailYDst, int detailYStep, Ipp32f* pDetailXYDst, int detailXYStep, IppiSize dstRoiSize, const IppiWTFwdSpec_32f_C1R* pSpec, Ipp8u* pBuffer);
IppStatus ippiWTFwd_32f_C3R (const Ipp32f* pSrc, int srcStep, Ipp32f* pApproxDst, int approxStep, Ipp32f* pDetailXDst, int detailXStep, Ipp32f* pDetailYDst, int detailYStep, Ipp32f* pDetailXYDst, int detailXYStep, IppiSize dstRoiSize, const IppiWTFwdSpec_32f_C3R* pSpec, Ipp8u* pBuffer);
```

### Parameters

<i>pSrc</i>	Pointer to the source image ROI.
-------------	----------------------------------

<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image buffer.
<i>pApproxDst</i>	Pointer to ROI of the destination approximation image.
<i>approxStep</i>	Distance in bytes between starts of consecutive lines in the approximation image buffer.
<i>pDetailXDst</i>	Pointer to ROI of the destination horizontal detail image.
<i>detailXStep</i>	Distance in bytes between starts of consecutive lines in the horizontal detail image buffer.
<i>pDetailYDst</i>	Pointer to ROI of the destination vertical detail image.
<i>detailYStep</i>	Distance in bytes between starts of consecutive lines in the vertical detail image buffer.
<i>pDetailXYDst</i>	Pointer to ROI of the destination diagonal detail image.
<i>detailXYStep</i>	Distance in bytes between starts of consecutive lines in the diagonal detail image buffer.
<i>dstRoiSize</i>	Size of the ROI in pixels for all destination images.
<i>pSpec</i>	Pointer to the allocated and initialized forward DWT context structure.
<i>pBuffer</i>	Pointer to the allocated buffer for intermediate operations

## Description

The function `ippiWTFwd` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)). This function performs one-level wavelet decomposition of a source image pointed to by `pSrc` into four destination subimages pointed to by `pApproxDst`, `pDetailXDst`, `pDetailYDst`, and `pDetailXYDst`. See [Figure "Equivalent Scheme of Wavelet Decomposition Algorithm"](#) for the equivalent algorithm of `ippiWTFwd` function operation.

Wavelet parameters are contained in the forward transform context structure `pSpec`. It must be allocated and initialized by the function `ippiWTFwdInitAlloc` beforehand.

The decomposition function needs a buffer `pBuffer` for temporary calculations. Its size can be computed by calling the function `ippiWTFwdGetBufSize`. The buffer size does not depend on the size of processed images, and the same buffer can be used for decomposing images of different size. However, it is not recommended to use the same buffer in different threads in a multithreaded mode. For better performance, use an aligned memory location for this buffer, which can be allocated by the function `ippiMalloc`.

### Optimization Notice

Intel's compilers may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include SSE2, SSE3, and SSSE3 instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on microprocessors not manufactured by Intel. Microprocessor-dependent optimizations in this product are intended for use with Intel microprocessors. Certain optimizations not specific to Intel microarchitecture are reserved for Intel microprocessors. Please refer to the applicable product User and Reference Guides for more information regarding the specific instruction sets covered by this notice.

Notice revision #20110804



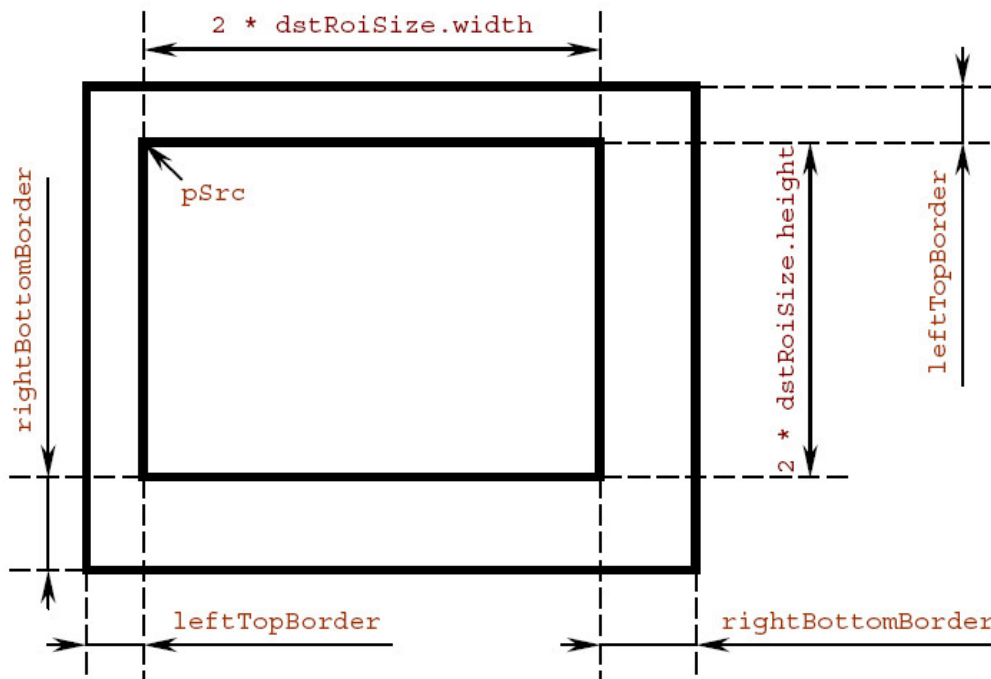
The pointer *pSrc* points to memory location of the source image rectangular ROI of size *srcWidth* by *srcHeight* which is uniquely determined by the size of destination ROI as:

$\text{srcWidth} = 2 * \text{dstRoiSize.width}$   $\text{srcHeight} = 2 * \text{dstRoiSize.height}$

Note the source image ROI does not include border pixels necessary to compute some destination pixels. It means that prior to using `ippiWTFwd` function the application program must apply some border extension model (symmetrical, wraparound or another) to the source image ROI through filling of neighboring memory locations. As a result, the size of memory block allocated for the source image must be extended to accommodate for added border pixels outside ROI formal boundaries.

Figure “Extended Source Image for Wavelet Decomposition” schematically shows the source image ROI and extended image area.

### Extended Source Image for Wavelet Decomposition



Use the following C-language expressions to calculate extended image border sizes:

```
int leftBorderLow  = lenLow  - 1 - anchorLow;
int leftBorderHigh = lenHigh - 1 - anchorHigh;
int rightBorderLow = lenLow  - 2 - leftBorderLow;
int rightBorderHigh = lenHigh - 2 - leftBorderHigh;
int leftTopBorder  = IPP_MAX(leftBorderLow, leftBorderHigh);
int rightBottomBorder = IPP_MAX(rightBorderLow, rightBorderHigh);
```

See the description of the function `ippiWTFwdInitAlloc` for the explanation of the parameters.

Note that the left and top borders have equal size. The same holds for the right and bottom borders which have equal size too.

The size of the source image area extended by border pixels can be defined as

```
srcWidthWithBorders = srcWidth + leftTopBorder + rightBottomBorder;
srcHeightWithBorders = srcHeight + leftTopBorder + rightBottomBorder;
```

All destination images have equal size specified by the parameter *dstRoiSize*.

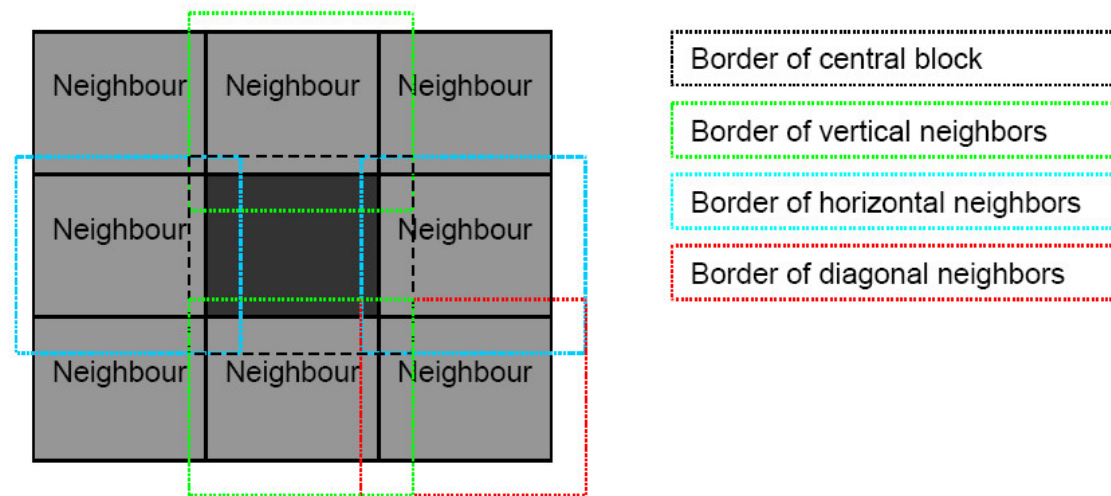
Conversely, to perform a wavelet reconstruction of the full size source image from the component images obtained by decomposition, use extended component images for the reconstruction pass. See the description of the function [ippiWTInv](#) for more details.

The ROI concept used in wavelet transform functions can be applied to processing large images by blocks, or 'tiles'. To accomplish this, the source image should be subdivided into overlapping blocks in the following way:

- Main part (ROI) of each block is adjacent to neighboring blocks and has no intersection with neighbor's ROIs;
- Extended borders of each block overlap with ROIs of neighboring blocks.

This subdivision scheme is illustrated in [Figure "Image Division into Blocks with Overlapping Borders"](#) .

## Image Division into Blocks with Overlapping Borders



[Example](#) shows how to use the function `ippiWTFwd_32f_C1R`.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if any of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <i>dstRoiSize</i> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if step through any buffer is less than or equal to zero.
<code>ippStsContextMatchErr</code>	Indicates an error condition if a pointer to an invalid context structure is passed.

## WTInvInitAlloc

*Allocates memory and initializes the inverse wavelet transform context structure.*

---

### Syntax

```
IppStatus ippiWTInvInitAlloc_32f_C1R (IppiWTInvSpec_32f_C1R** ppSpec, const Ipp32f* pTapsLow,
int lenLow, int anchorLow, const Ipp32f* pTapsHigh, int lenHigh, int anchorHigh);

IppStatus ippiWTInvInitAlloc_32f_C3R (IppiWTInvSpec_32f_C3R** ppSpec, const Ipp32f* pTapsLow,
int lenLow, int anchorLow, const Ipp32f* pTapsHigh, int lenHigh, int anchorHigh);
```

### Parameters

<i>ppSpec</i>	Double pointer to a new allocated and initialized inverse DWT context structure.
<i>pTapsLow</i>	Pointer to lowpass filter taps.
<i>lenLow</i>	Length of the lowpass filter.
<i>anchorLow</i>	Anchor position of the lowpass filter.
<i>pTapsHigh</i>	Pointer to highpass filter taps.
<i>lenHigh</i>	Length of the highpass filter.
<i>anchorHigh</i>	Anchor position of the highpass filter.

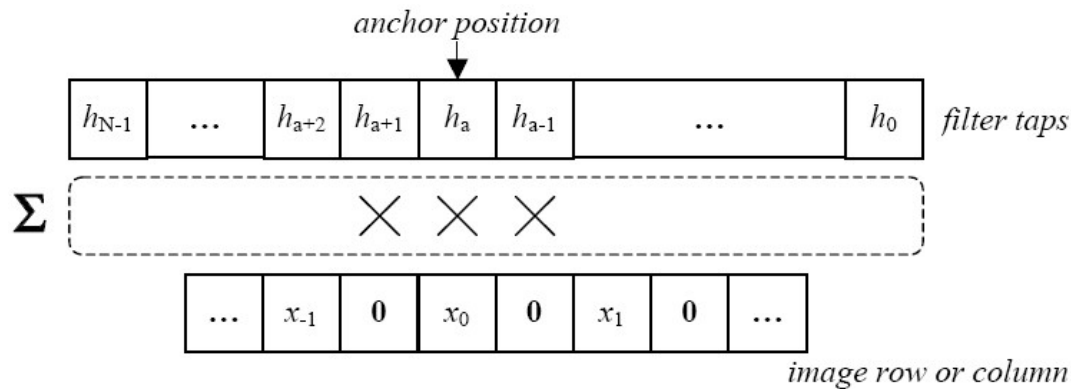
### Description

The function *ippiWTInvInitAlloc* is declared in the *ippi.h* file. This function allocates memory for the context structure *pSpec* of a one-level wavelet reconstruction and initializes this structure.

The inverse wavelet transform context structure contains parameters of a wavelet filter bank used for image reconstruction. The filter bank consists of two synthesis filters and includes the lowpass reconstruction filter (or *coarse* filter) and the highpass reconstruction filter (or *detail* filter).

The parameters *pTapsLow* and *pTapsHigh* specify coefficients, and *anchorLow* and *anchorHigh* - anchor positions for two synthesis filters. The anchor value sets the initial leftmost filter position relative to image row or column as shown in the figure below:

### Anchor Value and Initial Filter Position for Wavelet Reconstruction



Here  $a$  stands for anchor value,  $N$  is filter length,  $x_0$  is the starting pixel of the processed row or column, and  $x_{-1}, x_{-2}, \dots$  are the additional border pixels that are needed for calculations. As seen from this figure, anchor position is specified relative to upsampled source data. The anchor value and filter length completely determine right, left, top, and bottom border sizes for source images used in reconstruction. The corresponding C-language expressions to calculate border sizes are given in the description of the function [ippiWTInv](#).

Once allocated and initialized the context structure can be used in several threads of calculations as well as in a number of wavelet reconstruction levels.

[Example](#) shows how to use the function `ippiWTInvInitAlloc_32f_C1R`.

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pTapsLow</i> , <i>pTapsHigh</i> , or <i>pSpec</i> pointer is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if filter length <i>lenLow</i> or <i>lenHigh</i> is less than 2.
<code>ippStsAnchorErr</code>	Indicates an error condition if anchor position <i>anchorLow</i> or <i>anchorHigh</i> is less than zero.
<code>ippStsMemAllocErr</code>	Indicates an error condition if memory allocation fails.

### WTInvFree

*Frees memory allocated for an inverse wavelet transform context structure.*

### Syntax

```
IppStatus ippiWTInvFree_32f_C1R (IppiWTInvSpec_32f_C1R* pSpec);
```

```
IppStatus ippiWTInvFree_32f_C3R (IppiWTInvSpec_32f_C3R* pSpec);
```

## Parameters

*pSpec* Pointer to an allocated and initialized inverse DWT context structure.

## Description

The function `ippiWTInvFree` is declared in the `ippi.h` file. This function frees memory that is previously allocated by the function `ippiWTInvInitAlloc` for the inverse wavelet transform context structure *pSpec*.

[Example](#) shows how to use the function `ippiWTInvFree_32f_C1R`.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pSpec</i> pointer is <code>NULL</code> .
<code>ippStsContextMatchErr</code>	Indicates an error condition if a pointer to an invalid context structure is passed.

## WTInvGetBufSize

*Computes the size of external work buffer for an inverse wavelet transform.*

---

## Syntax

```
IppStatus ippiWTInvGetBufSize_C1R( const IppiWTInvSpec_32f_C1R* pSpec, int* pBufferSize);
IppStatus ippiWTInvGetBufSize_C3R( const IppiWTInvSpec_32f_C3R* pSpec, int* pBufferSize);
```

## Parameters

<i>pSpec</i>	Pointer to an allocated and initialized inverse DWT context structure.
<i>pBufferSize</i>	Pointer to a variable that will receive the size of work buffer for inverse wavelet transform.

## Description

The function `ippiWTInvGetBufSize` is declared in the `ippi.h` file. This function computes the size in bytes of the work buffer that is required for the inverse wavelet transform function `ippiWTInv` to operate.

[Example](#) shows how to use the function `ippiWTInvGetBufSize_32f_C1R`.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if any of the specified pointers is <code>NULL</code> .
<code>ippStsContextMatchErr</code>	Indicates an error condition if a pointer to an invalid context structure is passed.

## WTInv

Performs one-level wavelet reconstruction of an image.

### Syntax

```
IppStatus ippiWTInv_32f_C1R(const Ipp32f* pApproxSrc, int approxStep, const Ipp32f*
pDetailXSrc, int detailXStep, const Ipp32f* pDetailYSrc, int detailYStep, const Ipp32f*
pDetailXYSrc, int detailXYStep, IppiSize srcRoiSize, Ipp32f* pDst, int dstStep, const
IppiWTInvSpec_32f_C1R* pSpec, Ipp8u* pBuffer);
```

```
IppStatus ippiWTInv_32f_C3R(const Ipp32f* pApproxSrc, int approxStep, const Ipp32f*
pDetailXSrc, int detailXStep, const Ipp32f* pDetailYSrc, int detailYStep, const Ipp32f*
pDetailXYSrc, int detailXYStep, IppiSize srcRoiSize, Ipp32f* pDst, int dstStep, const
IppiWTInvSpec_32f_C3R* pSpec, Ipp8u* pBuffer);
```

### Parameters

<i>pApproxSrc</i>	Pointer to ROI of the source approximation image.
<i>approxStep</i>	Distance in bytes between starts of consecutive lines in the approximation image buffer.
<i>pDetailXSrc</i>	Pointer to ROI of the source horizontal detail image.
<i>detailXStep</i>	Distance in bytes between starts of consecutive lines in the horizontal detail image buffer.
<i>pDetailYSrc</i>	Pointer to ROI of the source vertical detail image.
<i>detailYStep</i>	Distance in bytes between starts of consecutive lines in the vertical detail image buffer.
<i>pDetailXYSrc</i>	Pointer to ROI of the source diagonal detail image.
<i>detailXYStep</i>	Distance in bytes between starts of consecutive lines in the diagonal detail image buffer.
<i>srcRoiSize</i>	Size of ROI in pixels for all source images.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image buffer.
<i>pSpec</i>	Pointer to the allocated and initialized inverse DWT context structure.
<i>pBuffer</i>	Pointer to the allocated buffer for intermediate operations.

### Description

The function `ippiWTInv` is declared in the `ippi.h` file. It operates with ROI (see Regions of Interest in Intel IPP ). This function performs wavelet reconstruction of the output image *pDst* from the four component images. See [Figure "Image Division into Blocks with Overlapping Borders"](#) for the equivalent algorithm of `ippiWTInv` function operation. Wavelet parameters are contained in the inverse transform context structure *pSpec*. It must be allocated and initialized by the function `ippiWTInvInitAlloc` beforehand.

The reconstruction function needs a buffer *pBuffer* for temporary calculations. Its size can be computed by calling the function [ippiWTInvGetBufSize](#). The buffer size does not depend on the size of processed images, and the same buffer can be used for reconstructing images of different size. However, it is not recommended to use the same buffer in different threads in a multithreaded mode. For better performance, use an aligned memory location for this buffer, which can be allocated by the function [ippiMalloc](#).

### Optimization Notice

Intel's compilers may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include SSE2, SSE3, and SSSE3 instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on microprocessors not manufactured by Intel. Microprocessor-dependent optimizations in this product are intended for use with Intel microprocessors. Certain optimizations not specific to Intel microarchitecture are reserved for Intel microprocessors. Please refer to the applicable product User and Reference Guides for more information regarding the specific instruction sets covered by this notice.

Notice revision #20110804

The pointers *pApproxSrc*, *pDetailXSrc*, *pDetailYSrc*, and *pDetailXYSrc* point to ROIs of source images excluding borders. All source ROIs have the same size *srcRoiSize*, while the destination image size is uniquely determined from the following relations:

```
dstWidth = 2 * srcRoiSize.width; dstHeight = 2 * srcRoiSize.height;
```

As source ROIs do not include border pixels required to computations, the application program have to apply a border extension model (symmetrical, wraparound or another) to ROIs of all source images filling the neighboring memory locations. Note the border sizes may be different for different source images. The following C-language expressions can be used to calculate extended image border sizes:

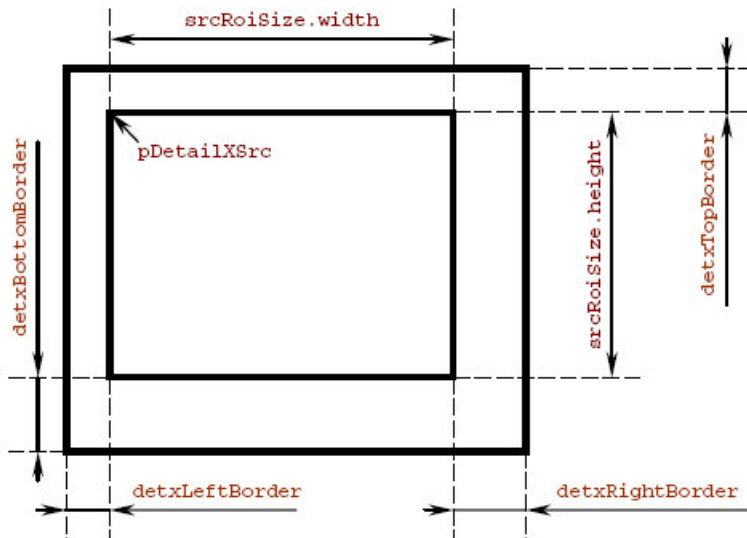
```
int leftBorderLow    = (lenLow - 1 - anchorLow) / 2;
int leftBorderHigh   = (lenHigh - 1 - anchorHigh) / 2;
int rightBorderLow   = (anchorLow + 1) / 2;
int rightBorderHigh  = (anchorHigh + 1) / 2;
int apprLeftBorder   = leftBorderLow;
int apprRightBorder  = rightBorderLow;
int apprTopBorder    = leftBorderLow;
int apprBottomBorder = rightBorderLow;
int detxLeftBorder   = leftBorderLow;
int detxRightBorder  = rightBorderLow;
int detxTopBorder    = leftBorderHigh;
int detxBottomBorder = rightBorderHigh;
int detyLeftBorder   = leftBorderHigh;
int detyRightBorder  = rightBorderHigh;
int detyTopBorder    = leftBorderLow;
int detyBottomBorder = rightBorderLow;
int detxyLeftBorder  = leftBorderHigh;
int detxyRightBorder = rightBorderHigh;
int detxyTopBorder   = leftBorderHigh;
int detxyBottomBorder = rightBorderHigh;
```

See the description of the function [ippiWTInvInitAlloc](#) for the explanation of the used parameters.

The above relations show that left and top borders always have equal size only for approximation and diagonal detail images. Right and bottom borders also have equal size only for approximation and diagonal detail images. Thus, the size of memory block allocated for each source image must be extended to accommodate for added border pixels outside ROI.

Figure “Extended Horizontal Detail Source Image for Wavelet Reconstruction” shows ROI and extended image area for the horizontal detail source image.

### Extended Horizontal Detail Source Image for Wavelet Reconstruction



Sizes of source images extended by border pixels can be calculated as follows:

```

apprWidthWithBorders  = srcWidth  + apprLeftBorder + apprRightBorder;
apprHeightWithBorders = srcHeight + apprTopBorder  + apprBottomBorder;
detxWidthWithBorders  = srcWidth  + detxLeftBorder + detxRightBorder;
detxHeightWithBorders = srcHeight + detxTopBorder  + detxBottomBorder;
detyWidthWithBorders  = srcWidth  + detyLeftBorder + detyRightBorder;
detyHeightWithBorders = srcHeight + detyTopBorder  + detyBottomBorder;
detxyWidthWithBorders = srcWidth  + detxyLeftBorder + detxyRightBorder;
detxyHeightWithBorders = srcHeight + detxyTopBorder + detxyBottomBorder;

```

The ROI concept can be used to reconstruct large images by blocks or 'tiles'.

To accomplish this, each the source images into blocks with overlapping borders, similar to what is considered in the description of the function `ippiWTFwd`. Each component must be subdivided into the same pattern of rectangular blocks.

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if any of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <code>srcRoiSize</code> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if step through any buffer is less than or equal to zero.
<code>ippStsContextMatchErr</code>	Indicates an error condition if a pointer to an invalid context structure is passed.



## Example

The example below shows how to use the function *ippiWTInv\_32f\_C1R*.

```
void func_wavelet() {
    IppiWTFwdSpec_32f_C1R* pSpec;
    IppiWTInvSpec_32f_C1R* pSpecInv;
    Ipp32f pTapsLow[3] = {0.25, 0.5, 0.25};
    int lenLow = 3;
    int anchorLow = 1;
    Ipp32f pTapsHigh[3] = { 0.75, -0.25, -0.125};
    int lenHigh = 3;
    int anchorHigh = 1;
    Ipp32f pSrc[8*8] = { 0.0,  0.0,  0.0, 11.1, 11.1,  0.0,  0.0,  0.0
                        0.0,  0.0,  0.0, 11.1, 11.1,  0.0,  0.0,  0.0,
                        0.0,  0.0,  0.0, 11.1, 11.1,  0.0,  0.0,  0.0,
                        11.1, 11.1, 11.1, 11.1, 11.1, 11.1, 11.1, 11.1,
                        11.1, 11.1, 11.1, 11.1, 11.1, 11.1, 11.1, 11.1,
                        0.0,  0.0,  0.0, 11.1, 11.1,  0.0,  0.0,  0.0,
                        0.0,  0.0,  0.0, 11.1, 11.1,  0.0,  0.0,  0.0,
                        0.0,  0.0,  0.0, 11.1, 11.1,  0.0,  0.0,  0.0};
    Ipp32f pSrcB[9*9];
    int srcStepB = 9*sizeof(Ipp32f);
    IppiSize roiSizeB = {9, 9};
    int srcStep = 8*sizeof(Ipp32f);
    IppiSize roiSize = {8, 8};
    Ipp32f pDetailXDst[4*4];
    Ipp32f pDetailYDst[4*4];
    Ipp32f pDetailXYDst[4*4];
    Ipp32f pApproxDst[4*4];
    IppiSize dstRoiSize = {4, 4};
    int bufSize, bufSizeInv;
    Ipp8u* pBuffer;
    Ipp8u* pBufferInv;
    Ipp32f pDstInv[8*8];
    Ipp32f pAppB[5*5];
    Ipp32f pXB[5*5];
```

```

Ipp32f pYB[5*5];
Ipp32f pXYB[5*5];
int StepB = 5*sizeof(Ipp32f);
IppiSize roiInvSize = {4, 4};
IppiSize roiInvSizeB = {5, 5};
int stepDstInv = 8*sizeof(Ipp32f);
int approxStep, detailXStep, detailYStep, detailXYStep;
approxStep = detailXStep = detailYStep = detailXYStep = 4*sizeof(Ipp32f);
//adds border to the source image
ippiCopyWrapBorder_32s_C1R((Ipp32s*)pSrc, srcStep, roiSize, (Ipp32s*)pSrcB, srcStepB, roiSizeB, 1, 1);
//performs forward wavelet transform

ippiWTFwdInitAlloc_32f_C1R ( &pSpec, pTapsLow, lenLow, anchorLow, pTapsHigh, lenHigh, anchorHigh);
ippiWTFwdGetBufSize_C1R(pSpec, &bufSize);
pBuffer = ippsMalloc_8u(bufSize);

ippiWTFwd_32f_C1R (pSrcB + roiSizeB.width + 1, srcStepB, pApproxDst, approxStep, pDetailXDst,
detailXStep, pDetailYDst, detailYStep, pDetailXYDst, detailXYStep,
dstRoiSize, pSpec, pBuffer);
//-----
ippiWTInvInitAlloc_32f_C1R (&pSpecInv, pTapsLow, lenLow, anchorLow, pTapsHigh, lenHigh, anchorHigh);
ippiWTInvGetBufSize_C1R(pSpecInv, &bufSizeInv);
pBufferInv = ippsMalloc_8u(bufSizeInv);
//adds border to four images obtained after ippiWTFwd
ippiCopyWrapBorder_32s_C1R((Ipp32s*)pApproxDst, approxStep, dstRoiSize, (Ipp32s*)pAppB, StepB, roiInvSizeB,
0, 0);
ippiCopyWrapBorder_32s_C1R((Ipp32s*)pDetailXDst, detailXStep, dstRoiSize, (Ipp32s*)pXB, StepB, roiInvSizeB,
0, 0);
ippiCopyWrapBorder_32s_C1R((Ipp32s*)pDetailYDst, detailYStep, dstRoiSize, (Ipp32s*)pYB, StepB, roiInvSizeB,
0, 0);

ippiCopyWrapBorder_32s_C1R((Ipp32s*)pDetailXYDst, detailXYStep, dstRoiSize, (Ipp32s*)pXYB, StepB, roiInvSizeB,
0, 0);

//performs inverse wavelet transform

```

```
ippiWTInv_32f_C1R( pAppB, StepB, pXB, StepB, pYB, StepB, pXYB, StepB, roiInvSize, pDstInv, stepDstInv,
pSpecInv,
pBufferInv);
```

```
ippiWTInvFree_32f_C1R (pSpecInv);
```

```
ippiWTFwdFree_32f_C1R (pSpec);
```

```
}
```

Result:

After WTFwd ->

pApproxDst

```
0.0  2.8   8.3  0.0
```

```
2.8  4.9   9.0  2.8
```

```
8.3  9.0  10.4 8.3
```

```
0.0  2.8   8.3  0.0
```

pDetailXDst

```
0.0   1.0  3.1   0.0
```

```
8.3   7.3  5.2   8.3
```

```
-4.2 -2.1  2.1 -4.2
```

```
0.0   1.0  3.1   0.0
```

pDetailYDst

```
0.0   8.3  -4.2   0.0
```

```
1.0   7.3  -2.1   1.0
```

```
3.1   5.2   2.1   3.1
```

```
0.0   8.3  -4.2   0.0
```

pDetailXYDst

```
0.0   3.1  -1.6   0.0
```

```
3.1   0.0   4.7   3.1
```

```
-1.6  4.7  -4.7 -1.6
```

```
0.0   3.1  -1.6   0.0
```

After WTFinv ->

pDstInv

```
0.0   2.8   -0.3  -0.6   2.1   1.1   0.0   0.0
```

```
2.8   5.9    2.7   4.9   3.4   5.4   2.8   5.1
```

```
-0.3   2.7   -0.6  -1.2   2.2   0.7  -0.3  -0.5
```

```
-0.6   4.9   -1.2  -2.2   3.9   1.2  -0.6  -1.0
```

```
2.1   3.4    2.2   3.9   1.8   3.7   2.1   3.8
```

```
1.1   5.4    0.7   1.2   3.7   3.2   1.1   1.9
```

0.0	2.8	-0.3	-0.6	2.1	1.1	0.0	0.0
0.0	5.1	-0.5	-1.0	3.8	1.9	0.0	0.0

This chapter provides some background for the computer vision concepts used in the Intel® IPP library as well as detailed descriptions of the Intel IPP image processing functions for computer vision. These functions are combined in groups by their functionality. Each group of functions is described in a separate section.

## Intel IPP Functions for Computer Vision Applications

Function Name	Description
<b>Feature Detection Functions</b>	
<a href="#">CannyGetSize</a>	Pre-calculates the size of the temporary buffer for the function <code>ippiCanny</code> .
<a href="#">Canny</a>	Finds edges in the image.
<a href="#">EigenValsVecsGetBufferSize</a>	Pre-calculates the size of the temporary buffer for the function <code>ippiEigenValsVecs</code> .
<a href="#">EigenValsVecs</a>	Calculates eigen values and eigen vectors of image blocks for corner detection.
<a href="#">MinEigenValGetBufferSize</a>	Pre-calculates the size of the temporary buffer for the function <code>ippiCornerMinEigenVal</code> .
<a href="#">MinEigenVal</a>	Calculates the minimal eigen value of image blocks for corner detection.
<a href="#">HoughLineGetSize</a>	Computes the size of the working buffer straight lines detection.
<a href="#">HoughLine</a>	Detects straight lines in the source image.
<a href="#">HoughLine_Region</a>	Detects straight lines with the specified range of parameters in the source image
<b>Distance Transform Functions</b>	
<a href="#">DistanceTransform</a>	Performs the distance transform operation.
<a href="#">GetDistanceTransformMask</a>	Calculates metrics for distance transform.
<a href="#">TrueDistanceTransformGet- BufferSize</a>	Calculates the size of the temporary working buffer for the function <code>ippiTrueDis- tanceTransform</code> .
<a href="#">TrueDistanceTransform</a>	Calculates the Euclidian distance to the closest zero pixel for all non-zero pixels of the source image.
<a href="#">FastMarchingGetBufferSize</a>	Computes the size of the working buffer for the peak search.
<a href="#">FastMarching</a>	Calculates distance transform to closest zero pixel for all non-zero pixels of source image using fast marching method.
<b>Image Gradients</b>	
<a href="#">GradientColorToGray</a>	Converts a color gradient image to grayscale.
<b>Flood Fill Functions</b>	
<a href="#">FloodFillGetSize</a>	Calculates size of temporary buffer for flood filling operation.
<a href="#">FloodFillGetSize_Grad</a>	Calculates size of temporary buffer for the gradient flood filling.
<a href="#">FloodFill</a>	Fills the connected area of the same pixel values on the image with a new value, considering 4 or 8 neighbor values of the pixel.
<a href="#">FloodFill_Grad</a>	Fills the connected area of the pixel values within a small threshold on the image with a new value, considering 4 or 8 neighbor values of the pixel.
<a href="#">FloodFill_Range</a>	Fills the connected region of the pixel values within a specified range on the image with a new value, considering 4 neighbor values of the pixel
<b>Motion Templates Functions</b>	

Function Name	Description
<a href="#">UpdateMotionHistory</a>	Updates the motion history image using the motion silhouette at a given timestamp.
<a href="#">OpticalFlowPyrLKInitAlloc</a>	Allocates memory and initializes a structure for optical flow calculation.
<a href="#">OpticalFlowPyrLKFree</a>	Frees memory allocated for the optical flow structure.
<a href="#">OpticalFlowPyrLK</a>	Calculates optical flow for the set of feature points using the pyramidal Lucas-Kanade algorithm.
<b>Pyramids Functions</b>	
<a href="#">PyrDownGetBufSize</a>	Calculates the size of a temporary buffer used by the function PyrDown.
<a href="#">PyrUpGetBufSize</a>	Calculates the size of a temporary buffer used by the function PyrUp.
<a href="#">PyrDown</a>	Applies the Gaussian to an image, then performs down-sampling.
<a href="#">PyrUp</a>	Performs up-sampling of an image, then applies the Gaussian multiplied by 4.
<b>Universal Pyramids Functions</b>	
<a href="#">PyramidInitAlloc</a>	Allocates memory and initializes a pyramid structure.
<a href="#">PyramidFree</a>	Frees memory allocated for the pyramid structure.
<a href="#">PyramidLayerDownInitAlloc</a>	Allocates memory and initializes a structure for creating a lower pyramid layer.
<a href="#">PyramidLayerUpInitAlloc</a>	Allocates memory and initializes a structure for creating an upper pyramid layer.
<a href="#">PyramidLayerDownFree</a>	Frees memory allocated for the lower pyramid layer structure.
<a href="#">PyramidLayerUpFree</a>	Frees memory allocated for the upper pyramid layer structure.
<a href="#">GetPyramidDownROI</a>	Computes the size of the lower pyramid layer.
<a href="#">GetPyramidUpROI</a>	Computes the size of the upper pyramid layer.
<a href="#">PyramidLayerDown</a>	Creates a lower pyramid layer.
<a href="#">PyramidLayerUp</a>	Creates an upper pyramid layer.
<b>Image Inpainting Functions</b>	
<a href="#">InpaintInitAlloc</a>	Allocates memory and initializes a structure for image inpainting.
<a href="#">InpaintFree</a>	Frees memory allocated for the structure for image inpainting.
<a href="#">Inpaint</a>	Restores unknown image pixels.
<b>Image Segmentation Functions</b>	
<a href="#">LabelMarkersGetBufferSize</a>	Computes the size of the working buffer for the marker labeling.
<a href="#">LabelMarkers</a>	Labels markers in image with different values.
<a href="#">SegmentWatershedGetBufferSize</a>	Computes the size of the working buffer for the watershed segmentation.
<a href="#">SegmentWatershed</a>	Performs watershed image segmentation using markers.
<a href="#">SegmentGradientGetBufferSize</a>	Computes the size of the working buffer for the gradient segmentation.
<a href="#">SegmentGradient</a>	Performs image segmentation by region growing to the least gradient direction.
<a href="#">BoundSegments</a>	Marks pixels belonging to segment boundaries.
<a href="#">ForegroundHistogramInitAlloc</a>	Allocates memory and initializes a state structure for foreground and background segmentation using histograms.
<a href="#">ForegroundHistogramFree</a>	Frees memory allocated for the foreground and background segmentation structure.
<a href="#">ForegroundHistogram</a>	Calculates foreground mask using histograms.
<a href="#">ForegroundHistogramUpdate</a>	Updates histogram statistical model for foreground segmentation.
<a href="#">ForegroundGaussianInitAlloc</a>	Allocates memory and initializes a state structure for foreground/background segmentation using Gaussian mixtures.
<a href="#">ForegroundGaussianFree</a>	Frees memory allocated for foreground/background segmentation using Gaussian mixtures.
<a href="#">ForegroundGaussian</a>	Performs foreground/background segmentation using Gaussian mixtures.
<b>Pattern Recognition Functions</b>	

Function Name	Description
<a href="#">HaarClassifierInitAlloc</a>	Allocates memory and initializes the structure for standard Haar classifiers.
<a href="#">TiltedHaarClassifierInitAlloc</a>	Allocates memory and initializes the structure for tilted Haar classifiers.
<a href="#">HaarClassifierFree</a>	Frees memory allocated for the Haar classifier structure.
<a href="#">GetHaarClassifierSize</a>	Returns the size of the Haar classifier.
<a href="#">TiltHaarFeatures</a>	Modifies a Haar classifier by tilting specified features.
<a href="#">ApplyHaarClassifier</a>	Applies a Haar classifier to an image.
<a href="#">ApplyMixedHaarClassifier</a>	Applies a mixed Haar classifier to an image.
<b>Camera Calibration and 3D Reconstruction</b>	
<a href="#">UndistortGetSize</a>	Computes the size of the external buffer.
<a href="#">UndistortRadial</a>	Corrects radial distortions of the single image.
<a href="#">CreateMapCameraUndistort</a>	Creates look-up tables of coordinates of corrected image.
<b>Image Enhancement Functions</b>	
<a href="#">SRHNInitAlloc_PSF3x3</a> , <a href="#">SRHNInitAlloc_PSF2x2</a>	Allocates memory and initializes the internal representation of the PSF table.
<a href="#">SRHNFree_PSF3x3</a> , <a href="#">SRHNFree_PSF2x2</a>	Deallocates the internal representation of the PSF table.
<a href="#">SRHNCalcResidual_PSF3x3</a> , <a href="#">SRHNCalcResidual_PSF2x2</a>	Computes residuals for the likelihood part of the target function or its gradient.
<a href="#">SRHNUpdateGradient_PSF3x3</a> , <a href="#">SRHNUpdateGradient_PSF2x2</a>	Updates the likelihood part of the gradient.

## UsingippiAdd for Background Differencing

This section describes functions that help build a statistical model of a background. This model can be used to subtract the background from an image.

Here, the term “background” stands for a set of motionless image pixels – that is, pixels that do not belong to any object, moving in front of the camera. The definition of background can vary if considered in other techniques of object extraction. For example, if the depth map of the scene can be obtained, for example, with the help of stereo, background can be determined as static parts of the scene that are located far enough from the camera.

The simplest background model assumes that every background pixel brightness varies independently, according to normal distribution. To calculate the characteristics of the background, several dozens of frames, as well as their squares, can be accumulated. That is, for every pixel location we find the sum of pixel values in this location  $S(x, y)$ , using the function [ippiAdd](#), and the sum of squares of the values  $Sq(x, y)$ , using the function [ippiAddSquare](#). Then mean is calculated as

$$m(x, y) = \frac{S(x, y)}{N},$$

where  $N$  is the number of collected frames, and standard deviation as

$$stddev(x, y) = \sqrt{\frac{Sq(x, y)}{N} - \left(\frac{S(x, y)}{N}\right)^2}$$

After that, the pixel in a certain pixel location within a certain frame is considered as belonging to a moving object, if the condition  $\text{abs}(p(x, y) - m(x, y)) < C * \text{stddev}(x, y)$ , where  $C$  is a constant, is met. If  $C$  is equal to 3, it satisfies the “three sigmas” rule. To obtain such background model, objects should be put away from the camera for a few seconds, so that the whole image from the camera represents the subsequent background observation.

Adapting the background differencing model to changes in lighting conditions and background scenes, for example, when the camera moves or an object passes behind the front object, can improve the described technique.

The mean brightness can be calculated through replacing the simple average with the running average found by using the function [ippiAddWeighted](#). Also, several techniques can be used to identify moving scene parts and exclude them while accumulating background information. These techniques include change detection (see the functions [ippiAbsDiff](#) in chapter 5 and [ippiThreshold](#) in chapter 7), optical flow, and some other operations.

Relevant addition functions used for background differencing include:

[Add\\_8u32f\\_C1IR](#), [Add\\_8s32f\\_C1IR](#), [Add\\_32f\\_C1IR](#),  
[Add\\_8u32f\\_C1IMR](#), [Add\\_8s32f\\_C1IMR](#), [Add\\_32f\\_C1IMR](#) (see [Add](#)),  
 and also all flavors of [AddSquare](#), [AddProduct](#), and [AddWeighted](#).

## Feature Detection Functions

This section describes feature detection functions.

The set of the Sobel derivative filters is generally used to find edges, ridges, and blobs, especially in case of scale-space images, for example, pyramids.

The following naming conventions are used in the equations described below:

- $D_x$  and  $D_y$  are the first  $x$  and  $y$  derivatives, respectively.
- $D_{xx}$  and  $D_{yy}$  are the second  $x$  and  $y$  derivatives, respectively.
- $D_{xy}$  is the partial  $x$  and  $y$  derivative.
- $D_{xxx}$  and  $D_{yyy}$  are the third  $x$  and  $y$  derivatives, respectively.
- $D_{xxy}$  and  $D_{xyy}$  are the third partial  $x$  and  $y$  derivatives.

### Corner Detection

The Sobel and Sharr first derivative operators are to be used to take the  $x$  and  $y$  derivatives of an image. Then a small region of interest (ROI) is to be defined to detect corners in.

A 2x2 matrix of sums of the  $x$  and  $y$  derivatives is created as follows:

$$\text{stddev}(x, y) = \text{sqrt}\left(\frac{Sq(x, y)}{N} - \left(\frac{S(x, y)}{N}\right)^2\right)$$

Solving

$$\det(C - \lambda I) = 0 ,$$



where  $\lambda$  is a column vector of the eigen values and  $I$  is the identity matrix, gives the eigen values. For the 2x2 matrix of the equation above, the solutions may be written in closed form:

$$\lambda = \frac{\Sigma D_x^2 + \Sigma D_y^2 \pm \sqrt{(\Sigma D_x^2 + \Sigma D_y^2)^2 - 4 \cdot (\Sigma D_x^2 \Sigma D_y^2 - (\Sigma D_x D_y)^2)}}{2}.$$

If  $\lambda_1, \lambda_2 > \epsilon$ , where  $\epsilon$  is some threshold, then a corner is considered to be found at that location. This can be very useful for object or shape recognition.

## Canny Edge Detector

This subsection describes a classic edge detector proposed by J.Canny, see [Canny86]. The detector uses a grayscale image as an input and outputs a black-and-white image, where non-zero pixels mark detected edges. The algorithm consists of three stages described below.

### Stage 1: Differentiation

Assuming two-dimensional convolution, the image data are differentiated with respect to the directions  $x$  and  $y$ . The gradient of the surface of the convoluted image function in any direction is possible to compute from the known gradient in any two directions.

From the computed  $x$  and  $y$  gradient values, the magnitude and angle of the slope can be calculated from the hypotenuse and arctangent.



**NOTE.** The `ippiSobel` functions perform the first stage and Canny edge detector functions use their output.

### Stage 2: Non-Maximum Suppression

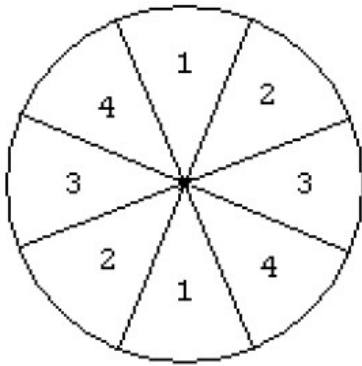
With the rate of intensity change found at each point in the image, edges must now be placed at the points of maximum, or rather non-maximum must be suppressed. A local maximum occurs at a peak in the gradient function, or alternatively where the derivative of the gradient function is set to zero. However, in this case it is preferable to suppress non-maximum perpendicular to the edge direction, rather than parallel to the edge direction, since the edge strength is expected to continue along an extended contour.

The algorithm starts off by reducing the angle of gradient to one of 4 sectors shown in Figure "Gradient Sectors". The algorithm passes 3x3 neighborhood across the magnitude array. At each point, the center element of neighborhood is compared with its two neighbors along line of the gradient given by the sector value.

If the central value is non-maximum, i.e., not greater than the neighbors, it is suppressed.

### Gradient Sectors

---



### Stage 3: Edge Thresholding

The Canny operator uses the so-called “hysteresis” thresholding. Most thresholders use a single threshold limit, which means that if the edge values fluctuate above and below this value, the line appears broken. This phenomenon is commonly referred to as “streaking”. Hysteresis counters streaking by setting an upper and lower edge value limit. Considering a line segment, if a value lies above the upper threshold limit it is immediately accepted. If the value lies below the low threshold it is immediately rejected. Points which lie between the two limits are accepted if they are connected to pixels which exhibit strong response. The likelihood of streaking is reduced drastically since the line segment points must fluctuate above the upper limit and below the lower limit for streaking to occur.

J.Canny recommends the ratio of high to low limit be in the range two or three to one, based on predicted signal-to-noise ratios.

**Example** shows how to use the Intel IPP functions for the Canny edge detector.

## CannyGetSize

*Calculates size of temporary buffer for function `ippiCanny`.*

### Syntax

```
IppStatus ippiCannyGetSize(IppiSize roiSize, int* pBufferSize);
```

### Parameters

<code>roiSize</code>	Size of the image ROI in pixels.
<code>pBufferSize</code>	Pointer to the computed size of the temporary buffer.

### Description

The function `ippiCannyGetSize` is declared in the `ippcv.h` file. This function calculates the size of a temporary buffer for the function `ippiCanny`.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <code>pBufferSize</code> pointer is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <code>pRoiSize</code> has a field with zero or negative value.

## Canny

*Implements Canny algorithm for edge detection.*

### Syntax

```
IppStatus ippCanny_16s8u_C1R(Ipp16s* pSrcDx, int srcDxStep, Ipp16s* pSrcDy, int srcDyStep,
Ipp8u* pDstEdges, int dstEdgeStep, IppiSize roiSize, Ipp32f lowThresh, Ipp32f highThresh,
Ipp8u* pBuffer);
```

```
IppStatus ippCanny_32f8u_C1R(Ipp32f* pSrcDx, int srcDxStep, Ipp32f* pSrcDy, int srcDyStep,
Ipp8u* pDstEdges, int dstEdgeStep, IppiSize roiSize, Ipp32f lowThresh, Ipp32f highThresh,
Ipp8u* pBuffer);
```

### Parameters

<code>pSrcDx</code>	Pointer to the source image ROI $x$ -derivative.
<code>srcDxStep</code>	Distance in bytes between starts of consecutive lines in the source image <code>pSrcDx</code> .
<code>pSrcDy</code>	Pointer to the source image ROI $y$ -derivative.
<code>srcDyStep</code>	Distance in bytes between starts of consecutive lines in the source image <code>pSrcDy</code> .
<code>pDstEdges</code>	Pointer to the output array of the detected edges.
<code>dstEdgeStep</code>	Distance in bytes between starts of consecutive lines in the output image.
<code>roiSize</code>	Size of the source image ROI in pixels.
<code>lowThresh</code>	Lower threshold for edges detection.
<code>highThresh</code>	Upper threshold for edges detection.
<code>pBuffer</code>	Pointer to the pre-allocated temporary buffer.

### Description

The function `ippCanny` is declared in the `ippcv.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)). This function finds edges in the source image ROI and stores them into the output image `pDstEdges` using the Canny algorithm. The function requires a temporary working buffer; its size should be computed previously by calling the function `ippCannyGetSize`.

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointers is <code>NULL</code> .

<code>ippStsSizeErr</code>	Indicates an error condition if <i>pRoiSize</i> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcDxStep</i> , <i>srcDyStep</i> or <i>dstEdgeStep</i> is less than <i>roi.width</i> * <i>&lt;pixelSize&gt;</i> .
<code>ippStsBadArgErr</code>	Indicates an error when <i>lowThresh</i> is negative or <i>highThresh</i> is less than <i>lowThresh</i> .
<code>ippStsNotEvenStepErr</code>	Indicates an error condition if one of the step values is not divisible by 2 for 16s images, and by 4 for 32f images.

### Example

The code example below shows how to use the function `ippiCanny_16s8u_C1R`.

```

IppStatus sts;

Ipp32f low=8.0f, high=12.0f;

int size, size1, srcStep, dxStep, dyStep, dstStep;

IppiSize roi;

Ipp8u *src, *dst, *buffer;

Ipp16s *dx, *dy;

...

sts = ippiFilterSobelNegVertGetBufferSize_8u16s_C1R(roi, ippMskSize3x3, &size);
sts = ippiFilterSobelHorizGetBufferSize_8u16s_C1R(roi, ippMskSize3x3, &size1);
if (size<size1) size=size1;
ippiCannyGetSize(roi, &size1);
if (size<size1) size=size1;
buffer = ippsMalloc_8u(size);

sts = ippiFilterSobelNegVertBorder_8u16s_C1R(src, srcStep, dx, dxStep, roi,
                                             ippMskSize3x3, ippBorderRepl, 0, buffer);

sts = ippiFilterSobelHorizBorder_8u16s_C1R(src, srcStep, dy, dyStep, roi,
                                             ippMskSize3x3, ippBorderRepl, 0, buffer);

sts = ippiCanny_16s8u_C1R(dx, dxStep, dy, dyStep, dst, dstStep, roi, low,
                           high, buffer);

ippsFree(buffer);

```

## EigenValsVecsGetBufferSize

Calculates size of temporary buffer for the function

`ippiEigenValsVecs`.

---

### Syntax

```

IppStatus ippiEigenValsVecsGetBufferSize_32f_C1R(IppiSize roiSize, int apertureSize, int
avgWindow, int* pBufferSize);

```

```
IppStatus ippiEigenValsVecsGetBufferSize_8u32f_C1R(IppiSize roiSize, int apertureSize,
int avgWindow, int* pBufferSize);
```

## Parameters

<i>roiSize</i>	Size of the source image ROI in pixels.
<i>apertureSize</i>	Size (pixels) of the derivative operator used by the function, possible values are 3 or 5.
<i>avgWindow</i>	Size of the blurring window in pixels, possible values are 3 or 5.
<i>pBufSize</i>	Pointer to the variable that returns the size of the temporary buffer.

## Description

The function `ippiEigenValsVecsGetBufferSize` is declared in the `ippcv.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function calculates the size of a temporary buffer to be used by the function [ippiEigenValsVecs](#).



**CAUTION.** The parameters *apertureSize* and *avgWindow* must be the same for both functions `ippiEigenValsVecsGetBufferSize` and `ippiEigenValsVecs`.

[Example](#) shows how to use the function `ippiEigenValsVecsGetBufferSize`.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiWidth</i> has zero or negative value, or if <i>apertureSize</i> or <i>avgWindow</i> has an illegal value.

## EigenValsVecs

*Calculates eigen values and eigen vectors of image blocks for corner detection.*

## Syntax

```
IppStatus ippiEigenValsVecs_8u32f_C1R(const Ipp8u* pSrc, int srcStep, Ipp32f* pEigenVV,
int eigStep, IppiSize roiSize, IppiKernelType kernType, int apertureSize, int avgWindow,
Ipp8u* pBuffer);
```

```
IppStatus ippiEigenValsVecs_32f_C1R(const Ipp32f* pSrc, int srcStep, Ipp32f* pEigenVV,
int eigStep, IppiSize roiSize, IppiKernelType kernType, int apertureSize, int avgWindow,
Ipp8u* pBuffer);
```

## Parameters

<i>pSrc</i>	Pointer to the source image ROI.				
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.				
<i>pEigenVV</i>	Image to store the results.				
<i>eigStep</i>	Distance in bytes between starts of consecutive lines in the output image.				
<i>roiSize</i>	Size of the source image ROI in pixels.				
<i>kernType</i>	Specifies the type of kernel used to compute derivatives, possible values are: <table data-bbox="503 703 1133 777"> <tr> <td><code>ippKernelSobel</code></td><td>Sobel kernel 3x3 or 5x5;</td></tr> <tr> <td><code>ippKernelScharr</code></td><td>Scharr kernel 3x3.</td></tr> </table>	<code>ippKernelSobel</code>	Sobel kernel 3x3 or 5x5;	<code>ippKernelScharr</code>	Scharr kernel 3x3.
<code>ippKernelSobel</code>	Sobel kernel 3x3 or 5x5;				
<code>ippKernelScharr</code>	Scharr kernel 3x3.				
<i>apertureSize</i>	Size of the derivative operator in pixels, possible values are 3 or 5.				
<i>avgWindow</i>	Size of the blurring window in pixels, possible values are 3 or 5.				
<i>pBuffer</i>	Pointer to the temporary buffer.				

## Description

The function `ippiEigenValsVecs` is declared in the `ippcv.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function takes a block around the pixel and computes the first derivatives  $D_x$  and  $D_y$ . This operation is performed for every pixel of the image using either Sobel or Scharr kernel in accordance with the *kernType* parameter. The size of the Sobel kernel may be specified by the parameter *apertureSize*. If this parameter is set to 3, the function used 3x3 kernel, if it is set to 5, the function uses 5x5 kernel. Only 3x3 size is available for the Scharr kernel, therefore the parameter *apertureSize* must be set to 3 if the Scharr kernel is used.



**CAUTION.** If the parameter *apertureSize* is set to 5 for operation with the Scharr kernel, the function returns error status.

Then, the function computes eigen values and vectors of the following matrix:

$$\begin{bmatrix} \Sigma D_x^2 & \Sigma D_x D_y \\ \Sigma D_x D_y & \Sigma D_y^2 \end{bmatrix}.$$

The summation is performed over the full block with averaging over the blurring window with size *avgWindow*.

The image *eigenVV* has the following format. For every pixel of the source image it contains six floating-point values -  $\lambda_1, \lambda_2, x_1, x_2, y_1, y_2$ . These values are defined as follows:

$\lambda_1, \lambda_2$	Eigen values of the above matrix ( $\lambda_1 \geq \lambda_2 \geq 0$ ).
$x_1, y_1$	Coordinates of the normalized eigen vector corresponding to $\lambda_1$ .
$x_2, y_2$	Coordinates of the normalized eigen vector corresponding to $\lambda_2$ .

In case of a singular matrix or when one eigen value is much smaller than the second one, all these six values are set to 0.

The function requires a temporary working buffer; its size should be computed previously by calling the function [ippiEigenValsVecsGetBufferSize](#).



**CAUTION.** The parameters *apertureSize* and *avgWindow* must be the same for both functions [ippiEigenValsVecsGetBufferSize](#) and [ippiEigenValsVecs](#).

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <i>pRoiSize</i> has a field with zero or negative value, or if <i>apertureSize</i> or <i>avgWindow</i> has an illegal value; or if <i>kernType</i> has wrong value.
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> is less than <i>roiSize.width*&lt;pixelSize&gt;</i> , or <i>eigStep</i> is less than <i>roiSize.width*sizeof(Ipp32f)*6</i> .
<code>ippStsNotEvenStepErr</code>	Indicates an error condition if steps for floating-point images are not divisible by 4.

### Example

The code example below shows how to use the function `ippiEigenValsVecs_8u32f_C1R`.

```

Ipp8u pSrc [4*3] = { 2, 17, 2, 21,
                    9,  4, 11,  5,
                    2, 32, 7,  2};

Ipp32f pEigenVV [24*3];
IppiSize roiSize = { 4, 3 };
int apertureSize = 3;
int avgWindow = 3;
int pBufferSize;

ippiEigenValsVecsGetBufferSize_8u32f_C1R ( roiSize, apertureSize, avgWindow, &pBufferSize );

Ipp8u* pBuffer = new Ipp8u [ pBufferSize ];

ippiEigenValsVecs_8u32f_C1R ( pSrc, 4, pEigenVV, 24*sizeof(Ipp32f), roiSize, ippiKernelSobel, apertureSize,
    avgWindow, pBuffer );

result:

2 17  2 21
9  4 11  5    pSrc
2 32  7  2

           pEigenVV

[0.3, 0.0, 0.9, 0.3, 0.3, -0.9] [0.2, 0.0, 1.0, 0.3, 0.3, -1.0] [0.3, 0.1,
-0.7, 0.8, -0.8, -0.7] [0.5, 0.1, -0.7, 0.7, -0.7, -0.7]

[0.6, 0.1, 1.0, 0.2, 0.2, -1.0] [0.5, 0.1, 1.0, 0.0, 0.0, -1.0] [0.4, 0.2,
-0.9, 0.4, -0.4, -0.9] [0.5, 0.2, -0.9, 0.5, -0.5, -0.9]

[0.9, 0.1, 1.0, 0.2, 0.2, -1.0] [0.9, 0.2, 1.0, 0.0, 0.0, -1.0] [0.5, 0.2,
-1.0, 0.1, -0.1, -1.0] [0.5, 0.2, -1.0, 0.2, -0.2, -1.0]

```

### MinEigenValGetBufferSize

*Calculates size of temporary buffer for the function*

`ippiMinEigenVal`.

---

#### Syntax

```

IppStatus ippiMinEigenValGetBufferSize_32f_C1R(IppiSize roiSize, int apertureSize, int
    avgWindow, int* pBufferSize);

IppStatus ippiMinEigenValGetBufferSize_8u32f_C1R(IppiSize roiSize, int apertureSize, int
    avgWindow, int* pBufferSize);

```

#### Parameters

<code>roiSize</code>	Size of the source image ROI in pixels.
----------------------	---



<i>apertureSize</i>	Size (in pixels) of the derivative operator used by the function, possible values are 3 or 5.
<i>avgWindow</i>	Size of the blurring window in pixels, possible values are 3 or 5.
<i>pBufferSize</i>	Pointer to the variable that returns the size of the temporary buffer.

## Description

The function `ippiMinEigenValGetBufferSize` is declared in the `ippcv.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)). This function calculates the size of a temporary buffer to be used by the function `ippiMinEigenVal`.



**CAUTION.** The parameters *apertureSize* and *avgWindow* must be the same for both functions `ippiMinEigenValGetBufferSize` and `ippiMinEigenVal`.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiWidth</i> has a zero or negative value, or if <i>apertureSize</i> or <i>avgWindow</i> has an illegal value.

## MinEigenVal

*Calculates the minimal eigen value of image blocks for corner detection.*

## Syntax

```
ippStatus ippiMinEigenVal_8u32f_C1R(const Ipp8u* pSrc, int srcStep, Ipp32f* pMinEigenVal,
int minValStep, IppiSize roiSize, IppiKernelType kernType, int apertureSize, int avgWindow,
Ipp8u* pBuffer);
```

```
ippStatus ippiMinEigenVal_32f_C1R(const Ipp32f* pSrc, int srcStep, Ipp32f* pMinEigenVal,
int minValStep, IppiSize roiSize, IppiKernelType kernType, int apertureSize, int avgWindow,
Ipp8u* pBuffer);
```

## Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pMinEigenVal</i>	Pointer to the image that is filled with the minimal eigen values.
<i>minValStep</i>	Distance in bytes between starts of consecutive lines in the output image.
<i>roiSize</i>	Size of the source image ROI in pixels.

<i>kernType</i>	Specifies the type of kernel used to compute derivatives, possible values are: ippKernelSobel          Sobel kernel 3x3 or 5x5; ippKernelScharr        Scharr kernel 3x3.
<i>apertureSize</i>	Size of the derivative operator in pixels, possible values are 3 or 5.
<i>avgWindow</i>	Size of the averaging window in pixels, possible values are 3 or 5.
<i>pBuffer</i>	Pointer to the temporary buffer.

### Description

The function `ippiMinEigenVal` is declared in the `ippcv.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)). This function takes a block around the pixel and computes the first derivatives  $D_x$  and  $D_y$ . This operation is performed for every pixel of the image using either Sobel or Scharr kernel in accordance with the *kernType* parameter. The size of the Sobel kernel may be specified the parameter *apertureSize*. If this parameter is set to 3, the function used 3x3 kernel, if it is set to 5, the function uses 5x5 kernel. Only 3x3 size is available for the Scharr kernel, therefore the parameter *apertureSize* must be set to 3 if the Scharr kernel is used.



**CAUTION.** If the parameter *apertureSize* is set to 5 for operation with the Scharr kernel, the function returns error status.

Then, the function computes the minimal eigen value  $\lambda$  ( $\lambda \geq 0$ ) of the following matrix:

$$\begin{bmatrix} \Sigma D_x^2 & \Sigma D_x D_y \\ \Sigma D_x D_y & \Sigma D_y^2 \end{bmatrix}.$$

The summation is performed over the full block with averaging over the blurring window with size *avgWindow*. The function requires a temporary working buffer; its size should be computed previously by calling the function `ippiMinEigenValGetBufferSize`.



**CAUTION.** The parameters *apertureSize* and *avgWindow* must be the same for both functions `ippiMinEigenValGetBufferSize` and `ippiMinEigenVal`.

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointers is <code>NULL</code> .

<code>ippStsSizeErr</code>	Indicates an error condition if <code>pRoiSize</code> has a field with zero or negative value, or if <code>apertureSize</code> or <code>avgWindow</code> has an illegal value; or if <code>kernType</code> has wrong value.
<code>ippStsStepErr</code>	Indicates an error condition if <code>srcStep</code> is less than <code>roiSize.width*&lt;pixelSize&gt;</code> , or <code>eigenvvStep</code> is less than <code>roiSize.width*sizeof(Ipp32f)</code> .
<code>ippStsNotEvenStepErr</code>	Indicates an error condition if steps for floating-point images are not divisible by 4.

## Hough Transform

The Hough transform is a general technique that allows to detect the flat curves in the binarised images [Gon93]. The current version of Intel IPP implements the detecton of the straight lines that are defined by the parametric equation

$r = x \cdot \cos(\Theta) + y \cdot \sin(\Theta)$ , where  $r$  and  $\Theta$  are the length and angle from the origin of a normal to the line respectively.

## HoughLineGetSize

*Computes the size of the working buffer for the straight lines detection.*

### Syntax

```
ippStatus ippHoughLineGetSize_8u_C1R(IppiSize roiSize, IppPointPolar delta, int
maxLineCount, int* pBufferSize);
```

### Parameters

<code>roiSize</code>	Size of the source image ROI in pixels.
<code>delta</code>	Step of discretization ( <code>delta.rho</code> - radial increment, <code>delta.theta</code> - angular increment).
<code>maxLineCount</code>	Number of elements of the destination buffer.
<code>pBufferSize</code>	Pointer to the size of the working buffer.

### Description

The function `ippHoughLineGetSize` is declared in the `ippcv.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function computes the size of the temporary working buffer that is required for the function `ippHoughLine`.

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error if <code>pBufferSize</code> is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <code>roiSize</code> or <code>delta</code> has a field with zero or negative value.

## HoughLine

*Detects straight lines in the source image.*

---

### Syntax

```
IppStatus ippiHoughLine_8u32f_C1R(const Ipp8u* pSrc, int srcStep, IppiSize roiSize,
IppPointPolar delta, int threshold, IppPointPolar* pLine, int maxLineCount, int* pLineCount,
Ipp8u* pBuffer);
```

### Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>roiSize</i>	Size of source image ROI in pixels.
<i>delta</i>	Step of discretization ( <i>delta.rho</i> - radial increment, <i>delta.theta</i> - angular increment).
<i>threshold</i>	Minimum number of points that are required to detect the line.
<i>pLine</i>	Pointer to the destination buffer for lines.
<i>maxLineCount</i>	Number of elements of the destination buffer.
<i>pBuffer</i>	Pointer to the working buffer.

### Description

The function `ippiHoughLine` is declared in the `ippcv.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

In the binarised source image *pSrc*, the function performs detection of the straight line defined by the [equation](#) given at the beginning of section Hough Transform. The level of discretization *delta* is specified as the input parameters. The performance and effectiveness of the function is strongly depends on this parameter. The function requires the external working buffer *pBuffer* whose size should be computed previously using the function [ippiHoughLineGetSize](#).



---

**CAUTION.** The value of the parameter *delta* must not be greater than the value of the parameter *delta* set when the size of the working buffer is computed.

---

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error if one of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value; or if <i>maxLineCount</i> is less than or equal to 0.
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> has an illegal value.

<code>ippStsBadArgErr</code>	Indicates an error condition if <i>threshold</i> is less than or equal to 0; or <i>delta.rho</i> is less than 0, or greater than sum of the width and height of the ROI; or <i>delta.theta</i> is less than 0, or greater than <i>p</i> .
<code>ippStsDstSizeLessExpected</code>	Indicates a warning if number of the detected lines is greater than the size of the destination buffer <i>maxLineCount</i> .

## HoughLine\_Region

*Detects straight lines with the specified range of parameters in the source image.*

### Syntax

```
ippStatus ippiHoughLine_Region_8u32f_C1R(const Ipp8u* pSrc, int srcStep, IppiSize roiSize,
IppPointPolar* pLine, IppPointPolar dstRoi[2], int maxLineCount, int* pLineCount,
IppPointPolar delta, int threshold, Ipp8u* pBuffer);
```

### Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>roiSize</i>	Size of source image ROI in pixels.
<i>pLine</i>	Pointer to the destination array of detected lines.
<i>dstRoi</i>	Specifies the range of parameters of straight lines to be detected.
<i>pLineCount</i>	Pointer to the number of detected lines.
<i>delta</i>	Step of discretization ( <i>delta.rho</i> - radial increment, <i>delta.theta</i> - angular increment).
<i>threshold</i>	Minimum number of points that are required to detect the line.
<i>maxLineCount</i>	Maximum number of lines in the destination buffer.
<i>pBuffer</i>	Pointer to the working buffer.

### Description

The function `ippiHoughLine_Region` is declared in the `ippcv.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

In the binarised source image *pSrc*, this function performs detection of the straight line defined by the [equation](#) given at the beginning of section Hough Transform. Only lines *line[n]* with the parameters satisfying the following conditions are detected:

$$dstRoi[0].rho \leq line[n].rho \leq dstRoi[1].rho;$$

$$dstRoi[0].theta \leq line[n].theta \leq dstRoi[1].theta;$$

where  $n = 0..pLineCount$ .

The level of discretization *delta* is specified as the input parameters. The performance and effectiveness of the function is strongly depends on this parameter. The function requires the external working buffer *pBuffer* whose size should be computed previously using the function [ippiHoughLineGetSize](#).



**CAUTION.** The value of the parameter *delta* must not be greater than the value of the parameter *delta* set when the size of the working buffer is computed.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error if one of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value; or if <i>maxLineCount</i> is less than or equal to 0.
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> has an illegal value.
<code>ippStsBadArgErr</code>	Indicates an error condition if <i>threshold</i> is less than or equal to 0; or <i>delta.rho</i> is less than 0, or greater than sum of the width and height of the ROI; or <i>delta.theta</i> is less than 0, or greater than <i>p</i> ; or some filed of <i>dstRoi[0]</i> is greater than the corresponding filed of <i>dstRoi[1]</i> .
<code>ippStsDstSizeLessExpected</code>	Indicates a warning if number of the detected lines is greater than the size of the destination buffer <i>maxLineCount</i> .

## Distance Transform Functions

This section describes the distance transform functions.

Distance transform is used for calculating the distance to an object. The input is an image with feature and non-feature pixels. The function labels every non-feature pixel in the output image with a distance to the closest feature pixel. Feature pixels are marked with zero.

Distance transform is used for a wide variety of subjects including skeleton finding and shape analysis.

### DistanceTransform

*Calculates distance to the closest zero pixel for all non-zero pixels of source image.*

#### Syntax

##### Case 1: Not-in-place operations

```
IppStatus ippiDistanceTransform_3x3_<mod>(const Ipp8u* pSrc, int srcStep, Ipp<dstDatatype>* pDst, int dstStep, IppiSize roiSize, Ipp32s* pMetrics);
```

```
IppStatus ippiDistanceTransform_5x5_<mod>(const Ipp8u* pSrc, int srcStep, Ipp<dstDatatype>*
pDst, int dstStep, IppiSize roiSize, Ipp32s* pMetrics);
```

Supported values for mod:

```
8u_C1R          8u16u_C1R
```

```
IppStatus ippiDistanceTransform_3x3_8u32f_C1R(const Ipp8u* pSrc, int srcStep, Ipp32f*
pDst, int dstStep, IppiSize roiSize, Ipp32f* pMetrics);
```

```
IppStatus ippiDistanceTransform_5x5_8u32f_C1R(const Ipp8u* pSrc, int srcStep, Ipp32f*
pDst, int dstStep, IppiSize roiSize, Ipp32f* pMetrics);
```

### Case 2: In-place operations

```
IppStatus ippiDistanceTransform_3x3_8u_C1IR(Ipp8u* pSrcDst, int srcDstStep, IppiSize
roiSize, Ipp32s* pMetrics);
```

```
IppStatus ippiDistanceTransform_5x5_8u_C1IR(Ipp8u* pSrcDst, int srcDstStep, IppiSize
roiSize, Ipp32s* pMetrics);
```

### Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the ROI in the destination distance image.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>pSrcDst</i>	Pointer to the source and destination image ROI for in-place operation.
<i>srcDstStep</i>	Distance in bytes between starts of consecutive lines in the source and destination image for in-place operation.
<i>roiSize</i>	Size of the image ROI in pixels.
<i>pMetrics</i>	Pointer to the array that specifies used metrics.

### Description

The distance transform function `ippiDistanceTransform` is declared in the `ippcv.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function approximates the actual distance from the closest zero pixel to each certain pixel with the sum of atomic distances from the fixed set. The set consists of two values for a 3x3 mask and three values for a 5x5 mask.

[Figure “3x3 Mask”](#) shows the result of the distance transform of a 7x7 image with zero point in the center. This example corresponds to a 3x3 mask. Two numbers specify metrics in case of the 3x3 mask:

- distance between two pixels that share an edge,
- distance between the pixels that share a corner.

In this case the values are 1 and 1.5 correspondingly.

**3x3 Mask**

4.5	4	3.5	3	3.5	4	4.5
4	3	2.5	2	2.5	3	4
3.5	2.5	1.5	1	1.5	2.5	3.5
3	2	1	0	1	2	3
3.5	2.5	1.5	1	1.5	2.5	3.5
4	3	2.5	2	2.5	3	4
4.5	4	3.5	3	3.5	4	4.5

Figure “5x5 Mask” shows the distance transform for the same image, but for a 5x5 mask.

For this mask yet another number is added to specify metrics - the additional distance, i.e., the distance between pixels corresponding to the chess knight move.

In this example, the additional distance is equal to 2.

**5x5 Mask**

4	3.5	3	3	3	3.5	4
3.5	3	2	2	2	3	3.5
3	2	1.5	1	1.5	2	3
3	2	1	0	1	2	3
3	2	1.5	1	1.5	2	3
3.5	3	2	2	2	3	3.5
4	3.5	3	3	3	3.5	4

**Return Values**

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <code>roiSize</code> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <code>srcStep</code> or <code>dstStep</code> is less than <code>roiSize.width*&lt;pixelSize&gt;</code> .
<code>ippStsNotEvenStepErr</code>	Indicates an error condition if step value is not divisible by 2 for 16u images, and by 4 for 32f images.
<code>ippStsCoeffErr</code>	Indicates an error condition if at least one element of <code>pMetrics</code> array has zero or negative value.



## Example

The code example below shows how to use the function `ippiDistanceTransform_3x3_8u_C1R`.

```

Ipp8u
src[7*7] = {
    1, 2, 3, 4, 5, 6, 7,
    1, 0, 3, 4, 5, 6, 7,
    1, 2, 3, 4, 5, 6, 7,
    1, 2, 3, 0, 5, 6, 7,
    1, 2, 3, 4, 5, 6, 7,
    1, 2, 3, 4, 5, 0, 7,
    1, 2, 3, 4, 5, 6, 7
};

Ipp8u dst[7*7];
IppiSize roiSize = { 7, 7 };
Ipp32s pMetrics[2] = { 2, 2 };

ippiDistanceTransform_3x3_8u_C1R ( src, 7, dst, 7, roiSize, pMetrics );
result:
1 2 3 4 5 6 7
1 0 3 4 5 6 7
1 2 3 4 5 6 7
1 2 3 0 5 6 7    src
1 2 3 4 5 6 7
1 2 3 4 5 0 7
1 2 3 4 5 6 7

2 2 2 4 6 6 6
2 0 2 4 4 4 6
2 2 2 2 2 4 6
4 4 2 0 2 4 4    dst
6 4 2 2 2 2 2
6 4 4 4 2 0 2
6 6 6 4 2 2 2

```

## GetDistanceTransformMask

Returns an optimal mask for a given type of metrics and given mask size.

### Syntax

```
IppStatus ippiGetDistanceTransformMask_<mod>(int kerSize, IppiNorm norm, Ipp<datatype>* pMetrics);
```

Supported values for mod:

32s                      32f

### Parameters

<i>kerSize</i>	Specifies the mask size as follows: 3 for 3x3 mask, 5 for 5x5 mask.
<i>norm</i>	Specifies the type of metrics. Possible values are: ippiNormInf(0) $L_{\infty}, \Delta = \max( x_1 - x_2 ,  y_1 - y_2 ),$ ippiNormL1(1) $L_1, \Delta =  x_1 - x_2  +  y_1 - y_2 ,$ ippiNormL2(2) $L_2, \Delta = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$
<i>pMetrics</i>	Pointer to the output array to store metrics parameters. The array contains the following number of elements: 2                          for 3x3 mask, 3                          for 5x5 mask.

### Description

The function ippiGetDistanceTransformMask is declared in the `ippcv.h` file. This function fills up the output array with metrics parameters for the given type of metrics and size of mask. The function returns the following results:

(1, 1)	$L_{\infty}$ , 3x3 mask,
(1, 2)	$L_1$ , 3x3 mask,
(2, 3)	$L_2$ , 3x3 mask, 32s data type,
(0.955, 1.3693)	$L_2$ , 3x3 mask, 32f data type,
(1, 1, 2)	$L_{\infty}$ , 5x5 mask,
(1, 2, 3)	$L_1$ , 5x5 mask,
(4, 6, 9)	$L_2$ , 5x5 mask, 32s data type,
(1.0, 1.4, 2.1969)	$L_2$ , 5x5 mask, 32f data type.

For more information, see [Bor86].



**NOTE.** For compatibility with the previous versions of the library the earlier function ippiGetDistanceTransformMask replaced by the function ippiGetDistanceTransformMask\_32f in the current version is also supported.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pMetrics</i> pointer is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <i>kerSize</i> has a wrong value.
<code>ippStsBadArgErr</code>	Indicates an error condition if <i>kerSize</i> or <i>norm</i> has a wrong value.

## TrueDistanceTransformGetBufferSize

*Calculates the size of the temporary working buffer for the function `ippiTrueDistanceTransform`.*

---

### Syntax

```
ippStatus ippiTrueDistanceTransformGetBufferSize_8u16u_C1RSfs(IppiSize roiSize, int*
pBufferSize);

ippStatus ippiTrueDistanceTransformGetBufferSize_8u32f_C1R(IppiSize roiSize, int*
pBufferSize);
```

### Parameters

<i>roiSize</i>	Size of the image ROI in pixels.
<i>pBufferSize</i>	Pointer to the computed size of the buffer.

### Description

The function `ippiTrueDistanceTransformGetBufferSize` is declared in the `ippcv.h`. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function computes the size of the working buffer required for the function [ippiTrueDistanceTransform](#).




---

**CAUTION.** THE FUNCTION `ippiTrueDistanceTransformGetBufferSize_8u16u_C1RSfs` IS DEPRECATED. Please use the function `ippiTrueDistanceTransformGetBufferSize_8u32f_C1R` instead.

---

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pBufferSize</i> pointer is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.

## TrueDistanceTransform

*Calculates the Euclidian distance to the closest zero pixel for all non-zero pixels of the source image.*

### Syntax

```
IppStatus ippiTrueDistanceTransform_8u32f_C1R(const Ipp8u* pSrc, int srcStep, Ipp32f* pDst, int dstStep, IppiSize roiSize, Ipp8u* pBuffer);
```

```
IppStatus ippiTrueDistanceTransform_8u16u_C1RSfs(const Ipp8u* pSrc, int srcStep, Ipp16u* pDst, int dstStep, IppiSize roiSize, int scaleFactor, Ipp8u* pBuffer);
```

### Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the ROI in the destination distance image.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>roiSize</i>	Size of the image ROI in pixels.
<i>pBuffer</i>	Pointer to the temporary working buffer.
<i>scaleFactor</i>	Scale factor (see <a href="#">Integer Result Scaling</a> ).

### Description

The function `ippiTrueDistanceTransform` is declared in the `ippcv.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function calculates the Euclidian distance to the closest zero pixel for all non-zero pixels of the source image [Felz04].

Figure "True Distance Transform, *scaleFactor*=-5" shows the result of the integer version of the true distance transform of a 7x7 image with zero point in the center and with the scale factor -5.

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> or <i>dstStep</i> is less than <i>roiSize.width</i> *<pixelSize>.
<code>ippStsNotEvenStepErr</code>	Indicates an error condition if step value is not divisible by 2 for 16u images, and by 4 for 32f images.

### True Distance Transform, *scaleFactor*=-5

136	115	101	96	101	115	136
115	91	72	64	72	91	115

101	72	45	36	45	72	101
96	64	36	0	36	64	96
101	72	45	36	45	72	101
115	91	72	64	72	91	115
136	115	101	96	101	115	136

## FastMarchingGetBufferSize

*Computes the size of the working buffer for the peak search.*

### Syntax

```
IppStatus ippiFastMarchingGetBufferSize_8u32f_C1R(int roiWidth, int* pBufferSize);
```

### Parameters

<i>roiWidth</i>	Maximum image size (in pixels).
<i>pBufferSize</i>	Pointer to the computed size of the buffer.

### Description

The function `ippiFastMarchingGetBufferSize` is declared in the `ippcv.h`. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function computes the size of the working buffer required for the `ippiFastMarching` function. The buffer with the length `pBufferSize [0]` can be used to filter images with width that is equal to or less than the parameter `roiWidth` specified for the function `ippiFastMarching`.

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <code>pBufferSize</code> pointer is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <code>roiWidth</code> is less than 1.

## FastMarching

*Calculates distance transform to closest zero pixel for all non-zero pixels of source image using fast marching method.*

### Syntax

```
IppStatus ippiFastMarching_8u32f_C1R(const Ipp8u* pSrc, int srcStep, Ipp32f* pDst, int dstStep, IppiSize roiSize, Ipp32f radius, Ipp8u* pBuffer);
```

### Parameters

<i>pSrc</i>	Pointer to the source image ROI.
-------------	----------------------------------

<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>roiSize</i>	Size of the source and destination image ROI.
<i>raduis</i>	Radius of the neighborhood of the marked area.
<i>pBuffer</i>	Pointer to the working buffer.

### Description

The function *ippiFastMarching* is declared in the *ippcv.h*. It operates with ROI (see Regions of Interest in Intel IPP).

This function computes the distance from the closest zero pixel to each image pixel according to the Fast Marching Method (FMM) [Telea04]. The FMM distance for area  $\Omega$  with the border  $\partial\Omega$  is a solution of the equations:

$$(|\nabla T(x, y)| = 1), \{x, y\} \in \Omega$$

$$(T(x, y) = 0), \{x, y\} \in \partial\Omega$$

The resulting distance complies with the equation

$$T(x, y) = \min \left( \frac{T(u_1, v_1) + T(u_2, v_2) + \sqrt{2 - (T(u_1, v_1) - T(u_2, v_2))^2}}{2}, \min(T(u_1, v_1), T(u_2, v_2)) + 1 \right)$$

Here  $\{u_1, v_1\}$  and  $\{u_2, v_2\}$  are coordinates for pair of pixels adjacent to the pixel with  $\{x, y\}$  coordinates.

The area  $\Omega$  is defined by the non-zero pixel of the image *pSrc*. If *raduis* is positive, then the FMM distance with the negative sign is calculated in Euclidean *raduis*-neighborhood of  $\Omega$ .

The function requires the working buffer *pBuffer* whose size should be computed by the function *ippiFastMarchingGetBufferSize* beforehand.

Figure "Result of the FFM Method" shows the result of the fast marching method for the 7x9 image with centered 3x5 non-zero mask and *raduis*=1.

### Result of the FFM Method

0.0000	0.0000	-1.0000	-1.0000	-1.0000	-1.0000	-1.0000	0.0000	0.0000
0.0000	0.7071	-1.e-10	-1.e-10	-1.e-10	-1.e-10	-1.e-10	0.7071	-1.0000
-1.0000	-1.e-10	0.7071	0.9659	0.9994	0.9659	0.7071	-1.e-10	-1.0000
-1.0000	-1.e-10	0.9659	1.6730	1.9579	1.6730	0.9659	-1.e-10	-1.0000
-1.0000	-1.e-10	0.7071	0.9659	0.9994	0.9659	0.7071	-1.e-10	-1.0000
0.0000	0.7071	-1.e-10	-1.e-10	-1.e-10	-1.e-10	-1.e-10	0.7071	-1.0000
0.0000	-1.0000	-1.0000	-1.0000	-1.0000	-1.0000	-1.0000	-1.0000	0.0000

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <code>roiSize</code> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <code>srcStep</code> or <code>dstStep</code> is less than <code>roiSize.width * &lt; pixelSize &gt;</code> .
<code>ippStsNotEvenStepErr</code>	Indicates an error condition if the step value is not divisible by 4 for floating-point images.
<code>ippStsBadArgErr</code>	Indicates an error condition if <code>raduis</code> is negative.

## Image Gradients

### GradientColorToGray

Converts a color gradient image to grayscale.

#### Syntax

```
ippStatus ippiGradientColorToGray_<mod>(const Ipp<datatype>* pSrc, int srcStep,
Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, IppiNorm norm);
```

Supported values for `mod`:

`8u_C3C1R`      `16u_C3C1R`      `32f_C1R`

#### Parameters

<code>pSrc</code>	Pointer to the source image ROI.						
<code>srcStep</code>	Distance in bytes between starts of consecutive lines in the source image.						
<code>pDst</code>	Pointer to the destination image ROI.						
<code>dstStep</code>	Distance in bytes between starts of consecutive lines in the destination image.						
<code>roiSize</code>	Size of the source and destination image ROI.						
<code>norm</code>	Type of norm to form the mask for dilation; following values are possible: <table data-bbox="678 1642 1144 1753"> <tr> <td><code>ippiNormInf</code></td><td>Infinity norm.</td></tr> <tr> <td><code>ippiNormL1</code></td><td>L1 norm.</td></tr> <tr> <td><code>ippiNormL2</code></td><td>L2 norm.</td></tr> </table>	<code>ippiNormInf</code>	Infinity norm.	<code>ippiNormL1</code>	L1 norm.	<code>ippiNormL2</code>	L2 norm.
<code>ippiNormInf</code>	Infinity norm.						
<code>ippiNormL1</code>	L1 norm.						
<code>ippiNormL2</code>	L2 norm.						

## Description

The function `ippiGradientColorToGray` is declared in the `ippcv.h`. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function creates the grayscale gradient image  $pDst$  from the source three-channel gradient image  $pSrc$ . The type of norm is specified by the parameter. Pixel values for destination image are computed for different type of norm in accordance with the following formula:

$$dst_{i,j} = \begin{cases} \max\{|src_{i,j,0}|, |src_{i,j,1}|, |src_{i,j,2}|\} & norm = \text{ippiNormInf} \\ |src_{i,j,0}| + |src_{i,j,1}| + |src_{i,j,2}| & norm = \text{ippiNormL1} \\ \sqrt{src_{i,j,0}^2 + src_{i,j,1}^2 + src_{i,j,2}^2} & norm = \text{ippiNormL2} \end{cases}$$

For integer flavors the result is scaled to the full range of the destination data type.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <code>roiSize</code> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <code>srcStep</code> or <code>dstStep</code> is less than <code>roiSize.width * &lt;pixelSize&gt;</code> .
<code>ippStsNotEvenStepErr</code>	Indicates an error condition if one of the step values is not divisible by 2 for integer images, or by 4 for floating-point images.
<code>ippStsBadArgErr</code>	Indicates an error condition if <code>norm</code> has an illegal value.

## Flood Fill Functions

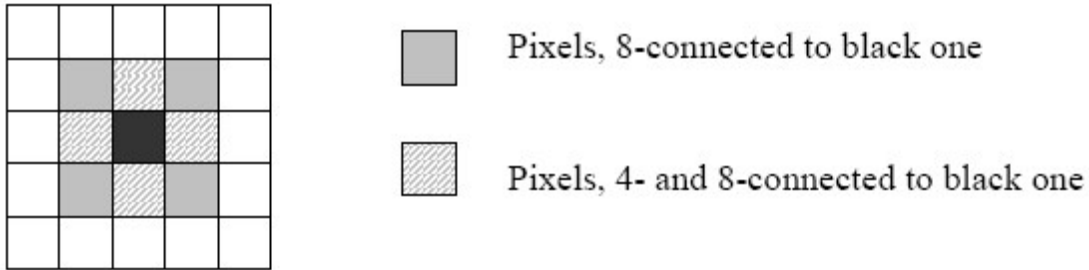
This section describes functions performing flood filling of connected areas. *Flood filling* means that a group of connected pixels with close values is filled with, or is set to, a certain value. The flood filling process starts with a specified point ("seed") and continues until it reaches the image ROI boundary or cannot find any new pixels to fill due to a large difference in pixel values. For every pixel filled, the functions analyze neighbor pixels:

- 4 neighbors (except diagonal neighbors); this kind of connectivity is called 4-connectivity and the corresponding function name includes `4Con`, or



- 8 neighbors (diagonal neighbors included); this kind of connectivity is called 8-connectivity and the corresponding function name includes 8Con.

### Pixels Connectivity Patterns



These functions can be used for:

- segmenting a grayscale image into a set of uni-color areas,
- marking each connected component with individual color for bi-level images.

### FloodFillGetSize

*Calculates size of temporary buffer for flood filling operation.*

#### Syntax

```
IppStatus ippiFloodFillGetSize(IppiSize roiSize, int* pBufferSize);
```

#### Parameters

<i>roiSize</i>	Size of the source image ROI in pixels.
<i>pBufferSize</i>	Pointer to the variable that returns the size of the temporary buffer.

#### Description

The function `ippiFloodFillGetSize` is declared in the `ippcv.h` file. This function calculates the size of the temporary buffer to be used by the function `ippiFloodFill`.

#### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pBufferSize</i> pointer is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.

## FloodFillGetSize\_Grad

*Calculates size of temporary buffer for the gradient flood filling.*

---

### Syntax

```
IppStatus ippiFloodFillGetSize_Grad(IppiSize roiSize, int* pBufferSize);
```

### Parameters

<i>roiSize</i>	Size of the source image ROI in pixels.
<i>pBufferSize</i>	Pointer to the variable that returns the size of the temporary buffer.

### Description

The function `ippiFloodFillGetSize_Grad` is declared in the `ippcv.h` file. This function calculates the size of the temporary buffer to be used by the function [ippiFloodFill\\_Grad](#).

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pBufferSize</i> pointer is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.

## FloodFill

*Performs flood filling of connected area.*

---

### Syntax

#### Case 1: Operations on one-channel data

```
IppStatus ippiFloodFill_4Con_<mod>(Ipp<datatype>* pImage, int imageStep, IppiSize roiSize, IppiPoint seed, Ipp<datatype> newVal, IppiConnectedComp* pRegion, Ipp8u* pBuffer);
```

```
IppStatus ippiFloodFill_8Con_<mod>(Ipp<DataType>* pImage, int imageStep, IppiSize roiSize, IppiPoint seed, Ipp<datatype> newVal, IppiConnectedComp* pRegion, Ipp8u* pBuffer);
```

Supported values for `mod`:

8u\_C1IR      16u\_C1IR      32f\_C1IR

#### Case 2: Operations on three-channel data

```
IppStatus ippiFloodFill_4Con_<mod>(Ipp<datatype>* pImage, int imageStep, IppiSize roiSize, IppiPoint seed, Ipp<datatype>* pNewVal, IppiConnectedComp* pRegion, Ipp8u* pBuffer);
```

```
IppStatus ippiFloodFill_8Con_<mod>(Ipp<DataType>* pImage, int imageStep, IppiSize roiSize,
IppiPoint seed, Ipp<datatype>* pNewVal, IppiConnectedComp* pRegion, Ipp8u* pBuffer);
```

Supported values for `mod`:

8u\_C3IR                      32f\_C3IR

## Parameters

<i>pImage</i>	Pointer to the ROI in the source and destination image (for the in-place operation).
<i>imageStep</i>	Distance in bytes between starts of consecutive lines in the image buffer.
<i>roiSize</i>	Size of the image ROI in pixels.
<i>seed</i>	Initial point.
<i>newVal</i>	Value to fill with for one-channel data.
<i>pNewVal</i>	Pointer to the vector containing values to fill with for three-channel data.
<i>pRegion</i>	Pointer to the connected components structure that stores information about the refilled area.
<i>pBuffer</i>	Pointer to the temporary buffer.

## Description

The function `ippiFloodFill` is declared in the `ippcv.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function performs [flood filling](#) (see [Flood Filling Functions](#)) of the group of connected pixels whose pixel values are equal to the value in the *seed* point. Values of these pixel is set to the *newVal* value.

The function requires a temporary buffer whose size should be computed with the function `ippiFloodFillGetSize` beforehand.

The functions with the “\_4con” suffixes check 4-connected neighborhood of each pixel, that is, side neighbors. The functions with the “\_8con” suffixes check 8-connected neighborhood of each pixel, that is, side and corner neighbors. See [Figure Pixels Connectivity Patterns](#)”.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <i>imageStep</i> is less than <code>pRoiSize.width * &lt;pixelSize&gt;</code> .
<code>ippStsNotEvenStepErr</code>	Indicates an error condition if steps for floating-point images are not divisible by 4, or steps for 16-bit integer images are not divisible by 2.

`ippStsOutOfRangeErr`

Indicates an error condition if the *seed* point is out of ROI.

## FloodFill\_Grad

*Performs gradient flood filling of connected area on an image.*

---

### Syntax

#### Case 1: Operations on one-channel data

```
IppStatus ippifloodfill_grad4Con_<mod>(Ipp<DataType>* pImage, int imageStep, IppiSize roiSize, IppiPoint seed, Ipp<datatype> newVal, Ipp<datatype> minDelta, Ipp<datatype> maxDelta, IppiConnectedComp* pRegion, Ipp8u* pBuffer);
```

```
IppStatus ippifloodfill_grad8Con_<mod>(Ipp<DataType>* pImage, int imageStep, IppiSize roiSize, IppiPoint seed, Ipp<datatype> newVal, Ipp<datatype> minDelta, Ipp<datatype> maxDelta, IppiConnectedComp* pRegion, Ipp8u* pBuffer);
```

Supported values for `mod`:

`8u_C1IR`      `16u_C1IR`      `32f_C1IR`

#### Case 2: Operations on three-channel data

```
IppStatus ippifloodfill_grad4Con_<mod>(Ipp<DataType>* pImage, int imageStep, IppiSize roiSize, IppiPoint seed, Ipp<datatype>* pNewVal, Ipp<datatype>* pMinDelta, Ipp<datatype>* pMaxDelta, IppiConnectedComp* pRegion, Ipp8u* pBuffer);
```

```
IppStatus ippifloodfill_grad8Con_<mod>(Ipp<DataType>* pImage, int imageStep, IppiSize roiSize, IppiPoint seed, Ipp<datatype>* pNewVal, Ipp<datatype>* pMinDelta, Ipp<datatype>* pMaxDelta, IppiConnectedComp* pRegion, Ipp8u* pBuffer);
```

Supported values for `mod`:

`8u_C3IR`      `32f_C3IR`

### Parameters

<i>pImage</i>	Pointer to the ROI in the source and destination image (in-place operation).
<i>imageStep</i>	Distance in bytes between starts of consecutive lines in the image buffer.
<i>roiSize</i>	Size of the image ROI in pixels.
<i>seed</i>	Initial point.
<i>minDelta</i>	Minimum difference between neighbor pixels for one-channel data.
<i>maxDelta</i>	Maximum difference between neighbor pixels for one-channel data.
<i>newVal</i>	Value to fill with for one-channel data.

<i>pMinDelta</i>	Pointer to the minimum differences between neighbor pixels for three-channel images.
<i>pMaxDelta</i>	Pointer to the maximum differences between neighbor pixels for three-channel images.
<i>pNewVal</i>	Pointer to the vector containing values to fill with for three-channel data.
<i>pRegion</i>	Pointer to the connected components structure that stores information about the refilled area.
<i>pBuffer</i>	Pointer to the temporary buffer.

## Description

The function `ippiFloodFill_Grad` is declared in the `ippcv.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function performs [flood filling](#) (see [Flood Filling Functions](#)) of the group of connected pixels in the *seed* pixel neighborhoods whose pixel values  $v$  satisfy the following conditions:

$$v_0 - d_{lw} \leq v \leq v_0 + d_{up},$$

where  $v_0$  is the value of at least one of the current pixel neighbors, which already belongs to the refilled area, and  $d_{lw}$ ,  $d_{up}$  are *minDelta*, *maxDelta*, respectively. Values of these pixels are set to the *newVal* value.

The function requires a temporary buffer whose size should be computed with the function `ippiFloodFillGetSize_Grad` beforehand.

The functions with the “\_4con” suffixes check 4-connected neighborhood of each pixel, i.e., side neighbors. The functions with the “\_8con” suffixes check 8-connected neighborhood of each pixel, i.e., side and corner neighbors. See [Figure Pixels Connectivity Patterns](#)”.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <i>imageStep</i> is less than <code>pRoiSize.width * &lt;pixelSize&gt;</code> .
<code>ippStsNotEvenStepErr</code>	Indicates an error condition if steps for floating-point images are not divisible by 4, or steps for 16-bit integer images are not divisible by 2.
<code>ippStsOutOfRangeErr</code>	Indicates an error condition if the <i>seed</i> point is out of ROI.

**Example**

The code example below shows how to use the function `ippiFloodFill_Grad4Con_8u_C1IR`.

```

Ipp8u
pImage[8*8] = { 0, 1, 2, 3, 4, 5, 6, 7,

                0, 1, 2, 3, 4, 5, 6, 7,

                0, 1, 2, 3, 4, 5, 6, 7,

                0, 1, 2, 3, 4, 5, 6, 7,

                0, 1, 2, 3, 4, 5, 6, 7,

                0, 1, 2, 3, 4, 5, 6, 7,

                0, 1, 2, 3, 4, 5, 6, 7};

IppiSize roiSize = { 4, 4 };
IppiPoint seed = { 2, 2 };
Ipp8u newVal = 9;
Ipp8u minDelta = 1;
Ipp8u maxDelta = 1;
int pBufferSize;
IppiConnectedComp pRegion;
ippiFloodFillGetSize_Grad ( roiSize, &pBufferSize );
Ipp8u* pBuffer = ippiMalloc_8u_C1 ( 8, 8, &pBufferSize );
ippiFloodFill_Grad4Con_8u_C1IR ( pImage, 8, roiSize, seed, newVal, minDelta,
                                maxDelta, &pRegion, pBuffer );

result:
0 1 2 3 4 5 6 7
0 1 2 3 4 5 6 7
0 1 2 3 4 5 6 7
0 1 2 3 4 5 6 7
0 1 2 3 4 5 6 7      pImage (source)
0 1 2 3 4 5 6 7
0 1 2 3 4 5 6 7
0 1 2 3 4 5 6 7

```

```

9 9 9 9 4 5 6 7
9 9 9 9 4 5 6 7
9 9 9 9 4 5 6 7
9 9 9 9 4 5 6 7
0 1 2 3 4 5 6 7      pImage (destination)
0 1 2 3 4 5 6 7
0 1 2 3 4 5 6 7
0 1 2 3 4 5 6 7

pRegion.area = 16
pRegion.value = 9.0
pRegion.rect.x = 0.0      pRegion (after FloodFill_Grad)
pRegion.rect.y = 0.0
pRegion.rect.width = 4.0
pRegion.rect.height = 4.0

```

## FloodFill\_Range

*Performs flood filling of pixels with values in the specified range in the connected area on an image.*

### Syntax

#### Case 1: Operations on one-channel data

```

IppStatus ippiFloodFill_Range4Con_<mod>(Ipp<DataType>* pImage, int imageStep, IppiSize
roiSize, IppiPoint seed, Ipp<datatype> newVal, Ipp<datatype> minDelta, Ipp<datatype>
maxDelta, IppiConnectedComp* pRegion, Ipp8u* pBuffer);

```

```

IppStatus ippiFloodFill_Range8Con_<mod>(Ipp<DataType>* pImage, int imageStep, IppiSize
roiSize, IppiPoint seed, Ipp<datatype> newVal, Ipp<datatype> minDelta, Ipp<datatype>
maxDelta, IppiConnectedComp* pRegion, Ipp8u* pBuffer);

```

Supported values for mod:

```

8u_C1IR      16u_C1IR      32f_C1IR

```

#### Case 2: Operations on three-channel data

```

IppStatus ippiFloodFill_Range4Con_<mod>(Ipp<DataType>* pImage, int imageStep, IppiSize
roiSize, IppiPoint seed, Ipp<datatype>* pNewVal, Ipp<datatype>* pMinDelta, Ipp<datatype>*
pMaxDelta, IppiConnectedComp* pRegion, Ipp8u* pBuffer);

```

```

IppStatus ippifloodFill_Range8Con_<mod>(Ipp<DataType>* pImage, int imageStep, IppiSize
roiSize, IppiPoint seed, Ipp<datatype>* pNewVal, Ipp<datatype>* pMinDelta, Ipp<datatype>*
pMaxDelta, IppiConnectedComp* pRegion, Ipp8u* pBuffer);

```

Supported values for mod:

8u\_C3IR                      32f\_C3IR

## Parameters

<i>pImage</i>	Pointer to the ROI in the source and destination image (in-place operation).
<i>imageStep</i>	Distance in bytes between starts of consecutive lines in the image buffer.
<i>roiSize</i>	Size of the image ROI in pixels.
<i>seed</i>	Initial point.
<i>minDelta</i>	Minimum difference between neighbor pixels for one-channel data.
<i>maxDelta</i>	Maximum difference between neighbor pixels for one-channel data.
<i>newVal</i>	Value to fill with for one-channel data.
<i>pMinDelta</i>	Pointer to the minimum differences between neighbor pixels for three-channel images.
<i>pMaxDelta</i>	Pointer to the maximum differences between neighbor pixels for three-channel images.
<i>pNewVal</i>	Pointer to the vector containing values to fill with for three-channel data.
<i>pRegion</i>	Pointer to the connected components structure that stores information about the refilled area.
<i>pBuffer</i>	Pointer to the temporary buffer.

## Description

The function `ippifloodFill_Grad` is declared in the `ippcv.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function performs [flood filling](#) (see [Flood Filling Functions](#)) of the group of connected pixels in the *seed* pixel neighborhoods whose pixel values  $v$  satisfy the following conditions:

$$v_0 - d_{lw} \leq v \leq v_0 + d_{up},$$

where  $v_0$  is the pixel value of the *seed* point, and  $d_{lw}$ ,  $d_{up}$  are *minDelta*, *maxDelta*, respectively. Values of these pixels are set to the *newVal* value.

The function requires a temporary buffer whose size should be computed with the function `ippifloodFillGetSize_Grad` beforehand.



The functions with the “\_4con” suffixes check 4-connected neighborhood of each pixel, i.e., side neighbors. The functions with the “\_8con” suffixes check 8-connected neighborhood of each pixel, i.e., side and corner neighbors. See [Figure Pixels Connectivity Patterns](#).”.

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <code>roiSize</code> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <code>imageStep</code> is less than <code>pRoiSize.width * &lt;pixelSize&gt;</code> .
<code>ippStsNotEvenStepErr</code>	Indicates an error condition if steps for floating-point images are not divisible by 4, or steps for 16-bit integer images are not divisible by 2.
<code>ippStsOutOfRangeErr</code>	Indicates an error condition if the <code>seed</code> point is out of ROI.

## Motion Analysis and Object Tracking

### Motion Template Functions

This section describes a motion templates function. This function generates motion templates images to rapidly determine where, how, and in which direction the motion occurred. The algorithms are based on [ [Davis97](#) ], and [ [Davis99](#) ]. The function operates on images that are the output of frame or background differencing, or other image segmentation operations. Thus, the input and output image types are all grayscale, that is, one color channel. The pixel types can be `8u`, `8s`, or `32f`.

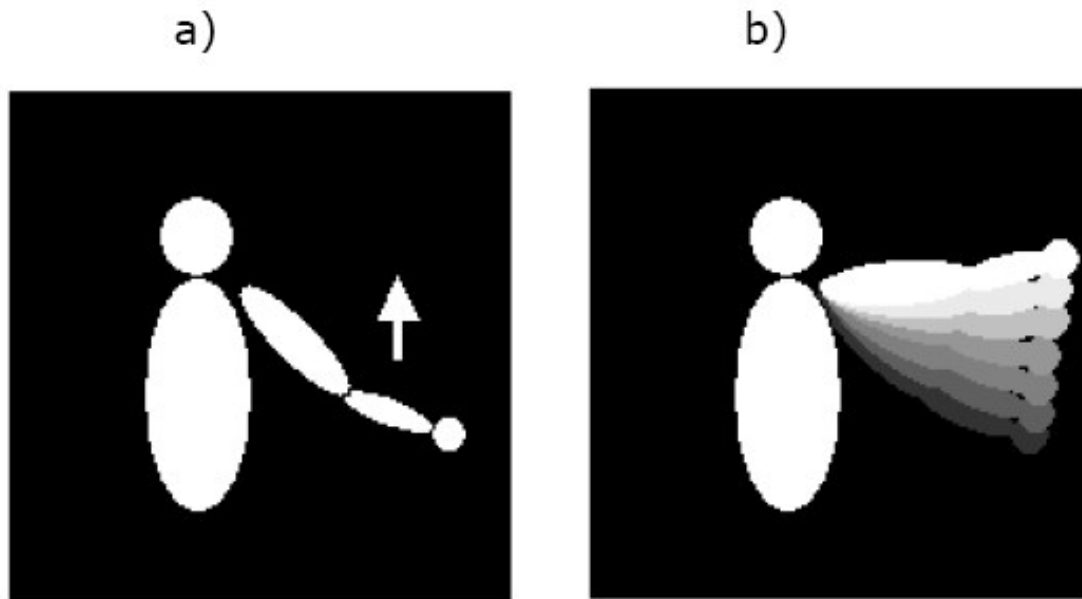
### Motion Representation

[Figure Motion Image History](#)” (a) shows capturing a foreground silhouette of the moving object or person. As the person or object moves, copying the most recent foreground silhouette as the highest values in the motion history image creates a “layered history” of the resulting motion. Typically, this “highest value” is just a floating-point timestamp of time since the code has been running in milliseconds. [Figure Motion Image History](#)”

(b) shows the result that may be called the *Motion History Image (MHI)*. The MHI in Figure Motion Image History” represents how the motion took place. A pixel level or a time delta threshold, as appropriate, is set such that pixel values in the MHI that fall below that threshold are set to zero.

### Motion Image History

---



The most recent motion has the highest value, earlier motions have decreasing values subject to a threshold below which the value is set to zero.

### Updating MHI Images

Generally, floating point images are used because system time differences, that is, time elapsing since the application was launched, are read in milliseconds to be further converted into a floating point number which is the value of the most recent silhouette. Then follows writing this current silhouette over the past silhouettes with subsequent thresholding away pixels that are too old to create the MHI.

## UpdateMotionHistory

*Updates motion history image using motion silhouette at given timestamp.*

---

### Syntax

```
IppStatus ippiUpdateMotionHistory_<mod>(const Ipp<srcDatatype>* pSilhouette, int silhStep,
Ipp32f* pMhi, int mhiStep, IppiSize roiSize, Ipp32f timeStamp, Ipp32f mhiDuration);
```

Supported values for mod:

8u32f\_C1IR      16u32f\_C1IR      32f\_C1IR

## Parameters

<i>pSilhouette</i>	Pointer to the silhouette image ROI that has non-zero values for those pixels where the motion occurs.
<i>roiSize</i>	Size of the image ROI in pixels.
<i>silhStep</i>	Distance in bytes between starts of consecutive lines in the silhouette image.
<i>pMhi</i>	Pointer to the motion history image which is both an input and output parameter.
<i>mhiStep</i>	Distance in bytes between starts of consecutive lines in the motion history image.
<i>timeStamp</i>	Timestamp in milliseconds.
<i>mhiDuration</i>	Threshold for MHI pixels. MHI motions older than this threshold are deleted.

## Description

The function `ippiUpdateMotionHistory` is declared in the `ippcv.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function updates the motion history image. It sets MHI pixels to the current *timestamp* value, if their values are non-zero.

The function deletes MHI pixels, if their values are less than the *mhiDuration* timestamp, that is, the pixels are “old.”

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <i>mhiStep</i> or <i>silhStep</i> is less than <i>roiSize.width</i> * <i>&lt;pixelSize&gt;</i> .
<code>ippStsNotEvenStepErr</code>	Indicates an error condition if the step value is not divisible by 2 for 16u images, and by 4 for 32f images.
<code>ippStsOutOfRangeErr</code>	Indicates an error when <i>mhiDuration</i> is negative.

## Optical Flow

This section describes the functions that calculate the optical flow using the pyramidal Lucas-Kanade algorithm [Bou99].

The optical flow between two images is generally defined as an apparent motion of image brightness. Let  $I(x, y, t)$  be the image brightness that changes in time to provide an image sequence.

Optical flow coordinates

$$u = \frac{\partial x}{\partial t}, v = \frac{\partial y}{\partial t}$$

can be found from so called *optical flow constraint equation*:

$$-\frac{\partial I}{\partial t} = \frac{\partial I}{\partial x}u + \frac{\partial I}{\partial y}v$$

The Lucas-Kanade algorithm assumes that the group of adjacent pixels has the same velocity and finds the approximate solution of the above equation using the least square method.

## OpticalFlowPyrLKInitAlloc

*Allocates memory and initializes a structure for optical flow calculation.*

---

### Syntax

```
IppStatus ippiOpticalFlowPyrLKInitAlloc_<mod>(IppiOptFlowPyrLK_<mod>** ppState, IppiSize roiSize, int winSize, IppHintAlgorithm hint);
```

Supported values for mod:

8u\_C1R                  16u\_C1R                  32f\_C1R

### Parameters

<i>ppState</i>	Pointer to the pointer to the optical flow structure being initialized.
<i>roiSize</i>	Size of the source image (zero level of the pyramid) ROI in pixels.
<i>winSize</i>	Size of the search window of each pyramid level.
<i>hint</i>	Option to select the algorithmic implementation of the transform function.

### Description

The function `ippiOpticalFlowPyrLKInitAlloc` is declared in the `ippcv.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)). This function allocates memory and initializes a structure *pState* for calculation the optical flow between two images using the pyramidal Lucas-Kanade algorithm in the centered window of size *winSize* \* *winSize*. Computation algorithm is specified by the *hint* argument. This structure is used by the function `ippiOpticalFlowPyrLK`.

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error if <i>ppState</i> is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value or if <i>winSize</i> is equal to or less than 0.

`ippStsMemAllocErr` Memory allocation error.

## OpticalFlowPyrLKFree

*Frees memory allocated for the optical flow structure.*

### Syntax

```
IppStatus ippOpticalFlowPyrLKFree_<mod> (IppiOptFlowPyrLK_<mod>* pState);
```

Supported values for mod:

8u\_C1R      16u\_C1R      32f\_C1R

### Parameters

*pState* Pointer to the optical flow structure.

### Description

The function `ippOpticalFlowPyrLKFree` is declared in the `ippcv.h` file. This function frees memory allocated by the function `ippOpticalFlowPyrLKInitAlloc` for the optical flow structure.

### Return Values

`ippStsNoErr` Indicates no error. Any other value indicates an error or a warning.  
`ippStsNullPtrErr` Indicates an error if *pState* is NULL.

## OpticalFlowPyrLK

*Calculates optical flow for the set of feature points using the pyramidal Lucas-Kanade algorithm.*

### Syntax

```
IppStatus ippOpticalFlowPyrLK_<mod>(IppiPyramid* pPyr1, IppiPyramid* pPyr2, const  
IppiPoint2D32f* pPrev, IppiPoint_32f* pNext, Ipp8s* pStatus, Ipp32f* pError, int numFeat,  
int winSize, int maxLev, int maxIter, Ipp32f threshold, IppiOptFlowPyrLK_<mod>* pState);
```

Supported values for mod:

8u\_C1R      16u\_C1R      32f\_C1R

### Parameters

*pPyr1* Pointer to the ROI in the first image pyramid structure.  
*pPyr2* Pointer to the ROI in the second image pyramid structure.  
*pPrev* Pointer to the array of initial coordinates of the feature points.  
*pNext* Pointer to the array of new coordinates of feature point; as input it contains hints for new coordinates.

<i>pStatus</i>	Pointer to the array of result indicators.
<i>pError</i>	Pointer to the array of differences between neighborhoods of old and new point positions.
<i>numFeat</i>	Number of feature points.
<i>winSize</i>	Size of the search window of each pyramid level.
<i>maxLev</i>	Pyramid level to start the operation.
<i>maxIter</i>	Maximum number of algorithm iterations for each pyramid level.
<i>threshold</i>	Threshold value.
<i>pState</i>	Pointer to the pyramidal optical flow structure.

## Description

The function `ippiOpticalFlowPyrLK` is declared in the `ippcv.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function implements the iterative version of the Lucas-Kanade algorithms [[Bou99](#)]. It computes with sub-pixel accuracy new coordinates of the *numFeat* feature points of two images at time *t* and *t+dt*. Their initial coordinates are places in the *pPrev* array. Computed values of new coordinates of the feature points are stored in the array *pNext*, that initially contains estimations of them (or hints), for example, based on the flow values for the previous image in sequence. If there are not such hints, the *pNext* array contains the same initial coordinates as the *pPrev* array.

*pStatus* and *pError* are arrays of size *numFeat* with the respective data type.

The images are presented by the pyramid structures *pPyr1* and *pPyr2* respectively (see description of the [ippiPyramidInitAlloc](#) function for more details). These structures should be initialized by calling the function `ippiPyramidInitAlloc` beforehand. Furthermore the function uses the pyramidal optical flow structure *pState* that also should be previously initialized by the function `ippiOpticalFlowPyrLKInitAlloc`.

The function starts operation on the highest pyramid level (smallest image) that is specified by the *maxLev* parameter in the centered search window whose size *winSize* could not exceed the corresponding value *winSize* that is specified in the function `ippiOpticalFlowPyrLKInitAlloc`. The operation for *i*-th feature point on the given pyramid level finishes if:

- new position of the point is found

$$(\sqrt{dx^2 + dy^2} < threshold),$$

- specified number of iteration *maxIter* is performed,
- the intersection between the pyramid layer and the search window became empty.

In first two cases for non-zero levels the new position coordinates are scaled to the next pyramid level and the operation continues on the next level. For zero level or for third case the operation stops, the number of the corresponding level is written to the *pStatus[i]* element, the new coordinates are scaled to zero level and are written to *pNext[i]*. The square root of the average squared difference between neighborhoods of old and new positions is written to *pError[i]*.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error if one of the specified pointer is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <code>numFeat</code> or <code>winSize</code> has zero or negative value.
<code>ippStsBadArgErr</code>	Indicates an error condition if <code>maxLev</code> or <code>threshold</code> has negative value, or <code>maxIter</code> has zero or negative value.

## Example

The code example below shows how to build pyramids and calculate the optical flow for two images.

```
void pyroptflow(
    const Ipp8u  *prevFrame, // previous frame
    int          prevStep,   // its row step
    const Ipp8u  *nextFrame, // next frame
    int          nextStep,   // its row step
    IppiSize     roiSize,    // frame size
    int          numLevel,   // pyramid level number (5)
    float        rate,       // pyramid rate (2.0f)
    Ipp16s       *pKernel,   // pyramid kernel
    int          kerSize,    // pyramid kernel size (5)
    IppiPoint_32f *prevPt,   // coordinates on previous frame
    IppiPoint_32f *nextPt,   // hint to coordinates on next frame
    Ipp8s        *pStatus,   // result indicators
    Ipp32f       *pError,    // differences
    int          numFeat,    // point number
    int          winSize,    // search window size (41)
    int          numIter,    // iteration number (5)
    float        threshold) // threshold (0.0001f)
{
    IppiPyramid *pPyr1, *pPyr2;
    IppiOptFlowPyrLK *pOF;
    ippPyramidInitAlloc (&pPyr1, numLevel, roiSize, rate);
    ippPyramidInitAlloc (&pPyr2, numLevel, roiSize, rate);
    {
        IppiPyramidDownState_8u_C1R **pStatel =
```

```

(IppiPyramidDownState_8u_C1R**)&(pPyr1->pState);
IppiPyramidDownState_8u_C1R **pState2 =

(IppiPyramidDownState_8u_C1R**)&(pPyr2->pState);
Ipp8u **pImg1 = pPyr1->pImage;
Ipp8u **pImg2 = pPyr2->pImage;
int *pStep1 = pPyr1->pStep;
int *pStep2 = pPyr2->pStep;
IppiSize *pRoi1 = pPyr1->pRoi;
IppiSize *pRoi2 = pPyr2->pRoi;
IppHintAlgorithm hint=ippAlgHintFast;
int i,level = pPyr1->level;

ippiPyramidLayerDownInitAlloc_8u_C1R(pState1,roiSize,rate,pKernel,kerSize,
    IPPI_INTER_LINEAR);
ippiPyramidLayerDownInitAlloc_8u_C1R(pState2,roiSize,rate,pKernel,kerSize,
    IPPI_INTER_LINEAR);

pImg1[0] = (Ipp8u*)prevFrame;
pImg2[0] = (Ipp8u*)nextFrame;
pStep1[0] = prevStep;
pStep2[0] = nextStep;
pRoi1[0] = pRoi2[0] = roiSize;
for (i=1; i<=level; i++) {
    pPyr1->pImage[i] = ippiMalloc_8u_C1(pRoi1[i].width,pRoi1[i].height,
        pStep1+i);
    pPyr2->pImage[i] = ippiMalloc_8u_C1(pRoi2[i].width,pRoi2[i].height,
        pStep2+i);
    ippiPyramidLayerDown_8u_C1R(pImg1[i-1],pStep1[i-1],pRoi1[i-1],

        pImg1[i],pStep1[i],pRoi1[i],*pState1);
    ippiPyramidLayerDown_8u_C1R(pImg2[i-1],pStep2[i-1],pRoi2[i-1],

```



```

        pImg2[i],pStep2[i],pRoi2[i],*pState2);
    }

   ippiOpticalFlowPyrLKInitAlloc_8u_C1R (&pOF,roiSize,winSize,hint);
   ippiOpticalFlowPyrLK_8u_C1R (pPyr1,pPyr2,prevPt,nextPt,pStatus,pError,
        numFeat,winSize,numLevel,numIter,threshold,pOF);

   ippiOpticalFlowPyrLKFree_8u_C1R(pOF);
    for (i=level; i>0; i--) {
        if (pImg2[i]) ippiFree(pImg2[i]);
        if (pImg1[i]) ippiFree(pImg1[i]);
    }
   ippiPyramidLayerDownFree_8u_C1R(*pState1);
   ippiPyramidLayerDownFree_8u_C1R(*pState2);
}
ippiPyramidFree (pPyr2);
ippiPyramidFree (pPyr1);
}

```

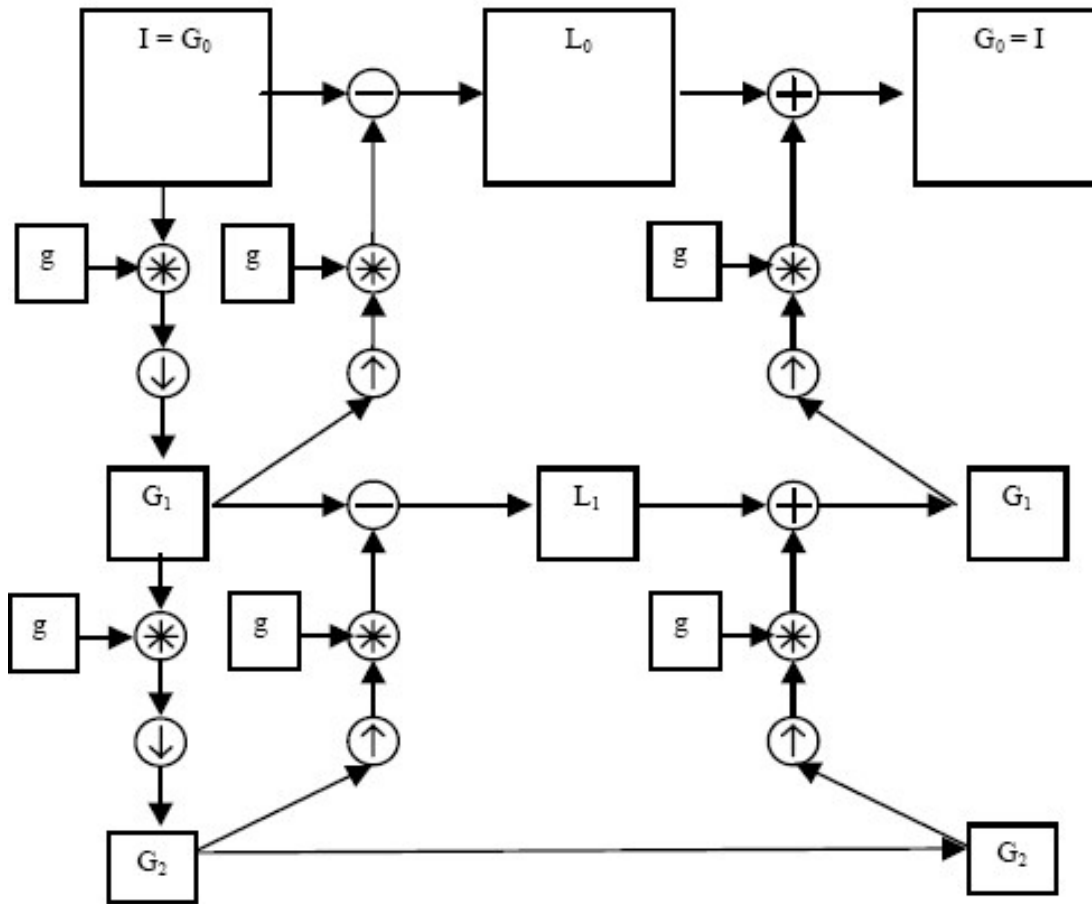
## Pyramids Functions

The functions described in this section generate and reconstruct Gaussian and Laplacian image pyramids.

[Figure “Three-level Gaussian and Laplacian Pyramid”](#) shows the basics of creating Gaussian or Laplacian pyramids. The original image  $G_0$  is convolved with a Gaussian, then down-sampled to get the reduced image  $G_1$ . This process can be continued as far as desired or until the image size is one pixel.

The Laplacian pyramid can be built from a Gaussian pyramid as follows: the Laplacian level  $k$ , that is,  $L_k$  can be built by up-sampling the lower level image  $G_{k+1}$ . Convolving the image with a Gaussian kernel  $g$  interpolates the pixels “missing” after up-sampling. The resulting image is subtracted from the Gaussian smoothed image  $G_k$ . To rebuild the original image, the process is reversed as Figure “Three-level Gaussian and Laplacian Pyramid” shows.

### Three-level Gaussian and Laplacian Pyramid



The Gaussian image pyramid on the left is used to create the Laplacian pyramid in the center, which is used to reconstruct the Gaussian pyramid and original image on the right. In [Figure “Three-level Gaussian and Laplacian Pyramid”](#),  $I$  is the original image,  $G_k$  is the Gaussian image,  $L_k$  is the Laplacian image, subscripts  $k$  denote level of the pyramid,  $g$  is a Gaussian kernel used to convolve the image before down-sampling or after up-sampling.

The Intel IPP implement only Gaussian pyramids. To build Laplacian pyramids, the following general rule for Intel IPP pyramids functions is to be followed:

$$L_i = G_i - \text{PyrUp}(\text{PyrDown}(G_i)),$$

where  $L_i$  is an image on the  $i$ -th level in the Laplacian pyramid,  $G_i$  is an image on the  $i$ -th level in the Gaussian pyramid.

## PyrDownGetBufSize

*Computes the size of temporary buffer for the function `ippiPyrDown`.*

---

### Syntax

```
IppStatus ippiPyrDownGetBufSize_Gauss5x5(int roiWidth, IppDataType dataType, int channels,
int* pBufferSize);
```

### Parameters

<code>roiWidth</code>	Width of the source image ROI in pixels.
<code>dataType</code>	Data type of the source image.
<code>channels</code>	Number of channels.
<code>pBufferSize</code>	Pointer to the computed size of the temporary buffer.

### Description

The function `ippiPyrDownGetBufSize` is declared in the `ippcv.h` file. This function calculates the size of the temporary buffer to be used by the function `ippiPyrDown`. The buffer of the calculated size can be used to process smaller images.

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtr</code>	Indicates an error condition if the <code>pBufferSize</code> pointer is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if the value of <code>roiWidth</code> is zero or negative.
<code>ippStsNumChannelsErr</code>	Indicates an error condition if <code>channels</code> is not 1 or 3.
<code>ippStsDataTypeErr</code>	Indicates an error condition if <code>dataType</code> is not <code>Ipp8u</code> , <code>Ipp8s</code> , or <code>Ipp32f</code> .

## PyrUpGetBufSize

*Calculates size of temporary buffer for the function `ippiPyrUp`.*

---

### Syntax

```
IppStatus ippiPyrUpGetBufSize_Gauss5x5(int roiWidth, IppDataType dataType, int channels,
int* pBufferSize);
```

### Parameters

<code>roiWidth</code>	Width of the source image ROI in pixels.
<code>dataType</code>	Data type of the source image.
<code>channels</code>	Number of channels.
<code>pBufferSize</code>	Pointer to the computed size of the temporary buffer.

## Description

The function `ippiPyrUpGetBufSize` is declared in the `ippcv.h` file. This function calculates the size of the temporary buffer to be used by the function `ippiPyrUp`. The buffer of the calculated size can be used to process smaller images.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtr</code>	Indicates an error condition if the <code>pBufferSize</code> pointer is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if the value of <code>roiWidth</code> is zero or negative.
<code>ippStsNumChannelsErr</code>	Indicates an error condition if <code>channels</code> is not 1 or 3.
<code>ippStsDataTypeErr</code>	Indicates an error condition if <code>dataType</code> is not <code>Ipp8u</code> , <code>Ipp8s</code> , or <code>Ipp32f</code> .

## PyrDown

*Applies the Gaussian to image and then performs down-sampling.*

---

## Syntax

```
ippStatus ippiPyrDown_Gauss5x5_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, Ipp8u* pBuffer);
```

Supported values for `mod`:

<code>8u_C1R</code>	<code>8s_C1R</code>	<code>32f_C1R</code>
<code>8u_C3R</code>	<code>8s_C3R</code>	<code>32f_C3R</code>

## Parameters

<code>pSrc</code>	Pointer to the source image ROI.
<code>srcStep</code>	Distance in bytes between starts of consecutive lines in the source image.
<code>pDst</code>	Pointer to the destination image ROI.
<code>dstStep</code>	Distance in bytes between starts of consecutive lines in the destination image.
<code>roiSize</code>	Size of the source image ROI in pixels.
<code>pBuffer</code>	Pointer to the temporary buffer.

## Description

The function `ippiPyrDown` is declared in the `ippcv.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function applies the 5x5 Gaussian to the source image *pSrc* and then down-samples it, that is, removes odd rows and columns from the image. The size of the destination image is  $(roiSize.height+1)/2 * (roiSize.width+1)/2$ . The following Gaussian mask is used:

$$\frac{1}{16} \begin{bmatrix} 1 \\ 4 \\ 6 \\ 4 \\ 1 \end{bmatrix} \times \frac{1}{16} \begin{bmatrix} 1 & 4 & 6 & 4 & 1 \end{bmatrix} = \frac{1}{256} \begin{bmatrix} 1 & 4 & 6 & 4 & 1 \\ 4 & 16 & 24 & 16 & 4 \\ 6 & 24 & 36 & 24 & 6 \\ 4 & 16 & 24 & 16 & 4 \\ 1 & 4 & 6 & 4 & 1 \end{bmatrix}$$

The function uses the temporary buffer *pBuffer* - its size should be computed using the function [ippiPyrDownGetBufSize](#) beforehand.

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtr</code>	Indicates an error condition if one of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> is less than <i>roiSize.width</i> * <i>&lt;pixelSize&gt;</i> , or <i>dstStep</i> is less than $(roiSize.width * <pixelSize>)/2$ .
<code>ippStsNotEvenStepErr</code>	Indicates an error condition if steps for floating-point images are not divisible by 4.

## PyrUp

Up-samples image and then applies the Gaussian.

### Syntax

```
IppStatus ippiPyrUp_Gauss5x5_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, Ipp8u* pBuffer);
```

Supported values for *mod*:

8u_C1R	8s_C1R	32f_C1R
8u_C3R	8s_C3R	32f_C3R

### Parameters

*pSrc* Pointer to the source image ROI.

<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>roiSize</i>	Size of the source image ROI in pixels.
<i>pBuffer</i>	Pointer to the temporary buffer.

### Description

The function `ippiPyrUp` is declared in the `ippcv.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)). This function up-samples the source image *pSrc*, that is, inserts odd zero rows and columns, and then applies the 5x5 Gaussian multiplied by 4 to it. The following mask is used:

$$4 \times \frac{1}{16} \begin{bmatrix} 1 \\ 4 \\ 6 \\ 4 \\ 1 \end{bmatrix} \times \frac{1}{16} \begin{bmatrix} 1 & 4 & 6 & 4 & 1 \end{bmatrix} = \frac{1}{64} \begin{bmatrix} 1 & 4 & 6 & 4 & 1 \\ 4 & 16 & 24 & 16 & 4 \\ 6 & 24 & 36 & 24 & 6 \\ 4 & 16 & 24 & 16 & 4 \\ 1 & 4 & 6 & 4 & 1 \end{bmatrix}$$

The function uses the temporary buffer *pBuffer* - its size should be computed using the function `ippiPyrUpGetBufSize` beforehand.

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtr</code>	Indicates an error condition if one of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> is less than <i>roiSize.width</i> * <i>&lt;pixelSize&gt;</i> , or <i>dstStep</i> is less than 2 * <i>roiSize.width</i> * <i>&lt;pixelSize&gt;</i> .
<code>ippStsNotEvenStepErr</code>	Indicates an error condition if when steps for floating-point images are not divisible by 4.

## Universal Pyramids

The functions described in this section operate with universal image pyramids. These pyramids use separable symmetric kernel (not only Gaussian type) and downsampling/upsampling with arbitrary factor (not only 2). The next pyramid layer can be built for an image of an arbitrary size. These pyramids are used in some computer vision algorithms, for example, in optical flow calculations.



**NOTE.** All universal pyramid functions use the mirrored border.

[Example](#) shows how to build pyramids and calculate the optical flow for two images.

## PyramidInitAlloc

*Allocates memory and initializes a pyramid structure.*

### Syntax

```
IppStatus ippPyramidInitAlloc(IppiPyramid** ppPyr, int level, IppiSize roiSize, Ipp32f rate);
```

### Parameters

<i>ppPyr</i>	Pointer to the pointer to the pyramid structure.
<i>level</i>	Maximum number of pyramid levels.
<i>roiSize</i>	Size of zero level image ROI in pixels.
<i>rate</i>	Ratio between neighbouring levels ( $1 < rate \leq 10$ ).

### Description

The function `ippPyramidInitAlloc` is declared in the `ippcv.h` file. This function allocates memory and initializes the structure for the pyramid with `level+1` levels. This structure is used by the [ippiOpticalFlowPyrLK](#) function for optical flow calculations.

The *IppiPyramid* structure contains the following fields:

<i>pImage</i>	Pointer to the array of $(level+1)$ layer images.
<i>pStep</i>	Pointer to the array of $(level+1)$ image row step values.
<i>pRoi</i>	Pointer to the array of $(level+1)$ layer image ROIs.
<i>pRate</i>	Pointer to the array of $(level+1)$ ratios of $i$ -th levels to the zero level ( $rate^{-i}$ ).
<i>pState</i>	Pointer to the structure to compute the next pyramid layer.
<i>level</i>	Number of levels in the pyramid.

The `ippPyramidInitAlloc` function fills *pRoi* and *pRate* arrays and the *level* field. The value of *level* is equal to the minimum of the input value of the *level* parameter and the maximum possible number of layers of the pyramid with given *rate* and zero level size.

Other fields should be specified by the user. The pyramid layer structure *pState* should be initialized by the functions [ippiPyramidLayerDownInitAlloc](#) or [ippiPyramidLayerUpInitAlloc](#). Pyramid layer images can be obtained in many different ways, for example, using the Intel IPP functions [ippiPyramidLayerDown](#) and [ippiPyramidLayerUp](#).

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error if <i>ppPyr</i> pointer is NULL.

<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.
<code>ippStsBadArgErr</code>	Indicates an error condition if <i>level</i> is equal to or less than 0, or if <i>rate</i> is out of the range.
<code>ippStsMemAllocErr</code>	Memory allocation error.

## PyramidFree

*Frees memory allocated for the pyramid structure.*

---

### Syntax

```
IppStatus ippPyramidFree(IppiPyramid* pPyr);
```

### Parameters

*pPyr*                                      Pointer to the pyramid structure.

### Description

The function `ippPyramidFree` is declared in the `ippcv.h` file. This function frees memory allocated by the function `ippPyramidInitAlloc` for the pyramid structure *pPyr*.

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error if <i>pPyr</i> pointer is NULL.

## PyramidLayerDownInitAlloc

*Allocates memory and initializes a structure for creating a lower pyramid layer.*

---

### Syntax

#### Case 1: Operation on integer data

```
IppStatus ippPyramidLayerDownInitAlloc_<mod>(IppiPyramidDownState_<mod>** ppState, IppiSize srcRoiSize, Ipp32f rate, Ipp16s* pKernel, int kerSize, int mode);
```

Supported values for *mod*:

<code>8u_C1R</code>	<code>16u_C1R</code>
<code>8u_C3R</code>	<code>16u_C3R</code>



## Case 2: Operation on floating point data

```
IppStatus ippPyramidLayerDownInitAlloc_<mod>(IppiPyramidDownState_<mod>** ppState, IppiSize
srcRoiSize, Ipp32f rate, Ipp32f* pKernel, int kerSize, int mode);
```

Supported values for `mod`:

32f\_C1R

32f\_C3R

## Parameters

<i>ppState</i>	Pointer to the pointer to the pyramid layer structure.
<i>srcRoiSize</i>	Maximal size of source image ROI in pixels.
<i>rate</i>	Ratio between neighbouring levels ( $1 < rate \leq 10$ ).
<i>pKernel</i>	Pointer to the symmetric separable convolution kernel.
<i>kerSize</i>	Size of the convolution kernel, should be odd and greater than 2.
<i>mode</i>	Specifies the type of interpolation for image downsampling; should be set to <code>IPPI_INTER_LINEAR</code> to perform bilinear interpolation.

## Description

The function `ippPyramidLayerDownInitAlloc` is declared in the `ippcv.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#))

This function allocates memory and initializes the structure *pState* to build a lower pyramid layer. This structure is used by the function [ippPyramidLayerDown](#) and can be applied to process images with size not greater than *srcRoiSize*. Generally the specified kernel *pKernel* should be symmetric. However if it is not symmetric, the function builds the symmetric kernel using its first half, and returns the warning. For integer rates downsampling is performed by discarding rows and columns that are not multiples of the rate value. For non-integer rate bilinear interpolation is used (see [appendix B "Linear Interpolation"](#) for more information). The symmetric separable kernel of odd size can be not Gaussian. If the sum of kernel elements is not equal to zero, then the kernel is normalized.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error if one of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <i>srcRoiSize</i> has a field with zero or negative value.
<code>ippStsBadArgErr</code>	Indicates an error condition if one of the parameters <i>rate</i> , <i>kerSize</i> , <i>mode</i> has wrong value.
<code>ippStsMemAllocErr</code>	Memory allocation error.
<code>ippStsSymKernelExpected</code>	Indicates a warning if the specified kernel is not symmetric.

## PyramidLayerDownFree

*Frees memory allocated for the lower pyramid layer structure.*

---

### Syntax

```
IppStatus ippiPyramidLayerDownFree_<mod>(IppiPyramidDownState_<mod>* pState);
```

Supported values for mod:

8u_C1R	16u_C1R	32f_C1R
8u_C3R	16u_C3R	32f_C3R

### Parameters

*pState*                                      Pointer to the pyramid layer structure .

### Description

The function `ippiPyramidLayerDownFree` is declared in the `ippcv.h` file. This function frees memory allocated by the function [ippiPyramidLayerDownInitAlloc](#) for the lower pyramid layer structure *pState*.

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error if <i>pState</i> is NULL.

## PyramidLayerUpInitAlloc

*Allocates memory and initializes a structure for creating an upper pyramid layer.*

---

### Syntax

#### Case 1: Operation on integer data

```
IppStatus ippiPyramidLayerUpInitAlloc_<mod>(IppiPyramidUpState_<mod>** ppState, IppiSize dstRoiSize, Ipp32f rate, Ipp16s* pKernel, int kerSize, int mode);
```

Supported values for mod:

8u_C1R	16u_C1R
8u_C3R	16u_C3R

## Case 2: Operation on floating point data

```
IppStatus ippiPyramidLayerUpInitAlloc_<mod>(IppiPyramidUpState_<mod>** ppState, IppiSize
dstRoiSize, Ipp32f rate, Ipp32f* pKernel, int kerSize, int mode);
```

Supported values for `mod`:

`32f_C1R`

`32f_C3R`

## Parameters

<i>pState</i>	Pointer to the pointer to the pyramid layer structure.
<i>dstRoiSize</i>	Maximal size of the destination image ROI in pixels.
<i>rate</i>	Ratio between neighbouring levels ( $1 < rate \leq 10$ ).
<i>pKernel</i>	Pointer to the symmetric separable convolution kernel.
<i>kerSize</i>	Size of the convolution kernel, should be odd and greater than 2.
<i>mode</i>	Specifies the type of interpolation for image upsampling; should be set to <code>IPPI_INTER_LINEAR</code> to perform bilinear interpolation.

## Description

The function `ippiPyramidLayerUpInitAlloc` is declared in the `ippcv.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function allocates memory and initializes the structure *pState* to build an upper pyramid layer. This structure is used by the function `ippiPyramidLayerUp` and can be applied to process images with size not greater than *dstRoiSize*. Generally the specified kernel *pKernel* should be symmetric. However if it is not symmetric, the function builds the symmetric kernel using its first half, and returns the warning. For integer rates upsampling is performed by inserting zero rows and columns that are not multiples of the rate value. For non-integer rate bilinear interpolation is used to calculate kernel coefficients for pixels with non-integer coordinates. The symmetric separable kernel of odd size can be not Gaussian. If the sum of kernel elements is not equal to zero, then the kernel is normalized.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error if one of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <i>dstRoiSize</i> has a field with zero or negative value.
<code>ippStsBadArgErr</code>	Indicates an error condition if one of the parameters <i>rate</i> , <i>kerSize</i> , <i>mode</i> has wrong value, or the sum of the kernel elements is not positive.
<code>ippStsMemAllocErr</code>	Memory allocation error.
<code>ippStsSymKernelExpected</code>	Indicates a warning if the specified kernel is not symmetric.

## PyramidLayerUpFree

*Frees memory allocated for the upper pyramid layer structure.*

---

### Syntax

```
IppStatus ippiPyramidLayerUpFree_<mod>(IppiPyramidUpState_<mod>* pState);
```

Supported values for mod:

8u_C1R	16u_C1R	32f_C1R
8u_C3R	16u_C3R	32f_C3R

### Parameters

*pState*                                      Pointer to the pyramid layer structure.

### Description

The function `ippiPyramidLayerUpFree` is declared in the `ippcv.h` file. This function frees memory allocated by the function `ippiPyramidLayerUpInitAlloc` for the upper pyramid layer structure *pState*.

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error if <i>pState</i> is NULL.

## GetPyramidDownROI

*Computes the size of the lower pyramid layer.*

---

### Syntax

```
IppStatus ippiGetPyramidDownROI(IppiSize srcRoiSize, IppiSize* pDstRoiSize, Ipp32f rate);
```

### Parameters

<i>srcRoiSize</i>	Size of the source pyramid layer ROI in pixels.
<i>pDstRoiSize</i>	Pointer to the size of the destination (lower) pyramid layer ROI in pixels.
<i>rate</i>	Ratio between source and destination layers ( $1 < rate \leq 10$ ).

### Description

The function `ippiGetPyramidDownROI` is declared in the `ippcv.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function computes the size of the lower pyramid layer *pDstRoiSize* for a source layer of a given size *srcRoiSize* and specified size ratio *rate* between them in accordance with the following formulas:

```
pDstRoiSize.width = max(1, min(⌈srcRoiSize.width / rate⌉, srcRoiSize.width - 1))
pDstRoiSize.height = max(1, min(⌈srcRoiSize.height / rate⌉, srcRoiSize.height - 1))
```



**NOTE.** Since for the non-integer *rate* results depend on the computational precision, it is strongly recommended to use this function in computations.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error if <i>pDstRoiSize</i> pointer is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>srcRoiSize</i> has a field with zero or negative value.
<code>ippStsBadArgErr</code>	Indicates an error condition if <i>rate</i> is out of the range.

## GetPyramidUpROI

*Computes the size of the upper pyramid layer.*

### Syntax

```
IppStatus ippGetPyramidUpROI(IppiSize srcRoiSize, IppiSize* pDstRoiSizeMin, IppiSize*
pDstRoiSizeMax, Ipp32f rate);
```

### Parameters

<i>srcRoiSize</i>	Size of the source pyramid layer ROI in pixels.
<i>pDstRoiSizeMin</i>	Pointer to the minimal size of the destination (upper) pyramid layer ROI in pixels.
<i>pDstRoiSizeMax</i>	Pointer to the maximal size of the destination (upper) pyramid layer ROI in pixels.
<i>rate</i>	Ratio between source and destination layers ( $1 < rate \leq 10$ ).

### Description

The function `ippGetPyramidUpROI` is declared in the `ippcv.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function computes possible sizes of the upper pyramid layer *pDstRoiSizeMin* and *pDstRoiSizeMax* for a source layer of a given size *srcRoiSize* and specified size ratio *rate* between them in accordance with the following formulas:

maximum size *pDstRoiSizeMax*:

```
pDstRoiMax.width = max(srcRoiSize.width + 1, ⌊srcRoiSize.width · rate⌋)
pDstRoiMax.height = max(srcRoiSize.height + 1, ⌊srcRoiSize.height · rate⌋)
```

minimum size *pDstRoiSizeMin*:

if the width and height of the source layer ROI is greater than 1,

```
pDstRoiMin.width = max(srcRoiSize.width+1, [(srcRoiSize.width-1)·rate])
pDstRoiMin.height = max(srcRoiSize.height+1, [(srcRoiSize.height-1)·rate])
```

if the width and height of the source layer ROI is equal to 1,

```
pDstRoiMin.width = 1
pDstRoiMin.height = 1
```




---

**NOTE.** Since for the non-integer *rate* results depend on the computational precision, it is strongly recommended to use this function in computations.

---

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error if one of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <i>srcRoiSize</i> has a field with zero or negative value.
<code>ippStsBadArgErr</code>	Indicates an error condition if <i>rate</i> is out of the range.

## PyramidLayerDown

*Creates a lower pyramid layer.*

---

### Syntax

```
IppStatus ippiPyramidLayerDown_<mod>(const Ipp<datatype>* pSrc, int srcStep, IppiSize
srcRoiSize, Ipp<datatype>* pDst, int dstStep, IppiSize dstRoiSize,
IppiPyramidDownState_<mod>* pState);
```

Supported values for *mod*:

8u_C1R	16u_C1R	32f_C1R
8u_C3R	16u_C3R	32f_C3R

### Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.

<i>srcRoiSize</i>	Size of the source image ROI in pixels.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>dstRoiSize</i>	Size of the destination image ROI in pixels.
<i>pState</i>	Pointer to the pyramid layer structure.

## Description

The function `ippiPyramidLayerDown` is declared in the `ippcv.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function creates a lower pyramid layer *pDst* from the source image *pSrc*, that is, it applies the convolution kernel to the source image using the mirror border and then performs downsampling. Before calling `ippiPyramidLayerDown`, the pyramid layer structure *pState* should be initialized by calling the function `ippiPyramidLayerDownInitAlloc` and the kernel and the downsampling ratio should be specified beforehand. The function can process images with *srcRoiSize* not greater than *srcRoiSize* parameter specified in the function `ippiPyramidLayerDownInitAlloc`.



**NOTE.** This function uses the mirrored border.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error if one of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <i>srcRoiSize</i> or <i>dstRoiSize</i> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> is less than <i>srcRoiSize.width</i> * <i>&lt;pixelSize&gt;</i> , or <i>dstStep</i> is less than <i>dstRoiSize.width</i> * <i>&lt;pixelSize&gt;</i> .
<code>ippStsNotEvenStepErr</code>	Indicates an error condition if one of the step values is not divisible by 4 for floating-point images, or by 2 for short-integer images.
<code>ippStsBadArgErr</code>	Indicates an error condition if <i>pState-&gt;rate</i> has wrong value.

## PyramidLayerUp

*Creates an upper pyramid layer.*

---

### Syntax

```
IppStatus ippPyramidLayerUp_<mod>(const Ipp<datatype>* pSrc, int srcStep, IppiSize srcRoiSize, Ipp<datatype>* pDst, int dstStep, IppiSize dstRoiSize, IppiPyramidUpState_<mod>* pState);
```

Supported values for mod:

8u_C1R	16u_C1R	32f_C1R
8u_C3R	16u_C3R	32f_C3R

### Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>srcRoiSize</i>	Size of source image ROI in pixels.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>dstRoiSize</i>	Size of destination image ROI in pixels.
<i>pState</i>	The pointer to the pyramid layer structure.

### Description

The function `ippPyramidLayerUp` is declared in the `ippcv.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function creates an upper pyramid layer *pDst* from the source image *pSrc*, that is, it performs upsampling of the source image and then applies the convolution kernel using the mirror border. Before calling `ippPyramidLayerUp` function, the pyramid layer structure *pState* should be initialized by calling the function `ippPyramidLayerUpInitAlloc` and the kernel and the upsampling ratio should be specified beforehand. The function can process images with *srcRoiSize* not greater than *srcRoiSize* parameter specified in the function `ippPyramidLayerUpInitAlloc`.



**NOTE.** This function uses the mirrored border.

---

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error if one of the specified pointers is <code>NULL</code> .



---

<code>ippStsSizeErr</code>	Indicates an error condition if <i>srcRoiSize</i> or <i>dstRoiSize</i> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> is less than <i>srcRoiSize.width * &lt;pixelSize&gt;</i> , or <i>dstStep</i> is less than <i>dstRoiSize.width * &lt;pixelSize&gt;</i> .
<code>ippStsNotEvenStepErr</code>	Indicates an error condition if one of the step values is not divisible by 4 for floating-point images, or by 2 for short-integer images.
<code>ippStsBadArgErr</code>	Indicates an error condition if <i>pState-&gt;rate</i> has wrong value.

### Example of Using General Pyramid Functions

The following example shows how different general pyramids functions can be used to create the Gaussian and Laplacian pyramids:

**Example**

```

void UsePyramids(Ipp32f *pSrc, IppiSize srcRoi, int srcStep, Ipp32f *pkernel, int kerSize) {
    float rate = 2.0f;
    IppiPyramid *gPyr; // pointer to Gaussian pyramid structure
    IppiPyramid *lPyr; // pointer to Laplacian pyramid structure
    // allocate pyramid structures
    ippiPyramidInitAlloc (&gPyr, 1000, srcRoi, rate);
    ippiPyramidInitAlloc (&lPyr, 1000, srcRoi, rate);
    {
        int i;
        IppiPyramidDownState_32f_C1R **gState = (IppiPyramidDownState_32f_C1R**)&(gPyr->pState);
        IppiPyramidUpState_32f_C1R **lState = (IppiPyramidUpState_32f_C1R**) &(lPyr->pState);
        Ipp32f **gImage = (Ipp32f**)(gPyr->pImage);
        Ipp32f **lImage = (Ipp32f**)(lPyr->pImage);
        IppiSize *pRoi = gPyr->pRoi;
        int *gStep = gPyr->pStep;
        int *lStep = lPyr->pStep;
        int level = gPyr->level;
        Ipp32f *ptr;
        int step;
        // allocate structures to calculate pyramid layers
        ippiPyramidLayerDownInitAlloc_32f_C1R
            (gState, srcRoi, rate, pkernel, kerSize, IPPI_INTER_LINEAR);
        ippiPyramidLayerUpInitAlloc_32f_C1R
            (lState, srcRoi, rate, pkernel, kerSize, IPPI_INTER_LINEAR);
        // build Gaussian pyramid with level+1 layers
        gImage[0] = pSrc;
        gStep[0] = srcStep;
        for (i=1; i<=level;i++) {
            gImage[i] = ippiMalloc_32f_C1(pRoi[i].width,pRoi[i].height,gStep+i);
            ippiPyramidLayerDown_32f_C1R
                (gImage[i-1], gStep[i-1], pRoi[i-1], gImage[i], gStep[i], pRoi[i], *gState);
        }
        // build Laplacian pyramid with level layers
        ptr = ippiMalloc_32f_C1(srcRoi.width,srcRoi.height,&step);
        for (i=level-1; i>=0; i--) {

```

```

    lImage[i] = ippiMalloc_32f_C1(pRoi[i].width,pRoi[i].height,lStep+i);
    ippiPyramidLayerUp_32f_C1R(gImage[i+1], gStep[i+1], pRoi[i+1], ptr, step, pRoi[i], *lState);
    ippiSub_32f_C1R(ptr, step, gImage[i], gStep[i], lImage[i], lStep[i], pRoi[i]);
}
ippiFree(ptr);
ippiPyramidLayerDownFree_32f_C1R(*gState);
ippiPyramidLayerUpFree_32f_C1R(*lState);
// use Gaussian and Laplacian pyramids
// free allocated images
for (i=1; i<=level; i++) {
    ippiFree(gImage[i]);
    ippiFree(lImage[i-1]);
}
}
// free pyramid structures
ippiPyramidFree (gPyr);
ippiPyramidFree (lPyr);
}

```

## Image Inpainting

The functions described in this section allows to restore the unknown image portions. They could be used to repair damaged parts of images and to remove some objects from images. Fast direct methods of inpainting that allow for run-time correcting of video frames are supported.

### InpaintInitAlloc

*Allocates memory and initializes a structure for image inpainting.*

#### Syntax

```

IppStatus ippiInpaintInitAlloc_8u_C1R(IppiInpaintState_8u_C1R** ppState, const Ipp32f*
pDist, int distStep, const Ipp8u* pMask, int maskStep, IppiSize roiSize, Ipp32f radius,
IppiInpaintFlag flags);

IppStatus ippiInpaintInitAlloc_8u_C3R(IppiInpaintState_8u_C3R** ppState, const Ipp32f*
pDist, int distStep, const Ipp8u* pMask, int maskStep, IppiSize roiSize, Ipp32f radius,
IppiInpaintFlag flags);

```

## Parameters

<i>ppState</i>	Double pointer to the state structure for the image inpainting.				
<i>pDist</i>	Pointer to the ROI of the image of distances.				
<i>distStep</i>	Distance in bytes between starts of consecutive lines in the image of distances.				
<i>pMsk</i>	Pointer to the mask image ROI.				
<i>mskStep</i>	Distance in bytes between starts of consecutive lines in the mask image.				
<i>roiSize</i>	Size of the image ROI in pixels.				
<i>radius</i>	Radius of the neighborhood used for inpainting.				
<i>flags</i>	Specifies algorithm for image inpainting; following values are possible: <table data-bbox="505 785 1122 856"> <tr> <td>IPP_INPAINT_TELEA</td><td>Telea algorithm;</td></tr> <tr> <td>IPP_INPAINT_NS</td><td>Navier-Stokes equation.</td></tr> </table>	IPP_INPAINT_TELEA	Telea algorithm;	IPP_INPAINT_NS	Navier-Stokes equation.
IPP_INPAINT_TELEA	Telea algorithm;				
IPP_INPAINT_NS	Navier-Stokes equation.				

## Description

The function `ippiInpaintInitAlloc` is declared in the `ippcv.h`. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function allocates memory and initializes the structure *ppState* for direct methods of image inpainting. This structure is used by the function `ippiInpaint` and can be applied to process images with ROI of the same size *roiSize*. Zero pixels of the mask image *pMsk* correspond to known image pixels, non-zero pixels - correspond to unknown image pixels that should be restored. The distance image *pDist* specifies the order of pixel inpainting. Values of unknown pixels are restored in ascending order in dependence on their distances. The parameter *radius* specifies the radius of the circular neighborhood that affects the restoration of the central pixel. The parameter specifies the methods of direct inpainting. Two methods are supported: Telea algorithm [ [Telea04](#) ], and Navier-Stokes equation [ [Bert01](#) ].

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <i>distStep</i> or <i>mskStep</i> is less than $roiSize.width * <pixelSize>$ .
<code>ippStsNotEvenStepErr</code>	Indicates an error condition if one of the step values is not divisible by 4 for floating-point images.
<code>ippStsBadArgErr</code>	Indicates an error condition if <i>radius</i> is less than 1, or <i>flags</i> has an illegal value.
<code>ippStsMemAllocErr</code>	Memory allocation error.

## InpaintFree

*Frees memory allocated for the structure for image inpainting.*

---

### Syntax

```
IppStatus ippiInpaintFree_8u_C1R(IppiInpaintState_8u_C1R* pState);
IppStatus ippiInpaintFree_8u_C3R(IppiInpaintState_8u_C3R* pState);
```

### Parameters

*pState*                                      Pointer to the state structure for the image inpainting.

### Description

The function `ippiInpaintFree` is declared in the `ippcv.h`. This function frees memory allocated by the function `ippiInpaintInitAlloc` for the inpainting structure *pState*.

### Return Values

`ippStsNoErr`                                      Indicates no error. Any other value indicates an error or a warning.  
`ippStsNullPtrErr`                                Indicates an error condition if the *pState* pointer is NULL.

## Inpaint

*Restores unknown image pixels.*

---

### Syntax

```
IppStatus ippiInpaint_8u_C1R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst, int dstStep,
IppiSize roiSize, IppiInpaintState_8u_C1R* pState);
IppStatus ippiInpaint_8u_C3R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst, int dstStep,
IppiSize roiSize, IppiInpaintState_8u_C3R* pState);
```

### Parameters

*pSrc*    Pointer to the source image ROI.  
*srcStep*    Distance in bytes between starts of consecutive lines in the source image.  
*pDst*    Pointer to the destination image ROI.  
*dstStep*    Distance in bytes between starts of consecutive lines in the destination image.  
*roiSize*    Size of the image ROI in pixels.  
*pState*    The pointer to the inpainting structure.

### Description

The function `ippiInpaint` is declared in the `ippcv.h`. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function reconstructs damaged part of the image, or removes a selected object (see [Figure “Image Inpainting”](#)). The image part to restore is defined by the mask that is created when the inpainting structure *pState* is initialized by the function [ippiInpaintInitAlloc](#). Different distant transforms can be used, but the Fast Marching Method ([ippiFastMarching](#)) provides the best results. The order of pixel restoration is defined by the distance through the initialization the inpainting structure *pState* by the function [ippiInpaintInitAlloc](#). Pixels are restored in according to the growing of their distance value. When a pixel is inpainted, it is treated as the known one.

Two algorithms of direct inpainting are supported (controlled by the parameter *flags* of the function [ippiInpaintInitAlloc](#)):

- image restoration of the unknown pixel by the weighted sum of approximations by known pixels in the neighborhood (*flags* = `IPP_INPAINT_TELEA`) [[Telea04](#),
- image restoration based on the Navier-Stokes equations (*flags* = `IPP_INPAINT_NS`) [[Bert01](#)].

The inpainting structure *pState* must be initialized by the function [ippiInpaintInitAlloc](#), and may be used to perform restoration of several different images of the same size *roiSize*.

### Image Inpainting

---



initial image

restored image

---

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value, or if differs from the corresponding parameter that is specified when the inpainting structure is initialized by the <a href="#">ippiInpaintInitAlloc</a> .
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> or <i>dstStep</i> is less than <i>roiSize.width</i> * <i>&lt; pixelSize &gt;</i> .

## Image Segmentation

This section describes the functions that perform image segmentation using different techniques. These functions allow to extract parts of the image that can be associated with objects of the real world. Watershed and gradient segmentation are region-based methods to split image into the distinctive areas.

Background/foreground segmentation allows for distinguishing between moving objects and stable areas of the background.

### LabelMarkersGetBufferSize

*Computes the size of the working buffer for the marker labeling.*

---

#### Syntax

```
ippStatus ippLabelMarkersGetBufferSize_8u_C1R(IppiSize roiSize, int* pBufferSize);
ippStatus ippLabelMarkersGetBufferSize_16u_C1R(IppiSize roiSize, int* pBufferSize);
```

#### Parameters

<i>roiSize</i>	Size of the source image ROI in pixels.
<i>pBufferSize</i>	Pointer to the computed size of the working buffer.

#### Description

The function `ippLabelMarkersGetBufferSize` is declared in the `ippcv.h`. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function computes the size of the working buffer required for the `ippLabelMarkers` function. The buffer with the length `pBufferSize[0]` can be used to segment images with width and/or height that is equal to or less than the corresponding field of the parameter `roiSize`.

#### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if the pointer <code>pBufferSize</code> is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <code>roiSize</code> has a field with zero or negative value.

### LabelMarkers

*Labels markers in image with different values.*

---

#### Syntax

```
ippStatus ippLabelMarkers_8u_C1R(Ipp8u* pMarker, int markerStep, IppiSize roiSize, int minLabel, int maxLabel, IppiNorm norm, int* pNumber, Ipp8u* pBuffer);
ippStatus ippLabelMarkers_16u_C1R(Ipp16u* pMarker, int markerStep, IppiSize roiSize, int minLabel, int maxLabel, IppiNorm norm, int* pNumber, Ipp8u* pBuffer);
```

## Parameters

<i>pMarker</i>	Pointer to the source and destination image ROI.				
<i>markerStep</i>	Distance in bytes between starts of consecutive lines in the source and destination image.				
<i>minLabel</i>	Minimal value of the marker label ( $0 < \text{minLabel} \leq \text{maxLabel}$ ).				
<i>maxLabel</i>	Maximal value of the marker label ( $\text{minLabel} \leq \text{maxLabel} < 255$ for 8-bit markers, and $\text{minLabel} \leq \text{maxLabel} < 65535$ for 16-bit markers).				
<i>roiSize</i>	Size of the source and destination image ROI in pixels.				
<i>norm</i>	Specifies type of the norm to form the mask for marker propagation: <table data-bbox="503 756 1209 840"> <tr> <td><code>ippiNormInf</code></td><td>Infinity norm (8-connectivity);</td></tr> <tr> <td><code>ippiNormL1</code></td><td>L1 norm (4-connectivity).</td></tr> </table>	<code>ippiNormInf</code>	Infinity norm (8-connectivity);	<code>ippiNormL1</code>	L1 norm (4-connectivity).
<code>ippiNormInf</code>	Infinity norm (8-connectivity);				
<code>ippiNormL1</code>	L1 norm (4-connectivity).				
<i>pNumber</i>	Pointer to the number of markers.				
<i>pBuffer</i>	Pointer to the working buffer.				

## Description

The function `ippiLabelMarkers` is declared in the `ippcv.h`. It operates with ROI (see Regions of Interest in Intel IPP).

This function labels markers in the image *pSrcDst* with different integer values. Each connected set of non-zero image pixels is treated as the separate marker. 4- or 8-connectivity can be used depending on the norm type. All pixels belonging to the same marker are set to the same value from the interval  $[\text{minLabel}, \text{maxLabel}]$ . Two markers can be labeled with the same value if the number of connected components exceeds  $\text{minLabel} - \text{maxLabel} + 1$ . The image with labeled markers can be used as the seed image for segmentation by functions `ippiSegmentWatershed` or `ippiSegmentGradient` functions.

The function requires the working buffer *pBuffer* whose size should be computed by the function `ippiLabelMarkersGetBufferSize` beforehand.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <i>markerStep</i> is less than $\text{roiSize.width} * < \text{pixelSize} >$ .
<code>ippStsNotEvenStepErr</code>	Indicates an error condition if <i>markerStep</i> for 16-bit images is not divisible by 2.
<code>ippStsBadArgErr</code>	Indicates an error condition if one of the <i>minLabel</i> , <i>maxLabel</i> , and <i>norm</i> has an illegal value.



## SegmentWatershedGetBufferSize

*Computes the size of the working buffer for the watershed segmentation.*

---

### Syntax

```
IppStatus ippiSegmentWatershedGetBufferSize_8u_C1R(IppiSize roiSize, int* pBufferSize);
IppStatus ippiSegmentWatershedGetBufferSize_8u16u_C1R(IppiSize roiSize, int* pBufferSize);
```

### Parameters

<i>roiSize</i>	Size of the source image ROI in pixels.
<i>pBufferSize</i>	Pointer to the computed size of the working buffer.

### Description

The function `ippiSegmentWatershedGetBufferSize` is declared in the `ippcv.h`. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function computes the size of the working buffer required for the `ippiSegmentWatershed` function. The buffer with the length `pBufferSize [0]` can be used to segment images with width and/or height that is equal to or less than the corresponding field of the parameter *roiSize*.

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if the pointer <i>pBufferSize</i> is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.

## SegmentWatershed

*Performs watershed image segmentation using markers.*

---

### Syntax

```
IppStatus ippiSegmentWatershed_8u_C1IR(const Ipp8u* pSrc, int srcStep, Ipp8u* pMarker,
int markerStep, IppiSize roiSize, IppiNorm norm, int flags, Ipp8u* pBuffer);
IppStatus ippiSegmentWatershed_8u16u_C1IR(const Ipp8u* pSrc, int srcStep, Ipp16u* pMarker,
int markerStep, IppiSize roiSize, IppiNorm norm, int flags, Ipp8u* pBuffer);
```

### Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pMarker</i>	Pointer to the ROI of the source and destination image of markers.

<i>markerStep</i>	Distance in bytes between starts of consecutive lines in the image of markers.								
<i>roiSize</i>	Size of the source and destination image ROI in pixels.								
<i>norm</i>	Specifies type of the norm to form the mask for marker propagation: <table> <tr> <td><code>ippiNormInf</code></td><td>Infinity norm (8-connectivity, 3x3 rectangular mask);</td></tr> <tr> <td><code>ippiNormL1</code></td><td>L1 norm (4-connectivity, 3x3 cross mask);</td></tr> <tr> <td><code>ippiNormL2</code></td><td>approximation of L2 norm (8-connectivity, 3x3 mask) [<a href="#">Bor86</a>];</td></tr> <tr> <td><code>ippiNormFM</code></td><td>fast marching distance [<a href="#">Telea04</a>];</td></tr> </table>	<code>ippiNormInf</code>	Infinity norm (8-connectivity, 3x3 rectangular mask);	<code>ippiNormL1</code>	L1 norm (4-connectivity, 3x3 cross mask);	<code>ippiNormL2</code>	approximation of L2 norm (8-connectivity, 3x3 mask) [ <a href="#">Bor86</a> ];	<code>ippiNormFM</code>	fast marching distance [ <a href="#">Telea04</a> ];
<code>ippiNormInf</code>	Infinity norm (8-connectivity, 3x3 rectangular mask);								
<code>ippiNormL1</code>	L1 norm (4-connectivity, 3x3 cross mask);								
<code>ippiNormL2</code>	approximation of L2 norm (8-connectivity, 3x3 mask) [ <a href="#">Bor86</a> ];								
<code>ippiNormFM</code>	fast marching distance [ <a href="#">Telea04</a> ];								
<i>flags</i>	Specifies the algorithm of segmentation; it is the logical sum of two values: mandatory value, one of the following values: <table> <tr> <td><code>IPP_SEGMENT_QUEUE</code></td><td>Priority queue is used to define the order of pixel processing;</td></tr> <tr> <td><code>IPP_SEGMENT_DISTANCE</code></td><td>distance transform is used for the segmentation;</td></tr> </table> and optional value, one of the following values: <table> <tr> <td><code>IPP_SEGMENT_BORDER_4</code></td><td>pixels of the 4-connectivity border between image segments are marked with value <code>IPP_MAX_8U</code> (255);</td></tr> <tr> <td><code>IPP_SEGMENT_BORDER_8</code></td><td>pixels of the 8-connectivity border between image segments are marked with value <code>IPP_MAX_8U</code> (255).</td></tr> </table>	<code>IPP_SEGMENT_QUEUE</code>	Priority queue is used to define the order of pixel processing;	<code>IPP_SEGMENT_DISTANCE</code>	distance transform is used for the segmentation;	<code>IPP_SEGMENT_BORDER_4</code>	pixels of the 4-connectivity border between image segments are marked with value <code>IPP_MAX_8U</code> (255);	<code>IPP_SEGMENT_BORDER_8</code>	pixels of the 8-connectivity border between image segments are marked with value <code>IPP_MAX_8U</code> (255).
<code>IPP_SEGMENT_QUEUE</code>	Priority queue is used to define the order of pixel processing;								
<code>IPP_SEGMENT_DISTANCE</code>	distance transform is used for the segmentation;								
<code>IPP_SEGMENT_BORDER_4</code>	pixels of the 4-connectivity border between image segments are marked with value <code>IPP_MAX_8U</code> (255);								
<code>IPP_SEGMENT_BORDER_8</code>	pixels of the 8-connectivity border between image segments are marked with value <code>IPP_MAX_8U</code> (255).								
<i>pBuffer</i>	Pointer to the working buffer.								

## Description

The function `ippiSegmentWatershed` is declared in the `ippcv.h`. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function performs image watershed segmentation with markers. Non-zero pixels of *pMarker* image belong to water source markers. Marker values propagate through the whole image according to the watershed algorithm. Image segments are formed by groups of connected *pMarker* pixels with the same value. The parameter *norm* controls marker propagation connectivity. Watershed segmentation is preferable for images with local minimums, for example, gradient images. Image markers generally correspond to these local minimums and can be created, for example, manually or using morphological reconstruction.

The parameter *flags* specifies how watershed segmentation is performed. This parameter is a logical sum of two values.

Required value specifies algorithm of segmentation. Possible values:

<code>IPP_SEGMENT_QUEUE</code>	specifies the classic watershed segmentation scheme with priority queue [ <a href="#">Vincent91</a> ].
--------------------------------	--

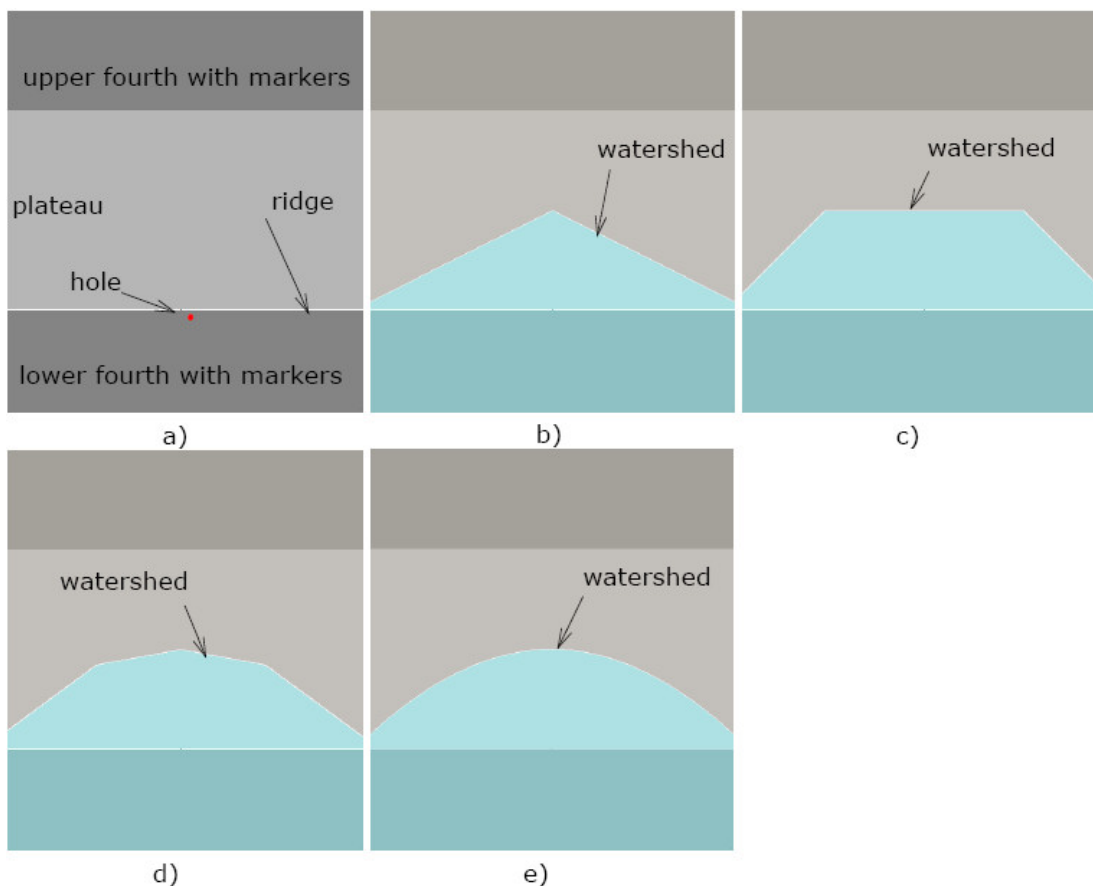
`IPP_SEGMENT_DISTANCE` specifies the watershed segmentation by calculating the topographic distance for each pixel [Lotufo00], [Meyer94].

Optional additional values of the *flags*: `IPP_SEGMENT_BORDER_4` and `IPP_SEGMENT_BORDER_8` specify the border of the segments. All pixels adjacent to the differently marked pixels are considered as border pixels, and their values are set to `IPP_MAX_8U` (255) for 8-bit markers, or `IPP_MAX_16U` (65535) for 16-bit markers. Note in this case value `IPP_MAX_8U` (`IPP_MAX_16U`) should not be used to mark segments. If these optional values are not specified, segments are formed without borders.

The function requires the working buffer *pBuffer* whose size should be computed by the function `ippiSegmentWatershedGetBufferSize` beforehand.

Figure “Watershed Segmentation with Different Norms” shows the plateau filling through the watershed segmentation with different values of the *norm* parameters. Initial image (a) has the labeled with markers upper and lower fourths with low pixel value, the central plateau between them, the ridge between the plateau and the lower fourth with one pixel hole in the center of it. The following pictures are segmentation results: b) - for L1 norm (block distance), c)- Linf norm (chessboard distance), d) - approximate L2 (Euclidian) norm [Bor86], e) Fast Marching distance.

### Watershed Segmentation with Different Norms



## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <code>roiSize</code> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if one of the <code>srcStep</code> or <code>markerStep</code> is less than <code>roiSize.width * &lt;pixelSize&gt;</code> .
<code>ippNotEvenStsStepErr</code>	Indicates an error condition if one of the <code>srcStep</code> or <code>markerStep</code> for 16-bit integer images is not divisible by 2.
<code>ippStsBadArgErr</code>	Indicates an error condition if <code>norm</code> has an illegal value.

## SegmentGradientGetBufferSize

*Computes the size of the working buffer for the gradient segmentation.*

---

### Syntax

```
ippStatus ippiSegmentGradientGetBufferSize_8u_C1R(IppiSize roiSize, int* pBufferSize);
ippStatus ippiSegmentGradientGetBufferSize_8u_C3R(IppiSize roiSize, int* pBufferSize);
```

### Parameters

<code>roiSize</code>	Size of the source image ROI in pixels.
<code>pBufferSize</code>	Pointer to the computed size of the working buffer.

### Description

The function `ippiSegmentGradientGetBufferSize` is declared in the `ippcv.h`. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function computes the size of the working buffer required for the `ippiSegmentGradient` function. The buffer with the length `pBufferSize [0]` can be used to segment images with width and/or height that is equal to or less than the corresponding field of the parameter `roiSize`.

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if the pointer <code>pBufferSize</code> is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <code>roiSize</code> has a field with zero or negative value.

## SegmentGradient

*Performs image segmentation by region growing to the least gradient direction.*

---

### Syntax

```
IppStatus ippiSegmentGradient_8u_C1IR(const Ipp8u* pSrc, int srcStep, Ipp8u* pMarker, int markerStep, IppiSize roiSize, IppiNorm norm, int flags, Ipp8u* pBuffer);
```

```
IppStatus ippiSegmentGradient_8u_C3IR(const Ipp8u* pSrc, int srcStep, Ipp8u* pMarker, int markerStep, IppiSize roiSize, IppiNorm norm, int flags, Ipp8u* pBuffer);
```

### Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
	<i>pMarker</i> Pointer to the ROI of the source and destination image of markers.
	<i>markerStep</i> Distance in bytes between starts of consecutive lines in the image of markers.
<i>roiSize</i>	Size of the source and destination image ROI in pixels.
<i>norm</i>	Specifies type of the norm to form the mask for marker propagation:
	<i>ippiNormInf</i> Infinity norm (8-connectivity, 3x3 rectangular mask);
	<i>ippiNormL1</i> L1 norm (4-connectivity, 3x3 cross mask);
<i>flags</i>	optional flag:
	<i>IPP_SEGMENT_BORDER_4</i> pixels of the 4-connectivity border between image segments are marked with value $(IPP\_MAX\_8U) - 1$ (254).
	<i>IPP_SEGMENT_BORDER_8</i> pixels of the 8-connectivity border between image segments are marked with value $(IPP\_MAX\_8U) - 1$ (254).
<i>pBuffer</i>	Pointer to the working buffer.

### Description

The function `ippiSegmentGradient` is declared in the `ippcv.h`. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function performs image segmentation by region growing with markers. Non-zero pixels of *pMarker* image belong to initial image regions. Marker values propagate through the whole image in the direction of the least value of the absolute value of the image gradient. For 3-channel image the gradient is calculated as the maximum

of channel gradients. Image segments are formed by groups of connected *pMarker* pixels with the same value. The parameter *norm* controls marker propagation connectivity. Gradient segmentation is generally done for an image without explicit calculation of the image gradient. [Meyer92]

If `IPP_SEGMENT_BORDER` flag is defined, then the pixels adjacent to differently marked pixels are assumed to be border pixels and are set to a special value (254). This value must not be used to mark segments in this case.

Another special value (255) is used inside the function and can not be used to mark segment in any case.

The function requires the working buffer *pBuffer* whose size should be computed by the function `ippiSegment-GradientGetBufferSize` beforehand.

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if one of the <i>srcStep</i> or <i>markerStep</i> is less than <i>roiSize.width</i> * < <i>pixelSize</i> >.
<code>ippStsBadArgErr</code>	Indicates an error condition if <i>norm</i> has an illegal value.

## BoundSegments

*Marks pixels belonging to segment boundaries.*

---

### Syntax

```
IppStatus ippiBoundSegments_8u_C1IR(Ipp8u* pMarker, int markerStep, IppiSize roiSize,
Ipp8u val, IppiNorm norm);
```

```
IppStatus ippiBoundSegments_16u_C1IR(Ipp16u* pMarker, int markerStep, IppiSize roiSize,
Ipp16u val, IppiNorm norm);
```

### Parameters

<i>pMarker</i>	Pointer to the ROI of the source and destination image of markers.
<i>markerStep</i>	Distance in bytes between starts of consecutive lines in the image of markers.
<i>roiSize</i>	Size of the source and destination image ROI in pixels.
<i>val</i>	Value of the boundary pixel.
<i>norm</i>	Specifies type of the norm for pixel neighborhood:
<code>ippiNormInf</code>	Infinity norm (8-connectivity);
<code>ippiNormL1</code>	L1 norm (4-connectivity).

### Description

The function `ippiBoundSegments` is declared in the `ippcv.h`. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function detects segment boundaries in the *pMarker* image and sets border pixels to the value *val*. A segment is the set of connected pixels of the *pMarker* image with the same value not equal to *val*. After boundaries are marked, the *pMarker* image does not contain any pair of adjacent in *norm* pixels with the same value that not equal to *val*.

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <i>markerStep</i> is less than <i>roiSize.width</i> * < <i>pixelSize</i> >.
<code>ippNotEvenStsStepErr</code>	Indicates an error condition if <i>markerStep</i> for 16-bit integer images is not divisible by 2.
<code>ippStsBadArgErr</code>	Indicates an error condition if <i>norm</i> has an illegal value.

## ForegroundHistogramInitAlloc

*Allocates memory and initializes a state structure for foreground/background segmentation using histograms.*

### Syntax

```
ippStatus ippiForegroundHistogramInitAlloc_8u_C1R(const Ipp8u* pSrc, int srcStep, IppiSize roiSize, IppFGHistogramModel* pModel, IppFGHistogramState_8u_C1R** ppState)
```

```
ippStatus ippiForegroundHistogramInitAlloc_8u_C3R(const Ipp8u* pSrc, int srcStep, IppiSize roiSize, IppFGHistogramModel* pModel, IppFGHistogramState_8u_C3R** ppState)
```

### Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>roiSize</i>	Size of the source and destination image ROI in pixels.
<i>pModel</i>	Pointer to the structure of the histogram statistical model.
<i>ppState</i>	Pointer to the pointer to the segmentation state structure.

### Description

The function `ippiForegroundHistogramInitAlloc` is declared in the `ippcv.h`. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function allocates memory, initializes the state structure for the histogram modeling of foreground and background and returns the pointer *ppState* to it. It is used by the functions [ippiForegroundHistogram](#) and [ippiForegroundHistogramUpdate](#) that detect foreground pixels and update the statistical model for the segmentation.

The histogram statistical model implements detecting both stationary and moving background objects [Li03]. The model keeps histograms of quantized pixel values and quantized velocities of pixel value changes.

The histogram statistical model is defined by following fields of *pModel* structure:

<i>valueQuant</i>	Number of quantization levels for pixel values (degree of 2, $valueQuant \geq 2$ ).
<i>changeQuant</i>	Number of quantization levels for the speed of pixel value change (degree of 2, $valueQuant \geq 2$ ).
<i>valueUse</i>	Number of most frequent pixel value levels used for classification ( $valueUse \geq valueAll$ ).
<i>valueAll</i>	Number of all pixel value levels monitored by the model ( $valueAll > 0$ ).
<i>changeUse</i>	Number of most frequent speed levels used for classification ( $changeUse \geq changeAll$ ).
<i>changeAll</i>	Number of all speed levels monitored by the model ( $changeAll > 0$ ).
<i>updBGProb</i>	Speed of updating of the background probability ( $0.5 > updBGProb > 0$ ).
<i>updBGRef</i>	Speed of updating of the reference background image ( $0.5 > updBGRef > 0$ ).
<i>numFrame</i>	Number of background frames for adjustment of shadows searching. Only for 3-channels images.
<i>detectionRate</i>	Speed of shadows searching. Only for 3-channels images ( $0 > detectionRate > 1$ ).
<i>brightnessDistortion</i>	Lower bound of the normalized brightness distortion for shadows. Only for 3-channels images ( $0 > brightnessDistortion > 1$ ).
<i>shadowBG</i>	Flag of shadows searching. If <i>shadowBG</i> is 1, then pixels belonging to colored shadows are marked as background, otherwise they may be included in foreground areas. Only for 3-channels images (0,1).

*valueAll* (*changeAll*) are the most frequent levels; they are monitored by the model. First of them *valueUse* (*changeUse*) are used for foreground/background classification. The probabilities to classify pixels as background and pixel values of the reference background image are updated using one pole IIR filters with parameters *updBGProb* and *updBGRef* respectively. If *shadowBG* is set to 1, then the search of colored shadows is performed for three-channel images. In this case foreground pixels, that are detected as shadow, are marked as background pixels.

The structure can be used to process image whose width and height are equal to or less than corresponding fields of *roiSize*.

## Return Values

<i>ippStsNoErr</i>	Indicates no error. Any other value indicates an error or a warning.
<i>ippStsNullPtrErr</i>	Indicates an error condition if one of the specified pointers is <code>NULL</code> .



<code>ippStsSizeErr</code>	Indicates an error condition if <code>roiSize</code> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if one of the <code>srcStep</code> is less than <code>roiSize.width * &lt;pixelSize&gt;</code> .
<code>ippStsBadArgErr</code>	Indicates an error condition if <code>pModel</code> has an illegal value.

## ForegroundHistogramFree

*Frees memory allocated for the foreground/background segmentation structure.*

---

### Syntax

```
IppStatus ippForegroundHistogramFree_8u_C1R(IppFGHistogramState_8u_C1R* pState);
IppStatus ippForegroundHistogramFree_8u_C3R(IppFGHistogramState_8u_C3R* pState);
```

### Parameters

`pState` Pointer to the segmentation state structure.

### Description

The function `ippForegroundHistogramFree` is declared in the `ippcv.h`. This function frees memory allocated by the function `ippForegroundHistogramInitAlloc` for the state structure `pState` for the histogram modeling of foreground and background.

### Return Values

`ippStsNoErr` Indicates no error. Any other value indicates an error or a warning.  
`ippStsNullPtrErr` Indicates an error if `pState` is `NULL`.

## ForegroundHistogram

*Calculates foreground mask using histograms.*

---

### Syntax

```
IppStatus ippForegroundHistogram_8u_C1R(const Ipp8u* pSrc, int srcStep, Ipp8u* pMask,
int maskStep, IppiSize roiSize, IppFGHistogramState_8u_C1R* pState);
IppStatus ippForegroundHistogram_8u_C3R(const Ipp8u* pSrc, int srcStep, Ipp8u* pMask,
int maskStep, IppiSize roiSize, IppFGHistogramState_8u_C3R* pState);
```

### Parameters

`pSrc` Pointer to the source image ROI.  
`srcStep` Distance in bytes between starts of consecutive lines in the source image.  
`pMask` Pointer to the ROI of the destination foreground mask image.

<i>maskStep</i>	Distance in bytes between starts of consecutive lines in the foreground mask image.
<i>roiSize</i>	Size of the source and destination image ROI in pixels.
<i>pState</i>	Pointer to the segmentation state structure.

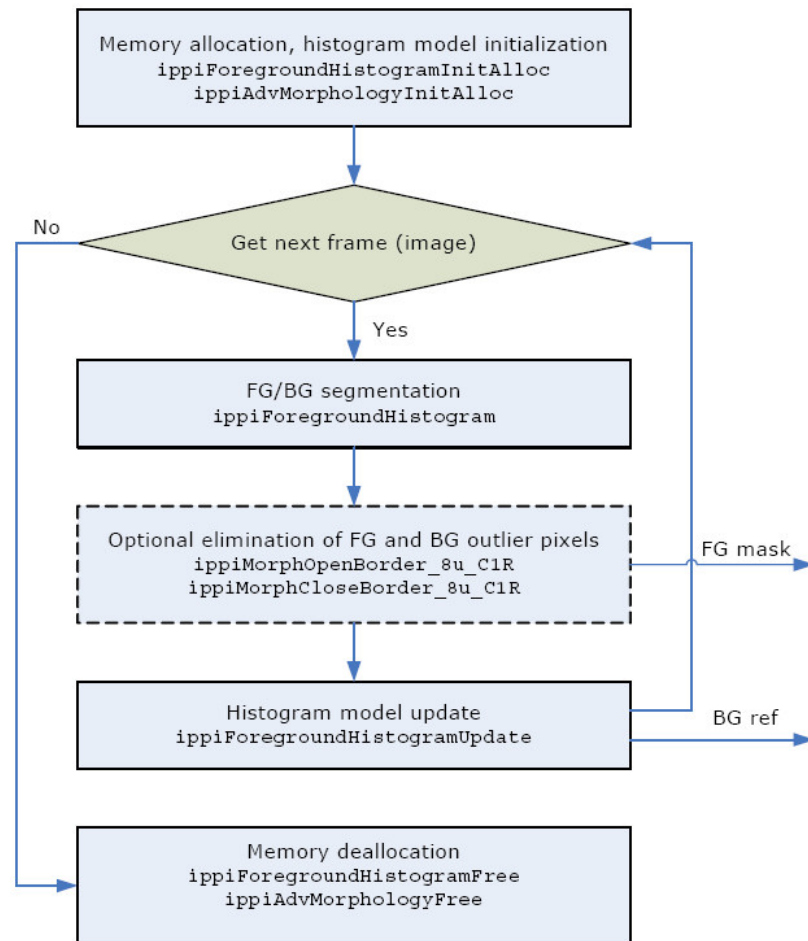
### Description

The function `ippiForegroundHistogram` is declared in the `ippcv.h`. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This foreground/background segmentation algorithm processes the sequence of images and trains statistical models for foreground and background pixels for this sequence. The algorithm applies the model to each image, classifies its pixels as belonging to the foreground or background of the scene, and then updates the model with the image data. [Figure "Foreground/Background Segmentation Using Histogram Model"](#) shows the scheme of the algorithm using the statistical histogram model for each pixel [Li03]. The function `ippiForegroundHistogram` applies the histogram statistical model to the image *pSrc*, and sets pixels of *pMask* image corresponding to the background pixels of the *pSrc* to zero, and foreground pixels - to non-zero value. Then the statistical model is updated by the function `ippiForegroundHistogramUpdate`.

The function `ippiForegroundHistogram` can process images with `roiSize` that is not greater than `roiSize` specified in the function `ippiForegroundHistogramInitAlloc`.

### Foreground/Background Segmentation Using Histogram Model



### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <code>roiSize</code> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if one of the <code>srcStep</code> or <code>maskStep</code> is less than <code>roiSize.width * &lt; pixelSize &gt;</code> .

## ForegroundHistogramUpdate

*Updates histogram statistical model for foreground segmentation.*

---

### Syntax

```
IppStatus ippiForegroundHistogramUpdate_8u_C1R(const Ipp8u* pSrc, int srcStep, const Ipp8u* pMask, int maskStep, Ipp8u* pRef, int refStep, IppiSize roiSize, IppFGHistogramState_8u_C1R* pState);
```

```
IppStatus ippiForegroundHistogramUpdate_8u_C3R(const Ipp8u* pSrc, int srcStep, const Ipp8u* pMask, int maskStep, Ipp8u* pRef, int refStep, IppiSize roiSize, IppFGHistogramState_8u_C3R* pState);
```

### Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pMask</i>	Pointer to the ROI of the foreground mask image.
<i>maskStep</i>	Distance in bytes between starts of consecutive lines in the foreground mask image.
<i>pRef</i>	Pointer to the ROI of the source and destination reference background mask image.
<i>refStep</i>	Distance in bytes between starts of consecutive lines in the reference background mask image.
<i>roiSize</i>	Size of the source and destination image ROI in pixels.
<i>pState</i>	Pointer to the segmentation state structure.

### Description

The function `ippiForegroundHistogramUpdate` is declared in the `ippcv.h`. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function updates the histogram statistical model using *pSrc* image and foreground mask *pMask*. Between calls of the function `ippiForegroundHistogram` and `ippiForegroundHistogramUpdate` functions, the mask image *pMask* is generally processed to delete foreground and background outlier pixels. When the model is updated, it is ready to process the next image.

This function uses the reference background image *pRef* that is initialized during the first call of the function. This image is updated after each call of the function `ippiForegroundHistogramUpdate`.

The function `ippiForegroundHistogram` can process images with *roiSize* that is not greater than *roiSize* specified in the function `ippiForegroundHistogramInitAlloc`.

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointers is <code>NULL</code> .

<code>ippStsSizeErr</code>	Indicates an error condition if <code>roiSize</code> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if one of the <code>srcStep</code> , <code>maskStep</code> , or <code>refStep</code> is less than <code>roiSize.width * &lt; pixelSize &gt;</code> .

## ForegroundGaussianInitAlloc

*Allocates memory and initializes a state structure for foreground/background segmentation using Gaussian mixtures.*

### Syntax

```
IppStatus ippForegroundGaussianInitAlloc_8u_C1R(const Ipp8u* pSrc, int srcStep, IppiSize
roiSize, IppFGGaussianModel* pModel, IppFGGaussianState_8u_C1R** ppState);

IppStatus ippForegroundGaussianInitAlloc_8u_C3R(const Ipp8u* pSrc, int srcStep, IppiSize
roiSize, IppFGGaussianModel* pModel, IppFGGaussianState_8u_C3R** ppState);
```

### Parameters

<code>pSrc</code>	Pointer to the source image ROI.
<code>srcStep</code>	Distance in bytes between starts of consecutive lines in the source image.
<code>roiSize</code>	Size of the source and destination image ROI in pixels.
<code>pModel</code>	Pointer to the structure of the Gaussian statistical model.
<code>ppState</code>	Pointer to the pointer to the segmentation state structure.

### Description

The function `ippForegroundGaussianInitAlloc` is declared in the `ippcv.h`. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function allocates memory, initializes the state structure for the foreground and background modeling by Gaussian mixtures and returns the pointer `ppState` to it. This structure is used by the function `ippForegroundGaussian` that detects foreground pixels and updates the statistical model for the segmentation.

The Gaussian mixture statistical model is defined by following fields of `pModel` structure:

<code>numGauss</code>	Number of components in the Gaussian mixture for each frame (usually, 3-5).
<code>priorBack</code>	Minimum prior probability of the background.
<code>updBGProb</code>	Speed of updating of parameters of selected mixture component ( $0.5 > \text{updBGProb} > 0$ ).
<code>winSize</code>	Number of last frames affecting model parameter update (100-200).
<code>numFrame</code>	Number of background frames for adjustment of shadows searching. Only for 3-channels images.

<i>detectionRate</i>	Speed of shadows searching. Only for 3-channels images ( $0 > \text{detectionRate} > 1$ ).
<i>brightnessDistortion</i>	Lower bound of the normalized brightness distortion for shadows. Only for 3-channels images ( $0 > \text{brightnessDistortion} > 1$ ).
<i>shadowBG</i>	Flag of shadows searching. If <i>shadowBG</i> is 1, then pixels belonging to colored shadows are marked as background, otherwise they may be included in foreground areas. Only for 3-channels images (0,1).

The Gaussian mixture statistical model models each background pixel by a Gaussian mixture with a small number of components *numGauss* [Kadew01]. Most important components with sum of weights greater than *priorBack* are used to model the background. Mixture components are updated while processing each new frame. The learning speed of the component that matches a background pixel is defined by *updBGChange*. All mixture parameters are then updated using statistics from last *winSize* frames. If *shadowBG* is set to 1, then the search of colored shadows is performed for three-channel images. In this case foreground pixels, that are detected as shadow, are marked as background pixels.

The structure can be used to process image whose width and height are equal to or less than corresponding fields of *roiSize*.

### Return Values

<i>ippStsNoErr</i>	Indicates no error. Any other value indicates an error or a warning.
<i>ippStsNullPtrErr</i>	Indicates an error condition if one of the specified pointers is <code>NULL</code> .
<i>ippStsSizeErr</i>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.
<i>ippStsStepErr</i>	Indicates an error condition if one of the <i>srcStep</i> is less than $\text{roiSize.width} * < \text{pixelSize} >$ .

## ForegroundGaussianFree

*Frees memory allocated for the foreground/background segmentation structure.*

---

### Syntax

```
IppStatus ippiForegroundGaussianFree_8u_C1R(IppFGGaussianState_8u_C1R* pState);
IppStatus ippiForegroundGaussianFree_8u_C3R(IppFGGaussianState_8u_C3R* pState);
```

### Parameters

*pState*                      Pointer to the segmentation state structure.

### Description

The function `ippiForegroundGaussianFree` is declared in the `ippcv.h`. This function frees memory allocated by the function `ippiForegroundGaussianInitAlloc` for the state structure *pState* for the Gaussian modeling of foreground and background.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error if <code>pState</code> is <code>NULL</code> .

## ForegroundGaussian

*Calculates foreground mask using Gaussians.*

### Syntax

```
ippStatus ippiForegroundGaussian_8u_C1R(const Ipp8u* pSrc, int srcStep, Ipp8u* pRef, int
refStep, Ipp8u* pDst, int dstStep, IppiSize roiSize, IppFGGaussianState_8u_C1R* pState);

ippStatus ippiForegroundGaussian_8u_C1R(const Ipp8u* pSrc, int srcStep, Ipp8u* pRef, int
refStep, Ipp8u* pDst, int dstStep, IppiSize roiSize, IppFGGaussianState_8u_C1R* pState);
```

### Parameters

<code>pSrc</code>	Pointer to the source image ROI.
<code>srcStep</code>	Distance in bytes between starts of consecutive lines in the source image.
<code>pRef</code>	Pointer to the ROI of the destination reference background mask image.
<code>refStep</code>	Distance in bytes between starts of consecutive lines in the reference background mask image.
<code>pDst</code>	Pointer to the ROI of the destination image.
<code>dstStep</code>	Distance in bytes between starts of consecutive lines in the destination image.
<code>roiSize</code>	Size of the source and destination image ROI in pixels.
<code>pState</code>	Pointer to the segmentation state structure.

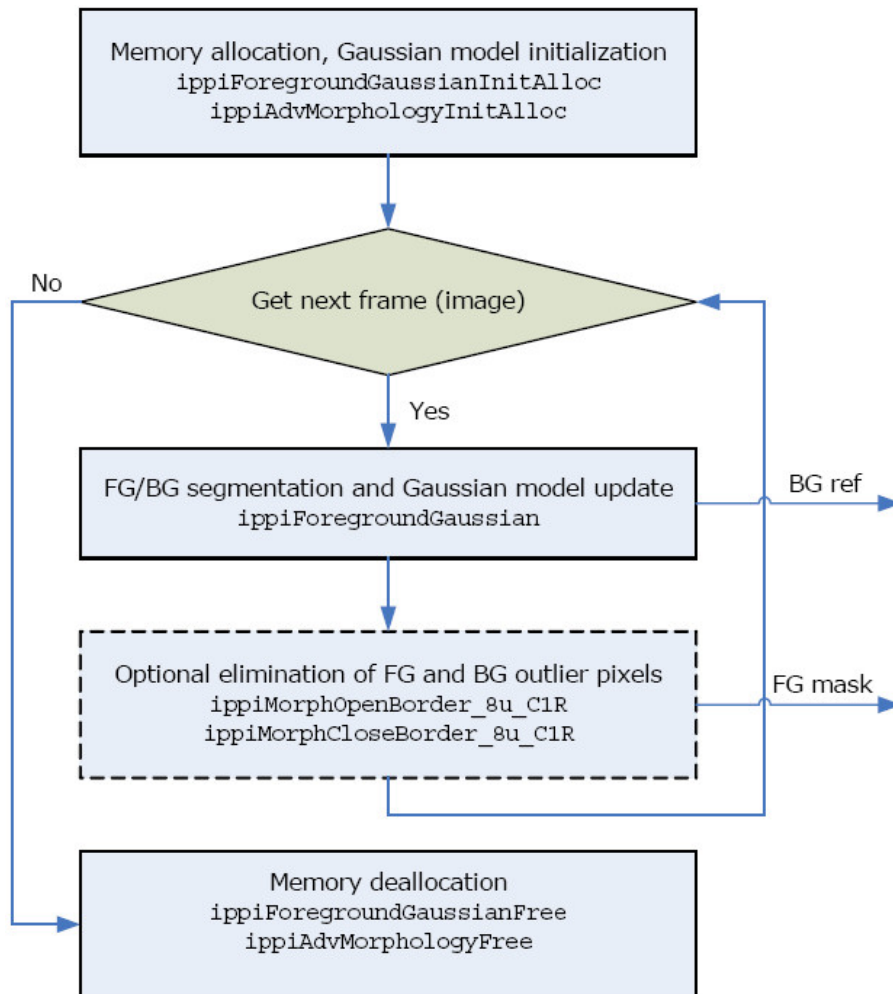
### Description

The function `ippiForegroundGaussian` is declared in the `ippcv.h`. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This foreground/background segmentation algorithm processes the sequence of images and trains statistical models for foreground and background pixels for this sequence. The algorithm applies the model to each image, classifies its pixels as belonging to the foreground or background of the scene, and then updates the model with the image data. [Figure "Foreground/Background Segmentation Using the Gaussian Model"](#) shows the scheme of the algorithm using the Gaussian mixture statistical model for each pixel [Kadew01]. The function `ippiForegroundGaussian` applies the Gaussian mixture statistical model to the image `pSrc`, and sets pixels of `pMask` image corresponding to the background pixels of the `pSrc` to zero, and foreground pixels - to non-zero value. Then the statistical model is updated. The mask image `pMask` is usually processed to delete foreground and background outlier pixels. The function calculates also the reference background image `pRef`.

The function `ippiForegroundGaussian` can process images with `roiSize` that is not greater than `roiSize` specified in the function `ippiForegroundGaussianInitAlloc`.

### Foreground/Background Segmentation Using the Gaussian Model



### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <code>roiSize</code> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if one of the <code>srcStep</code> , <code>dstStep</code> , or <code>refStep</code> is less than <code>roiSize.width * &lt;pixelSize&gt;</code> .

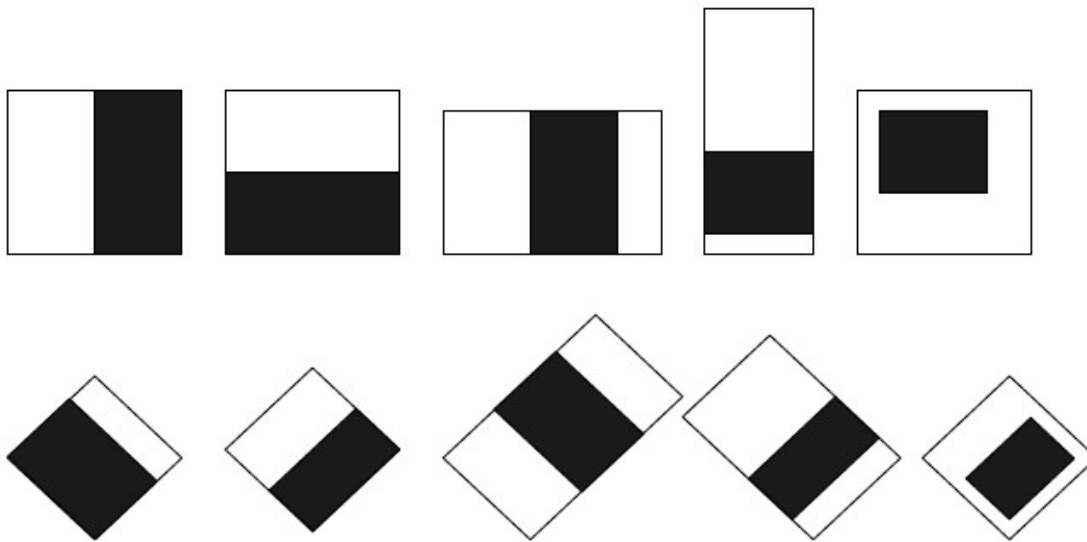


## Pattern Recognition

### Object Detection Using Haar-like Features

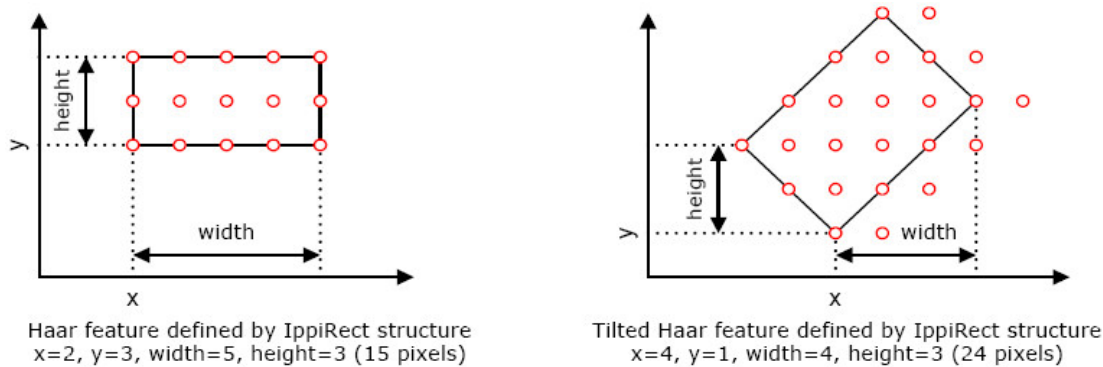
The object detector described in [ Viola01] and [ Lein02] is based on Haar classifiers. Each classifier uses  $k$  rectangular areas (Haar features) to make decision if the region of the image looks like the predefined image or not. Figure "Types of Haar Features" shows different types of Haar features.

#### Types of Haar Features



In the Intel IPP Haar features are represented using `IppiRect` structure. Figure "Representing Haar Features" shows how it can be done for common and tilted features.

#### Representing Haar Features



When the classifier  $\kappa_t$  is applied to the pixel  $(i, j)$  of the image  $A$ , it yields the value  $val1(t)$  if

$$\sum_{i=1}^k \left( w_1 \cdot \sum_{u=i+R_1Y}^{R_1Y+R_1height-1} \sum_{v=j+R_1YX}^{R_1Y+R_1width-1} A_{uv} \right) < norm(i, j) \cdot threshold(t)$$

and  $val2(t)$  otherwise.

Here  $w_1$  is a feature weight,  $norm(i, j)$  is the norm factor (generally the standard deviation on the rectangle containing all features),  $threshold(t)$ ,  $val1(t)$  and  $val2(t)$  are parameters of the classifier. For fast computation the integral representation of an image is used. Haar classifiers are organized in sequences called *stages* (*classification stages*). The stage value is the sum of its classifier values. During feature detecting stages are consequently applied to the region of the image until the stage value becomes less than the threshold value or all stages are passed.

The use of the Intel IPP pattern recognition functions is demonstrated in the face detecting sample. See *Intel IPP JPEG Computer Vision Samples* downloadable from

<http://www.intel.com/cd/software/products/asmo-na/eng/220046.htm>.

## HaarClassifierInitAlloc

*Allocates memory and initializes the structure for standard Haar classifiers.*

---

### Syntax

```
IppStatus ippiHaarClassifierInitAlloc_32f(IppiHaarClassifier_32f** pState, const IppiRect*
pFeature, const Ipp32f* pWeight, const Ipp32f* pThreshold, const Ipp32f* pVal1, const
Ipp32f* pVal2, const int* pNum, int length);
```

```
IppStatus ippiHaarClassifierInitAlloc_32s(IppiHaarClassifier_32s** pState, const IppiRect*
pFeature, const Ipp32s* pWeight, const Ipp32s* pThreshold, const Ipp32s* pVal1, const
Ipp32s* pVal2, const int* pNum, int length);
```

### Parameters

<i>pState</i>	Pointer to the pointer to the Haar classifier structure.
<i>pFeature</i>	Pointer to the array of features.
<i>pWeight</i>	Pointer to the array of feature weights.
<i>pThreshold</i>	Pointer to the array of classifier threshold values.
<i>pVal1, pVal2</i>	Pointers to the arrays of classifier result values.
<i>pNum</i>	Pointer to the array of classifier lengths.
<i>length</i>	Number of classifiers in the stage.

## Description

The function `ippiHaarClassifierInitAlloc` is declared in the `ippcv.h` file. This function allocates memory and initializes the structure for the sequence of Haar classifiers - classification stage.  $i$ -th classifier in the stage has  $pNum[i]$  rectangular features. Each feature is defined by the certain rectangle with horizontal and vertical sides. The length of vectors  $pThreshold$ ,  $pVal1$ , and  $pVal2$  is equal to  $length$ , the length of vectors  $pFeature$ ,  $pWeight$  is equal to:

$$length - 1 \sum_{i=0} pNum[i]$$

Result of applying classifiers to the image is computed using [the formula in the section "Object Detection Using Haar-like Features"](#).

All features of the classifier initialized by the function `ippiHaarClassifierInitAlloc` have vertical and horizontal sides (left part of [Figure "Representing Haar Features"](#)). Some of these features later can be tilted using the function `ippiTiltHaarFeatures`.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error if one of the specified pointer is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if $length$ or one of $pNum[i]$ is less than or equal to 0.
<code>ippStsBadArgErr</code>	Indicates an error condition if one of features is defined incorrectly.
<code>ippStsMemAllocErr</code>	Memory allocation error

## TiltedHaarClassifierInitAlloc

*Allocates memory and initializes the structure for tilted Haar classifiers.*

## Syntax

```
IppStatus ippiTiltedHaarClassifierInitAlloc_32f(IppiHaarClassifier_32f** pState, const IppiRect* pFeature, const Ipp32f* pWeight, const Ipp32f* pThreshold, const Ipp32f* pVal1, const Ipp32f* pVal2, const int* pNum, int length);
```

```
IppStatus ippiTiltedHaarClassifierInitAlloc_32s(IppiHaarClassifier_32s** pState, const IppiRect* pFeature, const Ipp32s* pWeight, const Ipp32s* pThreshold, const Ipp32s* pVal1, const Ipp32s* pVal2, const int* pNum, int length);
```

## Parameters

$ppState$	Pointer to the pointer to the Haar classifier structure.
$pFeature$	Pointer to the array of features.

<i>pWeight</i>	Pointer to the array of feature weights.
<i>pThreshold</i>	Pointer to the array of classifier threshold values.
<i>pVal1, pVal2</i>	Pointers to the arrays of classifier result values.
<i>pNum</i>	Pointer to the array of classifier lengths.
<i>length</i>	Number of classifiers in the stage.

## Description

The function `ippiHaarClassifierInitAlloc` is declared in the `ippcv.h` file. This function allocates memory and initializes the structure for the sequence of Haar classifiers - classification stage. *t*-th classifier in the stage has `pNum[i]` rectangular features. Each feature is defined by the certain rectangle with sides tilted by 45 degrees. The points with minimum and maximum row numbers should be specified. The length of vectors *pFeature*, *pWeight*, *pThreshold*, *pVal1*, and *pVal2* is equal to:

$$length - 1 \\ \sum_{i=0} pNum[i]$$

Result of applying classifiers to the image are computed using the [the formula in the section "Object Detection Using Haar-like Features"](#).

All features of the classifier initialized by the function `ippiHaarClassifierInitAlloc` have tilted sides (right part of [Figure "Representing Haar Features"](#)).

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointer is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <i>length</i> or one of <code>pNum[i]</code> is less than or equal to 0; or if the sum of all elements of <i>pNum</i> is not equal to <i>length</i> .
<code>ippStsBadArgErr</code>	Indicates an error condition if one of features is defined incorrectly.
<code>ippStsMemAllocErr</code>	Memory allocation error

## HaarClassifierFree

Frees memory allocated for the Haar classifier structure.

### Syntax

```
ippStatus ippiHaarClassifierFree_32f(IppiHaarClassifier_32f* pState);
ippStatus ippiHaarClassifierFree_32s(IppiHaarClassifier_32s* pState);
```

### Parameters

<i>pState</i>	Pointer to the Haar classifier structure.
---------------	---

## Description

The function `ippiHaarClassifierFree` is declared in the `ippcv.h` file. This function frees memory allocated for the Haar classifier structure `pState` by the function `ippiHaarClassifierInitAlloc` or `ippiTiltedHaarClassifierInitAlloc`.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if the <code>pState</code> pointer is <code>NULL</code> .

## GetHaarClassifierSize

*Returns the size of the Haar classifier.*

---

## Syntax

```
IppStatus ippiGetHaarClassifierSize_32f(IppiHaarClassifier_32f* pState, IppiSize* pSize);
IppStatus ippiGetHaarClassifierSize_32s(IppiHaarClassifier_32s* pState, IppiSize* pSize);
```

## Parameters

<code>pState</code>	Pointer to the Haar classifier structure.
<code>pSize</code>	Pointer to the size of Haar classifier structure.

## Description

The function `ippiGetHaarClassifierSize` is declared in the `ippcv.h` file. This function computes the minimum size of the window containing all features of the Haar classifier described by the `pState`.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if the <code>pState</code> pointer is <code>NULL</code> .

## TiltHaarFeatures

*Modifies a Haar classifier by tilting specified features.*

---

## Syntax

```
IppStatus ippiTiltHaarFeatures_32f(const Ipp8u* pMask, int flag, IppiHaarClassifier_32f* pState);
IppStatus ippiTiltHaarFeatures_32s(const Ipp8u* pMask, int flag, IppiHaarClassifier_32s* pState);
```

## Parameters

<code>pMask</code>	Pointer to the mask vector.
<code>flag</code>	Flag to choose the direction of feature tilting.

*pState* Pointer to the Haar classifier structure.

## Description

The function `ippiTiltHaarFeatures` is declared in the `ippcv.h` file. This function tilts specified features of the Haar classifier created by the function `ippiHaarClassifierInitAlloc`. Non-zero elements of previously prepared vector *pMask* indicates the features that will be tilted. The *flag* parameter specifies how the features are tilted: if its value is equal to 0, the feature is tilted around the left top corner clockwise, if it is equal to 1, the feature is tilted around the bottom left corner counter-clockwise.

This mixed classifier containing both common and tilted features can be used by the function `ippiApplyMixed-HaarClassifier`.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointer is <code>NULL</code> .
<code>ippStsBadArgErr</code>	Indicates an error condition if the classifier is tilted already.

## ApplyHaarClassifier

*Applies a Haar classifier to an image.*

---

## Syntax

```
ippStatus ippiApplyHaarClassifier_32f_C1R(const Ipp32f* pSrc, int srcStep, const Ipp32f*
pNorm, int normStep, Ipp8u* pMask, int maskStep, IppiSize roiSize, int* pPositive, Ipp32f
threshold, IppiHaarClassifier_32f* pState);
```

```
ippStatus ippiApplyHaarClassifier_32s32f_C1R(const Ipp32s* pSrc, int srcStep, const Ipp32f*
pNorm, int normStep, Ipp8u* pMask, int maskStep, IppiSize roiSize, int* pPositive, Ipp32f
threshold, IppiHaarClassifier_32f* pState);
```

```
ippStatus ippiApplyHaarClassifier_32s_C1RSfs(const Ipp32s* pSrc, int srcStep, const Ipp32s*
pNorm, int normStep, Ipp8u* pMask, int maskStep, IppiSize roiSize, int* pPositive, Ipp32s
threshold, IppiHaarClassifier_32s* pState, int scaleFactor);
```

## Parameters

<i>pSrc</i>	Pointer to the ROI in the source image of integrals.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pNorm</i>	Pointer to the ROI in the source image of norm factors.
<i>normStep</i>	Distance in bytes between starts of consecutive lines in the image of the norm factors.
<i>pMask</i>	Pointer to the source and destination image of classification decisions.
<i>maskStep</i>	Distance in bytes between starts of consecutive lines in the image of classification decisions.
<i>pPositive</i>	Pointer to the number of positive decisions.

<i>roiSize</i>	Size of the source and destination images ROI in pixels.
<i>threshold</i>	Stage threshold value.
<i>pState</i>	Pointer to the Haar classifier structure.
<i>scaleFactor</i>	Scale factor (see <a href="#">Integer Result Scaling</a> ).

## Description

The function `ippiApplyHaarClassifier` is declared in the `ippcv.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function applies the Haar classifier *pState* previously initialized by the function `ippiHaarClassifierInitAlloc` or `ippiTiltedHaarClassifierInitAlloc` to pixels of the source image ROI *pSrc*. The source image should be in the integral representation, it can be obtained by calling one of the [integral functions](#) beforehand. The sum of pixels on feature rectangles is computed as:

$$\sum_{l=1}^k (pSrc[i+y_l, j+x_l] - pSrc[i+Y_l, j+x_l] - pSrc[i+y_l, j+X_l] + pSrc[i+Y_l, j+X_l]) \cdot w_l$$

Here  $(y_l, x_l)$  and  $(Y_l, X_l)$  are coordinates of top left and right bottom pixels of *l*-th rectangle of the feature, and  $w_l$  is the feature weight. For  $i = 0..roiSize.height - 1$ ,  $j = 0..roiSize.width - 1$  all pixels referred in the above formula should be allocated in memory.

The input value of `pPositive[0]` is used as a hint to choose the calculation algorithm. If it is greater than or equal to `roiSize.width*roiSize.height`, the value of the classifier is calculated in accordance with the above formula for all pixels of the input image. Otherwise the value of the classifier is calculated for all non-zero pixels of *pMask* image. If the sum is less than *threshold* than the negative decision is made and the value of the corresponding pixel of the *pMask* image is set to zero. The number of positive decisions is assigned to the `pPositive[0]`.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error if one of the specified pointer is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if one of the image step values is less than <code>roiSize.width*pixelSize</code> .
<code>ippStsNorEvenStepErr</code>	Indicates an error condition if one of the image step values is not divisible by 4 for 32-bit images.

## ApplyMixedHaarClassifier

*Applies a mixed Haar classifier to an image.*

---

### Syntax

```
IppStatus ippiApplyMixedHaarClassifier_32f_C1R(const Ipp32f* pSrc, int srcStep, const
Ipp32f* pTilt, int tiltStep, const Ipp32f* pNorm, int normStep, Ipp8u* pMask, int maskStep,
IppiSize roiSize, int* pPositive, Ipp32f threshold, IppiHaarClassifier_32f* pState);
```

```
IppStatus ippiApplyMixedHaarClassifier_32s32f_C1R(const Ipp32s* pSrc, int srcStep, const
Ipp32s* pTilt, int tiltStep, const Ipp32f* pNorm, int normStep, Ipp8u* pMask, int maskStep,
IppiSize roiSize, int* pPositive, Ipp32f threshold, IppiHaarClassifier_32f* pState);
```

```
IppStatus ippiApplyMixedHaarClassifier_32s_C1RSfs(const Ipp32s* pSrc, int srcStep, const
Ipp32s* pTilt, int tiltStep, const Ipp32s* pNorm, int normStep, Ipp8u* pMask, int maskStep,
IppiSize roiSize, int* pPositive, Ipp32s threshold, IppiHaarClassifier_32s* pState, int
scaleFactor);
```

### Parameters

<i>pSrc</i>	Pointer to the ROI in the source image of integrals.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image of integrals.
<i>pTilt</i>	Pointer to the ROI in the source image of tilted integrals.
<i>tiltStep</i>	Distance in bytes between starts of consecutive lines in the source image of tilted integrals.
<i>pNorm</i>	Pointer to the ROI in the source image of norm factors.
<i>normStep</i>	Distance in bytes between starts of consecutive lines in the image of the norm factors.
<i>pMask</i>	Pointer to the source and destination image of classification decisions.
<i>maskStep</i>	Distance in bytes between starts of consecutive lines in the image of classification decisions.
<i>pPositive</i>	Pointer to the number of positive decisions.
<i>roiSize</i>	Size of the source and destination images ROI in pixels.
<i>threshold</i>	Stage threshold value.
<i>pState</i>	Pointer to the mixed Haar classifier structure.
<i>scaleFactor</i>	Scale factor (see Integer <a href="#">Integer Result Scaling</a> ).

### Description

The function `ippiApplyMixedHaarClassifier` is declared in the `ippcv.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).



This function applies the mixed Haar classifier *pState* to the ROI of the source images *pSrc* and *pTilt*. The mixed Haar classifier is a classifier initialized by the function [ippiHaarClassifierInitAlloc](#) and then modified by the function [ippiTiltHaarFeatures](#). The source images must be in the integral representation, they can be obtained by calling one of the [integral functions](#) beforehand. Common features are applied to the *pSrc* image, and tilted features are applied to the *pTilt* image. The sum of pixels on feature rectangles is computed as:

$$\sum_{l=1}^k (pSrc[i+y_l, j+x_l] - pSrc[i+y_l, j+x_l] - pSrc[i+y_l, j+x_l] + pSrc[i+y_l, j+x_l]) \cdot w_l$$

or

$$\sum_{l=1}^k (pTilt[i+y_l, j+x_l] - pTilt[i+y_l, j+x_l] - pTilt[i+y_l, j+x_l] + pTilt[i+y_l, j+x_l]) \cdot w_l$$

Here  $(y_l, x_l)$  and  $(Y_l, X_l)$  are coordinates of top left and right bottom pixels of *l*-th rectangle of the feature, and  $w_l$  is the feature weight. For  $i = 0..roiSize.height - 1$ ,  $j = 0..roiSize.width - 1$  all pixels referred in the above formula should be allocated in memory.

The input value of *pPositive* [0] is used as a hint to choose the calculation algorithm. If it is greater than or equal to *roiSize.width\*roiSize.height* the value of the classifier is calculated in accordance with the above formula for all pixels of the input image. Otherwise the value of the classifier is calculated for all non-zero pixels of *pMask* image. If the sum is less than *threshold* then the negative decision is made and the value of the corresponding pixel of the *pMask* image is set to zero. The number of positive decisions is assigned to the *pPositive*[0].

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error if one of the specified pointer is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if one of the image step values is less than <i>roiSize.width*pixelSize</i> .
<code>ippStsNorEvenStepErr</code>	Indicates an error condition if one of the image step values is not divisible by 4 for 32-bit images.

# Camera Calibration and 3D Reconstruction

## Correction of Camera Lens Distortion

Digital camera usually introduces significant distortion caused by the camera and lens. These distortions cause errors in any analysis of the image. The functions described in this section correct these distortion using intrinsic camera parameters and distortion coefficients. These intrinsic camera parameters are focal lengths  $f_x$ ,  $f_y$ , and principal point coordinates  $c_x$ ,  $c_y$ . The distortion is characterized by two coefficients of radial distortions  $k_1$ ,  $k_2$  and two coefficients of tangential distortions  $p_1$ ,  $p_2$ .

The undistorted coordinates  $x_u$  and  $y_u$  of point with coordinates  $(x_d, y_d)$  are computed in accordance with the following formulas:

$$x_u = x_d \cdot \left(1 + k_1 r^2 + k_2 r^4\right) + 2p_1 x_d y_d + p_2 \cdot \left(r^2 + 2x_d^2\right)$$

$$y_u = y_d \cdot \left(1 + k_1 r^2 + k_2 r^4\right) + 2p_2 x_d y_d + p_1 \cdot \left(r^2 + 2y_d^2\right)$$

Here  $r^2 = x_d^2 + y_d^2$ ,  $x_d = (j-cx)/fx$ ,  $y_d = (i-cy)/fy$ ;  $i$  and  $j$  are row and columns numbers of the pixel. The pixel value is computed using bilinear interpolation of four nearest pixel of the source image. If undistorted coordinates are outside the image, then the destination pixel is not changed.

## UndistortGetSize

*Computes the size of the external buffer.*

---

### Syntax

```
IppStatus ippUndistortGetSize(IppiSize roiSize, int* pBufferSize);
```

### Parameters

<i>roiSize</i>	Size of source and destination images ROI in pixels.
<i>pBufferSize</i>	Pointer to the computed value of the buffer size.

### Description

The function `ippUndistortGetSize` is declared in the `ippcv.h` file. This function computes the size of the temporary external buffer that is used by the functions `ippiUndistortRadial`. The buffer of the computed size can be used to process smaller images as well.

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error if <i>pBufferSize</i> is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.

## UndistortRadial

*Corrects radial distortions of the single image.*

### Syntax

```
IppStatus ippiUndistortRadial_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, Ipp32f fx, Ipp32f fy, Ipp32f cx, Ipp32f cy, Ipp32f k1, Ipp32f k2, Ipp8u* pBuffer);
```

Supported values for mod:

8u_C1R	16u_C1R	32f_C1R
8u_C3R	16u_C3R	32f_C3R

### Parameters

<i>pSrc</i>	Pointer to the ROI in the source distorted image.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the ROI in the destination corrected image.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>roiSize</i>	Size of source and destination images ROI in pixels.
<i>fx</i>	Focal lengths along the x axis.
<i>fy</i>	Focal lengths along the y axis.
<i>cx</i>	x-coordinate of the principal point.
<i>cy</i>	y-coordinate of the principal point.
<i>k1</i>	First coefficient of radial distortion.
<i>k2</i>	Second coefficient of radial distortion.
<i>pBuffer</i>	Pointer to the external buffer.

### Description

The function `ippiUndistortRadial` is declared in the `ippcv.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function corrects radial distortions of the single source image *pSrc* and stores corrected image in the *pDst*. Correction is performed accounting camera parameters *fx*, *fy*, *cx*, *cy* and radial distortion parameters *k1*, *k2*. The function can also pass the pointer to the external buffer *pBuffer* whose size should be computed previously using the function `ippiUndistortGetSize`. If a null pointer is passed, slower computations without an external buffer will be performed.

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error if <i>pSrc</i> or <i>pDst</i> is NULL.

<code>ippStsSizeErr</code>	Indicates an error condition if <code>roiSize</code> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <code>srcStep</code> is less than <code>roiSize.width * &lt;pixelSize&gt;</code> , or <code>dstStep</code> is less than <code>roiSize.width * &lt;pixelSize&gt;</code> .
<code>ippStsNotEvenStepErr</code>	Indicates an error condition if one of the step values is not divisible by 4 for floating-point images, or by 2 for short-integer images.
<code>ippStsBadArgErr</code>	Indicates an error if <code>fx</code> or <code>fy</code> is equal to 0.

## CreateMapCameraUndistort

Creates look-up tables of coordinates of corrected image.

### Syntax

```
IppStatus ippiCreateMapCameraUndistort_32f_C1R(Ipp32f* pxMap, int xStep, Ipp32f* pyMap,
int yStep, IppiSize roiSize, Ipp32f fx, Ipp32f fy, Ipp32f cx, Ipp32f cy, Ipp32f k1, Ipp32f
k2, Ipp32f p1, Ipp32f p2, Ipp8u* pBuffer);
```

### Parameters

<code>pxMap</code>	Pointer to the destination x coordinate look-up buffer.
<code>xStep</code>	Distance in bytes between starts of consecutive lines in the <code>pxMap</code> image.
<code>pyMap</code>	Pointer to the destination y coordinate look-up buffer.
<code>yStep</code>	Distance in bytes between starts of consecutive lines in the <code>pyMap</code> image.
<code>roiSize</code>	Size of source and destination images ROI in pixels.
<code>fx</code>	Focal lengths along the <i>x</i> axis.
<code>fy</code>	Focal lengths along the <i>y</i> axis.
<code>cx</code>	<i>x</i> -coordinate of the principal point.
<code>cy</code>	<i>y</i> -coordinate of the principal point.
<code>k1</code>	First coefficient of radial distortion.
<code>k2</code>	Second coefficient of radial distortion.
<code>p1</code>	First coefficient of tangential distortion.
<code>p2</code>	Second coefficient of tangential distortion.
<code>pBuffer</code>	Pointer to the external buffer.

### Description

The function `ippiCreateMapCameraUndistort` is declared in the `ippcv.h` file. It operates with ROI (see Regions of Interest in Intel IPP ).

This function creates the look-up tables of  $x$ - and  $y$ -coordinates *pxMap* and *pyMap* respectively. These coordinates are computed in accordance with camera parameters  $f_x$ ,  $f_y$ ,  $c_x$ ,  $c_y$ , and distortion parameters  $k_1$ ,  $k_2$ ,  $p_1$ ,  $p_2$ . The created tables can be used by the Intel IPP function [ippiRemap](#) to remap the distorted source image and get the corrected image.

To accelerate the computations the function can pass the pointer to the external buffer *pBuffer* whose size should be computed previously using the function [ippiUndistortGetSize](#). If a null pointer is passed, slower computations without an external buffer will be performed.

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error if <i>pxMap</i> or <i>pyMap</i> is <i>NULL</i> .
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if : <i>xStep</i> is less than <i>roiSize.width</i> * <i>&lt;pixelSize&gt;</i> , or <i>yStep</i> is less than <i>roiSize.width</i> * <i>&lt;pixelSize&gt;</i> .
<code>ippStsNotEvenStepErr</code>	Indicates an error condition if one of the step values is not divisible by 4 for floating-point images.
<code>ippStsBadArgErr</code>	Indicates an error when $f_x$ or $f_y$ is equal to 0.

### Example

The code example below demonstrates how to correct camera lens distortion for set of images using Intel IPP functions.

```

Ipp32f *pxMap, *pyMap, *pBuffer;
Int xStep, yStep, buflen;
IppiSize roiSize;
IppiRect rect;
ippiUndistortGetSize (roiSize, &buflen);
buffer = ippiMalloc_8u(buflen);
xMap = ippiMalloc_32f(roiSize.x, roiSize.y, *xStep);
yMap = ippiMalloc_32f(roiSize.x, roiSize.y, *yStep);
ippiCreateMapCameraUndistort_32f_C1R
(pxMap, pxStep, yMap, yStep, roiSize,
    fx, fy, cx, cy, k1, k2, 0, 0, pBuffer);

rect.x = 0;
    rect.y = 0;

rect.width = roiSize.width;
rect.height = roiSize.height;

...

ippiRemap_32f(pSrc, roiSize, srcStep, rect, pxMap, xStep, pyMap, yStep,
```

```

        pDst, dstStep, roiSize, IPPI_INTER_LINEAR);

...

ippiFree(yMap);
ippiFree(xMap);
ippsFree(buffer);

```

## Image Enhancement Functions

This section describes the functions that are used in the high-quality upsampling, or increasing the spatial resolution of the video stream (Haussecker-Nestares super-resolution algorithm).

### Algorithm Overview

For every frame in the input stream the algorithm computes the corresponding high-resolution (upsampled) frame by combining information from the original low-resolution frame (the central frame) and several low-resolution frames preceding and following the central frame in the video stream. The algorithm consists of 2 parts:

- Registration, or finding position in the central frame of each pixel of each low-resolution frame. That is, the dense optical flow from each surrounding frames to the central frame is calculated.
- Robust estimation of value of each pixel in the high-resolution frame using the Bayesian framework.

The Intel IPP functions described below implement only second part.

Once the optical flow is computed, the Bayesian formulation is reduced to the following optimization problem:

$$g = \underset{g}{\operatorname{argmin}} C(g);$$

$$C(g) = \sum_{i,j} w_{i,j} \varphi_L\left(\frac{f_{i,j} - A_{i,j}g}{\sigma_L}\right) + \lambda \sum_{(i,j): i < j \ \& \ |x(i) - x(j)| + |y(i) - y(j)| = 1} \varphi_P\left(\frac{g_i - g_j}{\sigma_P}\right)$$

where:

$g$  - unknown high-resolution frame represented as a 1D vector which elements are pixel intensities in the raster scan order.

$w_{i,j}$  - confidence weights that make certain links between low- and high-resolution frames more or less important depending on the accuracy of the estimated motion field at corresponding areas and so on.

$f_i$  -  $i$ -th low-resolution frame, represented as a 1D vector.

$f_{i,j}$  - intensity of the  $j$ -th pixel of  $f_i$ .

$A_i$  - mapping from  $g$  to  $f_i$ , also known as the point spread function PSF (see "Point Spread Function" for more details). It combines optical flow, blurring and downsampling. As  $A_i$  is a linear mapping (each low-resolution pixel is a linear combination of several pixels in high-resolution image,  $A_{i,j}g$  is a dot product of the  $j$ -th row of matrix  $A_i$  and vector  $g$ ).

$\sigma_L, \sigma_P$  - standard deviations in the likelihood and prior parts of the target function, respectively.

$\phi_L, \phi_P$  - robust error functions (see "Error Functions") for the likelihood and prior parts of the target function, respectively.

$\lambda$  - parameter that regulates smoothness, or the relative weight of the prior part.

$(i, j): i < j \ \& \ |x(i) - x(j)| + |y(i) - y(j)| = 1$  means that the summation is done over all ordered pairs  $(g_i, g_j)$  where  $g_i$  and  $g_j$  are intensities of the horizontally or vertically adjacent pixels of  $g$ .

This problem is solved using the conjugate gradient algorithm, which main steps are:

- Computing gradient of  $C(g)$ :

$$\nabla C(g) = \sum_{i,j} \frac{w_{i,j}}{\sigma_L} A_{i,j}^T \psi_L \left( \frac{f_{i,j} - A_{i,j}g}{\sigma_L} \right) + \lambda \sum_{(i,j): |x(i)-x(j)|+|y(i)-y(j)|=1} \frac{I_i}{\sigma_P} \psi_P \left( \frac{g_i - g_j}{\sigma_P} \right)$$

where  $\psi_L$  and  $\psi_P$  are derivatives of the  $\sigma_L$  and  $\sigma_P$ , respectively, and  $I_i$  is the  $i$ -th column of the identity matrix.

- Solving 1D optimization problem:

$$\alpha = \underset{\alpha}{\operatorname{argmin}} C(g + \alpha d)$$

where  $d$  is the current "conjugate" direction. The search is performed using the Newton-Raphson method, if it fails then the golden-section search (GSS) is used.

- Computing cost  $C(g)$  to determine when to stop the algorithm (as soon as the decrease of the cost function becomes small enough or even negative).

## Point Spread Function

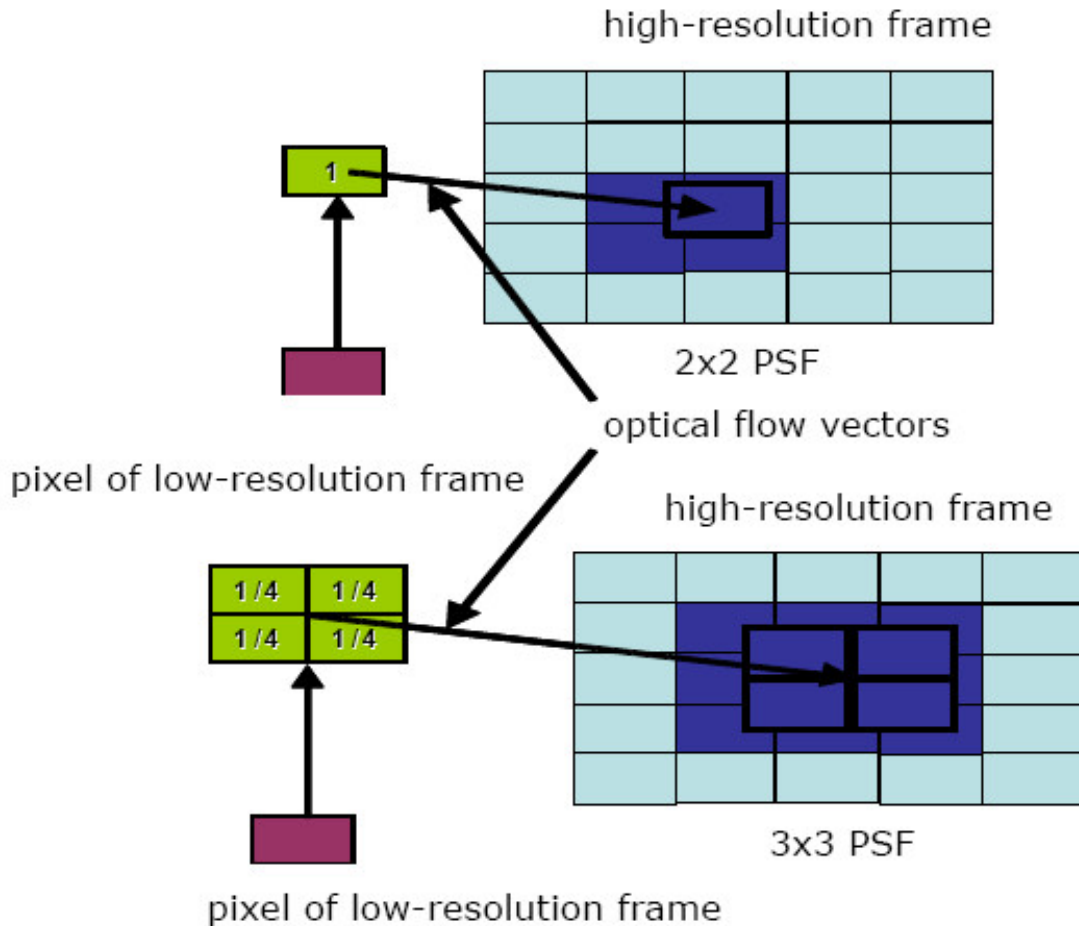
The point spread function (PSF) maps  $f_{i,j}$  to  $g$  or vice versa. To do this, blurring, decimation and motion need to be modeled at high resolution.

In case of superresolution with scale factor 2 a simple 2x2 box filter is used for blurring and downsampling. For motion field more complex interpolation is used. Nearest neighbor interpolation results in the 3x3 PSF, in other word each low-resolution image pixel is mapped to 3x3 area in high-resolution image.

Another useful mode of this algorithm is x1 upsampling, it may be used for quick de-noising and de-blocking of chrominance planes. In this case motion field is not interpolated, and blurring is not performed, thus it can be done with the 2x2 PSF.

Figure “Mapping Low-resolution frame with the PSF” shows the above described types of PSF.

### Mapping Low-resolution frame with the PSF



### Error Functions

Error function must be a symmetrical non-negative function that has second derivative. Moreover it must be convex near the zero. A robust error function is an error function that grows slower than the quadratic function, thus outliers do not influence the target cost function much. There are many functions that satisfy the above requirements. Intel IPP provides the Cauchy function (see “Intel IPP Reference Manual” vol.1):

$$\varphi(x) = \frac{1}{2} \log \left( 1 + \left( \frac{x}{C} \right)^2 \right)$$

where  $C$  is a user-supplied parameter.



## SRHNInit Alloc\_PSF3x3, SRHNInitAlloc\_PSF2x2

*Allocates memory and initializes the internal representation of the PSF table.*

---

### Syntax

```
IppStatus ippISRHNInitAlloc_PSF3x3(IppiSRHNSpec_PSF3x3** ppPSF, const Ipp32f coeffTab[][9],
int tabSize);

IppStatus ippISRHNInitAlloc_PSF2x2(IppiSRHNSpec_PSF2x2** ppPSF, const Ipp32f coeffTab[][4],
int tabSize);
```

### Parameters

<i>ppPSF</i>	Double pointer to the PSF table.
<i>coeffTab</i>	Array containing the input PSF table with the coefficients.
<i>tabSize</i>	The number of elements in the <i>coeffTab</i> .

### Description

The functions `ippISRHNInitAlloc_PSF3x3` and `ippISRHNInitAlloc_PSF2x2` are declared in the `ippcv.h` file. These functions allocate memory and initialize the internal table representation of the 3x3 or 2x2 point spread functions (PSF) respectively.

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error if the <i>ppPSF</i> pointer is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error if the <i>tabSize</i> is less than or equal to 0.

## SRHNFree\_PSF3x3, SRHNFree\_PSF2x2

*Deallocates the internal representation of the PSF table.*

---

### Syntax

```
IppStatus ippSRHNFree_PSF3x3(IppiSRHNSpec_PSF3x3* pPSF);

IppStatus ippSRHNFree_PSF2x2(IppiSRHNSpec_PSF2x2* pPSF);
```

### Parameters

<i>pPSF</i>	Pointer to the PSF table.
-------------	---------------------------

### Description

The functions `ippSRHNFree_PSF3x3` and `ippSRHNFree_PSF2x2` are declared in the `ippcv.h` file. These functions deallocates the internal table representation of the 3x3 or 2x2 point spread functions (PSF) respectively.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error if the <i>pPSF</i> pointer is <i>NULL</i> .

## SRHNCalcResidual\_PSF3x3, SRHNCalcResidual\_PSF2x2

*Computes residuals for the likelihood part of the target function or its gradient.*

---

## Syntax

```
IppStatus ippISRHNCalcResidual_PSF3x3_8u32f_C1R(const Ipp32f* pHighRes, int highResStep,
const Ipp8u* pLowRes, const Ipp32s* pOffset, const Ipp16u* pCoeff, Ipp32f* pRes, int len,
const IppiSRHNSpec_PSF3x3* pPSF);
```

```
IppStatus ippISRHNCalcResidual_PSF3x3_16u32f_C1R(const Ipp32f* pHighRes, int highResStep,
const Ipp16u* pLowRes, const Ipp32s* pOffset, const Ipp16u* pCoeff, Ipp32f* pRes, int len,
const IppiSRHNSpec_PSF3x3* pPSF);
```

```
IppStatus ippISRHNCalcResidual_PSF2x2_16u32f_C1R(const Ipp32f* pHighRes, int highResStep,
const Ipp16u* pLowRes, const Ipp32s* pOffset, const Ipp16u* pCoeff, Ipp32f* pRes, int len,
const IppiSRHNSpec_PSF2x2* pPSF);
```

```
IppStatus ippISRHNCalcResidual_PSF2x2_8u32f_C1R(const Ipp32f* pHighRes, int highResStep,
const Ipp8u* pLowRes, const Ipp32s* pOffset, const Ipp16u* pCoeff, Ipp32f* pRes, int len,
const IppiSRHNSpec_PSF2x2* pPSF);
```

## Parameters

<i>pHighRes</i>	Pointer to high-resolution image.
<i>highResStep</i>	Distance in bytes between starts of consecutive lines in the <i>pHighRes</i> image.
<i>pLowRes</i>	Pointer to the array of pixel values from low-resolution images.
<i>pOffset</i>	Pointer to the array of offsets of 3x3 or 2x2 areas in the high-resolution image.
<i>pCoeff</i>	Pointer to the array of indexes of the coefficients in the input PSF table.
<i>pRes</i>	Pointer to the output array containing the calculated residuals.
<i>len</i>	Number of elements in the arrays <i>pLowRes</i> , <i>pOffset</i> , <i>pCoeff</i> and <i>pRes</i> .
<i>pPSF</i>	Pointer to the initialized table for 3x3 or 2x2 PSF.

## Description

The functions `ippISRHNCalcResidual_PSF3x3` and `ippISRHNCalcResidual_PSF2x2` are declared in the `ippcv.h` file.

These functions compute residuals  $pRes$  for the likelihood part of the target function  $C(g)$  or its gradient  $\nabla C(g)$ . The calculated values are used as the arguments of the robust error functions or their derivatives. The following pseudocode shows how the calculation is performed.

```
for( int i = 0; i < len; i++ ) {
    const Ipp32f* ptr = hiRes + ofs[i];
    const Ipp32f* c = psf->cTab[coeff[i]];
    Ipp32f sum = 0;
    for( int dy = 0; dy < K; dy++ )
        for( int dx = 0; dx < K; dx++ )
            sum += c[dy*K+dx]*ptr[dy*hiResStep + dx];
    residual[i] = sum - lowRes[i];
}
```

Here  $K=3$  for the 3x3 PSF, and  $K=2$  for the 2x2 PSF.

The results stored in the  $pRes$  then can be passed to the robust error functions: `ippsCauchy`, `ippsCauchyD`, and `ippsCauchyDD2` (see section "Arithmetic Functions" in the "Intel IPP Reference Manual" vol.1).

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error if one of the specified pointer is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error if <code>len</code> is less than 0.
<code>ippStsStepErr</code>	Indicates an error if <code>highResStep</code> is less than $2 * \text{sizeof}(\text{Ipp32f})$ for 3x3 PSF, and less than $3 * \text{sizeof}(\text{Ipp32f})$ for the 2x2 PSF.

## SRHNUpdateGradient\_PSF3x3, SRHNUpdateGradient\_PSF2x2

*Updates the likelihood part of the gradient.*

### Syntax

```
IppStatus ippiSRHNUpdateGradient_PSF3x3_32f_C1R(Ipp32f* pGrad, int gradStep, const Ipp32s*
pOffset, const Ipp16u* pCoeff, const Ipp32f* pDerivRes, const Ipp8u* pWeight, int len,
const Ipp32f* pWeightTab, const IppiSRHNSpec_PSF3x3* pPSF);
```

```
IppStatus ippiSRHNUpdateGradient_PSF2x2_32f_C1R(Ipp32f* pGrad, int gradStep, const Ipp32s*
pOffset, const Ipp16u* pCoeff, const Ipp32f* pDerivRes, const Ipp8u* pWeight, int len,
const Ipp32f* pWeightTab, const IppiSRHNSpec_PSF2x2* pPSF);
```

### Parameters

<code>pGrad</code>	Pointer to the gradient.
<code>gradStep</code>	Distance in bytes between starts of consecutive lines in the <code>pGrad</code> .
<code>pOffset</code>	Pointer to the array of offsets of 3x3 or 2x2 areas in the high-resolution image.
<code>pCoeff</code>	Pointer to the array of indexes of the coefficients in the input PSF table.
<code>pDerivRes</code>	Pointer to the array containing the calculated derivatives of the robust error function.

<i>pWeight</i>	Pointer to the array of indexes of the values in the <i>pWeightTab</i> .
<i>len</i>	Number of elements in the arrays <i>pOffset</i> , <i>pCoeff</i> , <i>pDerivRes</i> , and <i>pWeight</i> .
<i>pWeightTab</i>	Pointer to the table of the confidence weights.
<i>pPSF</i>	Pointer to the initialized table for 3x3 or 2x2 PSF.

## Description

The functions `ippiSRHNUpdateGradient_PSF3x3` and `ippiSRHNUpdateGradient_PSF2x2` are declared in the `ippcv.h` file. They operate with ROI (see [Regions of Interest in Intel IPP](#)).

These functions update gradient of the target function  $\nabla C(g)$  using the calculated derivatives of the robust error function. The following pseudocode shows how the calculation is performed.

```
for( int i = 0; i < len; i++ ) {
    const Ipp32f* ptr = grad + ofs[i];
    const Ipp32f* c = psf->cTab[coeff[i]];
    Ipp32f w = wTab[weight[i]]*dRes[i];
    for( int dy = 0; dy < K; dy++ )
        for( int dx = 0; dx < K; dx++ )
            ptr[dy*gradStep+dx] += c[dy*K+dx]*w;
}
```

Here  $K=3$  for the 3x3 PSF, and  $K=2$  for the 2x2 PSF.



**CAUTION.** When the gradient is calculated, the functions `ippiSRHNCalcResidual` and `ippiSRHNUpdateGradient` are called sequentially (and the function `ippsCauchyD` between them, see section "Arithmetic Functions" in the "Intel IPP Reference Manual" vol.1). The same array *pOffset* can be passed to both these functions (with the same PSF type) but only when the parameters *highResStep* and *gradStep* are equal.

---

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error if one of the specified pointer is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error if <i>len</i> is less than 0.
<code>ippStsStepErr</code>	Indicates an error if <i>highResStep</i> is less than $2 \times \text{sizeof}(\text{Ipp32f})$ for 3x3 PSF, and less than $3 \times \text{sizeof}(\text{Ipp32f})$ for the 2x2 PSF.

# *Image Compression Functions*

---

This chapter describes the Intel® IPP image processing functions that prepare data and perform still image compression and coding in accordance with JPEG, JPEG2000, and JPEG XR standards (see [ISO10918], [ISO15444], and [ISO29199]). These functions are grouped into the following sections:

## Support Functions

### **JPEG Coding:**

Color Conversion Functions

Combined Color Conversion Functions

Quantization Functions

Combined DCT Functions

Level Shift Functions

Sampling Functions

Planar-to-Pixel and Pixel-to-Planar Conversion Functions

Huffman Codec Functions

### **Lossless Mode of Operation:**

Functions for Lossless JPEG Coding

### **JPEG2000 Coding:**

Wavelet Transform Functions

JPEG2000 Entropy Coding and Decoding Functions

Component Transform Functions

### **RLE Coding:**

Functions for RLE Coding

### **Texture Image Coding:**

Texture Compression Functions

### **High Definition Photo Coding:**

Photo Core Transform Functions

### **JPEG XR Coding:**

Core Transform Functions

VLC and FLC Functions

Each section starts with a table that lists functions described in more detail later in this section, together with brief description of operations these functions perform.



**NOTE.** For simplicity, `ippi` prefix is omitted in function group names given in the summary tables. Function prototypes and references in the text contain full names.

The use of the Intel IPP JPEG functions is demonstrated in Intel IPP Samples. See *Intel IPP JPEG Processing Samples* downloadable from <http://www.intel.com/cd/software/products/asmo-na/eng/220046.htm>.

## Support Functions

This section describes the support function that returns information about the current version of the used Intel IPP JPEG codec software.

### ippjGetLibVersion

*Returns information of the library version.*

#### Syntax

```
const IppLibraryVersion* ippjGetLibVersion(void);
```

#### Description

The function `ippjGetLibVersion` is declared in the `ippj.h` file. This function returns a pointer to a static data structure `IppLibraryVersion` that contains information about the current version of the Intel IPP JPEG codec library. There is no need to free memory referenced by this pointer, as it points to a static variable. The `IppLibraryVersion` structure contains the following fields:

<i>major</i>	The major number of the current library version.
<i>minor</i>	The minor number of the current library version.
<i>Name</i>	The name of the library.
<i>Version</i>	The library version ASCII string.
<i>BuildDate</i>	The library version build date.

#### Return Values

The return value of the function is a pointer to the structure `IppLibraryVersion`.

## Color Conversion Functions

This section describes Intel IPP color conversion functions that are specific for JPEG codec.

The table below provides the list if these functions.

### Color Conversion Functions

Function Base Name	Description
<a href="#">RGBToY_JPEG</a>	Converts RGB images to gray scale.

Function Base Name	Description
<a href="#">BGRToY_JPEG</a>	Converts BGR images to gray scale.
<a href="#">RGBToYCbCr_JPEG</a>	Converts RGB images to the YCbCr color model.
<a href="#">YCbCrToRGB_JPEG</a>	Converts YCbCr images to the RGB color model.
<a href="#">RGB565ToYCbCr_JPEG, RGB555ToYCbCr_JPEG</a>	Convert 16-bit RGB images to the YCbCr color model.
<a href="#">YCbCrToRGB565_JPEG, YCbCrToRGB555_JPEG</a>	Convert YCbCr images to the 16-bit RGB images.
<a href="#">BGRToYCbCr_JPEG</a>	Converts BGR images to the YCbCr color model.
<a href="#">YCbCrToBGR_JPEG</a>	Converts YCbCr images to the BGR color model.
<a href="#">BGR565ToYCbCr_JPEG, BGR555ToYCbCr_JPEG</a>	Convert 16-bit BGR images to the YCbCr color model.
<a href="#">YCbCrToBGR565_JPEG, YCbCrToBGR555_JPEG</a>	Convert YCbCr images to the 16-bit BGR images.
<a href="#">RGBToYCbCr422_JPEG</a>	Converts RGB images to the 4:2:2 YCbCr images.
<a href="#">YCbCr422ToRGB_JPEG</a>	Converts 4:2:2 YCbCr images to the RGB images.
<a href="#">BGRToYCbCr422_JPEG</a>	Converts BGR images to the 4:2:2 YCbCr images.
<a href="#">YCbCr422ToBGR_JPEG</a>	Converts 4:2:2 YCbCr images to the BGR images.
<a href="#">RGBToYCbCr111_JPEG</a>	Converts RGB images to the 4:1:1 YCbCr images.
<a href="#">YCbCr111To RGB_JPEG</a>	Converts 4:1:1 YCbCr images to the RGB images.
<a href="#">BGRToYCbCr111_JPEG</a>	Converts BGR images to the 4:1:1 YCbCr images.
<a href="#">YCbCr111To BGR_JPEG</a>	Converts 4:1:1 YCbCr image to the BGR image.
<a href="#">CMYKToYCCK_JPEG</a>	Converts CMYK images to the YCCK color model.
<a href="#">YCCKToCMYK_JPEG</a>	Converts YCCK images to the CMYK color model.

## RGBToY\_JPEG

*Converts an RGB image to gray scale.*

### Syntax

#### Case 1: Operation on planar data

```
IppStatus ippiRGBToY_JPEG_8u_P3C1R(const Ipp8u* pSrcRGB[3], int srcStep, Ipp8u* pDstY,
int dstStep, IppiSize roiSize);
```

#### Case 2: Operation on pixel-order data

```
IppStatus ippiRGBToY_JPEG_8u_C3C1R(const Ipp8u* pSrcRGB, int srcStep, Ipp8u* pDstY, int
dstStep, IppiSize roiSize);
```

## Parameters

<i>pSrcRGB</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDstY</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>roiSize</i>	Size of the source and destination ROI in pixels.

## Description

The function `ippiRGBToY_JPEG` is declared in the `ippj.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts the RGB image to gray scale using the following formula:

$$Y = 0.299 * R + 0.587 * G + 0.114 * B$$

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if any of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> or <i>dstStep</i> has a zero or negative value.



## Example

The code example below shows how to use the function `ippiRGBToY_JPEG_8u_C3C1R`.

```

Ipp8u srcRGB[9*3] = { 255, 0, 0, 255, 0, 0, 255, 0, 0,
                      0, 255, 0, 0, 255, 0, 0, 255, 0,
                      0, 0, 255, 0, 0, 255, 0, 0, 255};

Ipp8u dstY[3*3];
IppiSize roiSize = { 9, 3 };
ippiRGBToY_JPEG_8u_C3C1R ( srcRGB, 9, dstY, 9, roiSize );
result:
255 0   0   255 0   0   255 0   0
0   255 0   0   255 0   0   255 0       src
0   0   255 0   0   255 0   0   255

76  76  76
150 150 150      dst
29   29  29

```

## BGRToY\_JPEG

*Converts a BGR image to gray scale.*

### Syntax

```

IppStatus ippiBGRToY_JPEG_8u_C3C1R(const Ipp8u* pSrcBGR, int srcStep, Ipp8u* pDstY, int
dstStep, IppiSize roiSize);

```

### Parameters

<i>pSrcBGR</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDstY</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>roiSize</i>	Size of the source and destination ROI in pixels.

### Description

The function `ippiBGRToY_JPEG` is declared in the `ippj.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts a BGR image to gray scale using the following formula:

$$Y = 0.299 * R + 0.587 * G + 0.114 * B$$

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if any of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <code>roiSize</code> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <code>srcStep</code> or <code>dstStep</code> has a zero or negative value.

## RGBToYCbCr\_JPEG

*Converts an RGB image to YCbCr color model.*

---

### Syntax

#### Case 1: Operation on planar data

```
ippStatus ippRGBToYCbCr_JPEG_8u_P3R(const Ipp8u* pSrc[3], int srcStep, Ipp8u* pDst[3],
int dstStep, IppiSize roiSize);
```

#### Case 2: Operation on pixel-order data

```
ippStatus ippRGBToYCbCr_JPEG_8u_C3P3R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst[3],
int dstStep, IppiSize roiSize);
```

```
ippStatus ippRGBToYCbCr_JPEG_8u_C4P3R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst[3],
int dstStep, IppiSize roiSize);
```

### Parameters

<code>pSrc</code>	Pointer to the source image ROI for pixel-order data. Array of pointers to the source ROI in each plane for planar image.
<code>srcStep</code>	Distance in bytes between starts of consecutive lines in the source image.
<code>pDst</code>	Array of pointers to the ROI in each plane of the destination image.
<code>dstStep</code>	Distance in bytes between starts of consecutive lines in the destination image.
<code>roiSize</code>	Size of the source and destination ROI in pixels.

### Description

The function `ippRGBToYCbCr_JPEG` is declared in the `ippj.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts an RGB image to the YCbCr color model using the following formulas:

$$Y = 0.299 * R + 0.587 * G + 0.114 * B$$

$$Cb = -0.16874 * R - 0.33126 * G + 0.5 * B + 128$$

$$Cr = 0.5 * R - 0.41869 * G - 0.08131 * B + 128$$

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if any of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <code>roiSize</code> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <code>srcStep</code> or <code>dstStep</code> has a zero or negative value.

## YCbCrToRGB\_JPEG

*Converts an YCbCr image to the RGB color model.*

### Syntax

```
ippStatus ippYCbCrToRGB_JPEG_8u_P3R(const Ipp8u* pSrc[3], int srcStep, Ipp8u* pDst[3],
int dstStep, IppiSize roiSize);

ippStatus ippYCbCrToRGB_JPEG_8u_P3C3R(const Ipp8u* pSrc[3], int srcStep, Ipp8u* pDst,
int dstStep, IppiSize roiSize);

ippStatus ippYCbCrToRGB_JPEG_8u_P3C4R(const Ipp8u* pSrc[3], int srcStep, Ipp8u* pDst,
int dstStep, IppiSize roiSize, Ipp8u aval);
```

### Parameters

<code>pSrc</code>	Array of pointers to ROI in each plane of the source image.
<code>srcStep</code>	Distance in bytes between starts of consecutive lines in planes of the source image.
<code>pDst</code>	Array of pointers to ROI in each plane of the planar destination image. Pointer to the destination image ROI for pixel-order image.
<code>dstStep</code>	Distance in bytes between starts of consecutive lines in the destination image.
<code>roiSize</code>	Size of the source and destination ROI in pixels.
<code>aval</code>	Constant value to create the fourth channel.

### Description

The function `ippYCbCrToRGB_JPEG` is declared in the `ippj.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts an YCbCr image to the RGB image according to the following formulas:

$$R = Y + 1.402 * Cr - 179.456$$

$$G = Y - 0.34414 * Cb - 0.71414 * Cr + 135.45984$$

$$B = Y + 1.772 * Cb - 226.816$$

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
--------------------------	---------------------

<code>ippStsNullPtrErr</code>	Indicates an error condition if any of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <code>roiSize</code> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <code>srcStep</code> or <code>dstStep</code> has a zero or negative value.

## RGB565ToYCbCr\_JPEG, RGB555ToYCbCr\_JPEG

*Convert a 16-bit RGB image to YCbCr color model.*

---

### Syntax

```
ippStatus ippRGB565ToYCbCr_JPEG_16u8u_C3P3R(const Ipp16u* pSrc, int srcStep, Ipp8u*
pDst[3], int dstStep, IppiSize roiSize);
```

```
ippStatus ippRGB555ToYCbCr_JPEG_16u8u_C3P3R(const Ipp16u* pSrc, int srcStep, Ipp8u*
pDst[3], int dstStep, IppiSize roiSize);
```

### Parameters

<code>pSrc</code>	Pointer to the source image ROI.
<code>srcStep</code>	Distance in bytes between starts of consecutive lines in the source image.
<code>pDst</code>	An array of pointers to the ROIs in the separate destination color planes.
<code>dstStep</code>	Distance in bytes between starts of consecutive lines in the destination image.
<code>roiSize</code>	Size of the source and destination ROI in pixels.

### Description

The functions `ippRGB555ToYCbCr_JPEG` and `ippRGB565ToYCbCr_JPEG` are declared in the `ippj.h` file. These functions convert an RGB image to the YCbCr image using the same formulas as the `ippiRGBToYCbCr_JPEG` function for computing Y, Cb, and Cr component values. The source image `pSrc` has a reduced bit depth of 16 bits per pixel (see [Figure "16-bit pixel formats"](#)), and it can be in one of two possible formats : RGB565 (5 bits for red, 6 bits for green, 5 bits for blue), or RGB555 (5 bits for red, green, blue).

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if any of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <code>roiSize</code> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <code>srcStep</code> or <code>dstStep</code> has a zero or negative value.

## YCbCrToRGB565\_JPEG, YCbCrToRGB555\_JPEG

*Convert an YCbCr image to the RGB color model.*

### Syntax

```
IppStatus ippiYCbCrToRGB565_JPEG_8u16u_P3C3R(const Ipp8u* pSrc[3], int srcStep, Ipp16u* pDst, int dstStep, IppiSize roiSize);
```

```
IppStatus ippiYCbCrToRGB555_JPEG_8u16u_P3C3R(const Ipp8u* pSrc[3], int srcStep, Ipp16u* pDst, int dstStep, IppiSize roiSize);
```

### Parameters

<i>pSrc</i>	An array of pointers to the ROIs in the separate source color planes.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>roiSize</i>	Size of the source and destination ROI in pixels.

### Description

The functions `ippiYCbCrToRGB565_JPEG` and `ippiYCbCrToRGB555_JPEG` are declared in the `ippj.h` file. They operate with ROI (see [Regions of Interest in Intel IPP](#)).

These functions convert an YCbCr image to the RGB format using the same formulas as the `ippiYCbCrToRGB_JPEG` function for computing *R*, *G*, and *B* component values. The destination image *pDstRGB* has a reduced bit depth of 16 bits per pixel (see [Figure "16-bit pixel formats"](#)), and it can be in one of two possible formats : RGB565 (5 bits for red, 6 bits for green, 5 bits for blue), or RGB555 (5 bits for red, green, blue).

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if any of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> or <i>dstStep</i> has a zero or negative value.

## BGRToYCbCr\_JPEG

*Converts a BGR image to YCbCr color model.*

### Syntax

```
IppStatus ippiBGRToYCbCr_JPEG_8u_C3P3R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst[3], int dstStep, IppiSize roiSize);
```

```
IppStatus ippiBGRToYCbCr_JPEG_8u_C4P3R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst[3],
int dstStep, IppiSize roiSize);
```

### Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Array of pointers to the ROI in each planes of the destination image.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>roiSize</i>	Size of the source and destination ROI in pixels.

### Description

The function `ippiBGRToYCbCr_JPEG` is declared in the `ippj.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts a BGR image to the YCbCr color model using the same formulas as the `ippiRGBToYCbCr_JPEG` function for computing Y, Cb and Cr component values.

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if any of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> or <i>dstStep</i> has a zero or negative value.

## YCbCrToBGR\_JPEG

Converts an YCbCr image to the BGR color model.

### Syntax

```
IppStatus ippiYCbCrToBGR_JPEG_8u_P3C3R(const Ipp8u* pSrc[3], int srcStep, Ipp8u* pDst,
int dstStep, IppiSize roiSize);
```

```
IppStatus ippiYCbCrToBGR_JPEG_8u_P3C4R(const Ipp8u* pSrc[3], int srcStep, Ipp8u* pDst,
int dstStep, IppiSize roiSize);
```

### Parameters

<i>pSrc</i>	Array of pointers to the ROI in each plane of the source image.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in planes of the source image.
<i>pDst</i>	Pointer to the destination image ROI.

<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>roiSize</i>	Size of the source and destination ROI in pixels.
<i>aval</i>	Constant value to create the fourth channel.

## Description

The function `ippiYCbCrToBGR_JPEG` is declared in the `ippj.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts an YCbCr image to the BGR image using the same formulas as the `ippiYCbCrToRGB_JPEG` function for computing B, G, and R component values.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if any of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> or <i>dstStep</i> has a zero or negative value.

## BGR565ToYCbCr\_JPEG, BGR555ToYCbCr\_JPEG

Convert a BGR image to YCbCr color model.

## Syntax

```
ippStatus ippiBGR565ToYCbCr_JPEG_16u8u_C3P3R(const Ipp16u* pSrc, int srcStep, Ipp8u* pDst[3], int dstStep, IppiSize roiSize);
```

```
ippStatus ippiBGR555ToYCbCr_JPEG_16u8u_C3P3R(const Ipp16u* pSrc, int srcStep, Ipp8u* pDst[3], int dstStep, IppiSize roiSize);
```

## Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	An array of pointers to the ROIs in the separate destination color planes.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>roiSize</i>	Size of the source and destination ROI in pixels.

## Description

The functions `ippiBGR555ToYCbCr_JPEG` and `ippiBGR565ToYCbCr_JPEG` are declared in the `ippj.h` file. They operate with ROI (see [Regions of Interest in Intel IPP](#)).

These functions convert an BGR image to the YCbCr image using the same formulas as in [ippiRGBToYCbCr\\_JPEG](#) function for computing *Y*, *Cb*, and *Cr* component values. The source image *pSrcBGR* has a reduced bit depth of 16 bits per pixel (see [Figure "16-bit pixel formats"](#)), and it can be in one of two possible formats : BGR565 (5 bits for blue, 6 bits for green, 5 bits for red), or BGR555 (5 bits for blue, green, red).

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if any of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> or <i>dstStep</i> has a zero or negative value.

## YCbCrToBGR565\_JPEG, YCbCrToBGR555\_JPEG

*Convert an YCbCr image to the BGR color model.*

---

### Syntax

```
ippStatus ippiYCbCrToBGR565_JPEG_8u16u_P3C3R(const Ipp8u* pSrcYCbCr[3], int srcStep,
Ipp16u* pDstBGR, int dstStep, IppiSize roiSize);
```

```
ippStatus ippiYCbCrToBGR555_JPEG_8u16u_P3C3R(const Ipp8u* pSrcYCbCr[3], int srcStep,
Ipp16u* pDstBGR, int dstStep, IppiSize roiSize);
```

### Parameters

<i>pSrcYCbCr</i>	An array of pointers to the ROIs in the separate source color planes.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDstBGR</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>roiSize</i>	Size of the source and destination ROI in pixels.

### Description

The functions `ippiYCbCrToBGR565_JPEG` and `ippiYCbCrToBGR555_JPEG` are declared in the `ippj.h` file. They operate with ROI (see [Regions of Interest in Intel IPP](#)).

These functions convert an YCbCr image to the BGR format using the same formulas as the [ippiYCbCrToRGB\\_JPEG](#) function for computing *R*, *G*, and *B* component values. The destination image *pDstBGR* has a reduced bit depth of 16 bits per pixel (see [Figure "16-bit pixel formats"](#)), and it can be in one of two possible formats : BGR565 (5 bits for blue, 6 bits for green, 5 bits for red), or BGR555 (5 bits for blue, green, red).

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
--------------------------	---------------------



<code>ippStsNullPtrErr</code>	Indicates an error condition if any of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <code>roiSize</code> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <code>srcStep</code> or <code>dstStep</code> has a zero or negative value.

## RGBToYCbCr422\_JPEG

Converts an RGB image to 4:2:2 YCbCr color model.

### Syntax

```
IppStatus ippiRGBToYCbCr422_JPEG_8u_C3P3R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst[3],
int dstStep[3], IppiSize roiSize);
```

```
IppStatus ippiRGBToYCbCr422_JPEG_8u_C4P3R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst[3],
int dstStep[3], IppiSize roiSize);
```

### Parameters

<code>pSrc</code>	Pointer to the source image ROI.
<code>srcStep</code>	Distance in bytes between starts of consecutive lines in the source image.
<code>pDst</code>	Array of pointers to the ROI in each plane of the destination image.
<code>dstStep</code>	Array of distance values in bytes between starts of consecutive lines in each planes of the destination image.
<code>roiSize</code>	Size of the source and destination ROI in pixels.

### Description

The function `ippiRGBToYCbCr422_JPEG` is declared in the `ippj.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts an RGB image to the YCbCr color model and stores results in `pDst` packed in **4:2:2 sampling** format (see [Figure "Different JPEG Sampling Formats"](#) for more details).

The function performs color conversion using the following formulas:

$$Y = 0.299 * R + 0.587 * G + 0.114 * B$$

$$Cb = -0.16874 * R - 0.33126 * G + 0.5 * B + 128$$

$$Cr = 0.5 * R - 0.41869 * G - 0.08131 * B + 128$$

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if any of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <code>roiSize</code> has a field with zero or negative value.

<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> or <i>dstStep</i> has a zero or negative value.
----------------------------	--

## YCbCr422ToRGB\_JPEG

Converts 16-bit per pixel YCbCr image to 24-bit per pixel RGB image.

---

### Syntax

#### Case 1: Operation on pixel-order data

```
IppStatus ippYCbCr422ToRGB_JPEG_8u_C2C3R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst,
int dstStep, IppiSize roiSize);
```

#### Case 2: Operation on planar data

```
IppStatus ippYCbCr422ToRGB_JPEG_8u_P3C3R(const Ipp8u* pSrc[3], int srcStep[3], Ipp8u*
pDst, int dstStep, IppiSize roiSize);
```

```
IppStatus ippYCbCr422ToRGB_JPEG_8u_P3C4R(const Ipp8u* pSrc[3], int srcStep[3], Ipp8u*
pDst, int dstStep, IppiSize roiSize, Ipp8u aval);
```

### Parameters

<i>pSrc</i>	Pointer to the source image ROI. Array of pointers to ROI in each plane of the planar source image.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in planes of the source image. Array of distance values in each plane of the planar source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>roiSize</i>	Size of the source and destination ROI in pixels.
<i>aval</i>	Constant value to create the fourth channel.

### Description

The function `ippYCbCr422ToRGB_JPEG` is declared in the `ippj.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts an YCbCr image *pSrc*, packed in [4:2:2 sampling](#) format (see [Figure “Different JPEG Sampling Formats”](#) for more details) to the RGB image *pDst* according to the same formulas as the function `ippYCbCrToRGB_JPEG` does. The output RGB values are saturated to the range `[0..255]`.

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pSrc</i> or <i>pDst</i> pointer is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.

## BGRToYCbCr422\_JPEG

*Converts a BGR image to 4:2:2 YCbCr color model.*

### Syntax

```
IppStatus ippBGRToYCbCr422_JPEG_8u_C3P3R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst[3],
int dstStep[3], IppiSize roiSize);
```

```
IppStatus ippBGRToYCbCr422_JPEG_8u_C4P3R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst[3],
int dstStep[3], IppiSize roiSize);
```

### Parameters

<i>pSrc</i>	Pointer to the source image ROI for pixel-order data. Array of pointers to the source ROI in each plane for planar image.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Array of pointers to the ROI in each plane of the destination image.
<i>dstStep</i>	Array of distance values in bytes between starts of consecutive lines in each planes of the destination image.
<i>roiSize</i>	Size of the source and destination ROI in pixels.

### Description

The function `ippBGRToYCbCr422_JPEG` is declared in the `ippj.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts a BGR image to the YCbCr color model and stores results in *pDst* packed in 4:2:2 sampling format (see [Figure "Different JPEG Sampling Formats"](#) for more details).

The function performs color conversion using the following formulas:

$$Y = 0.299 * R + 0.587 * G + 0.114 * B$$

$$Cb = -0.16874 * R - 0.33126 * G + 0.5 * B + 128$$

$$Cr = 0.5 * R - 0.41869 * G - 0.08131 * B + 128$$

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if any of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> or <i>dstStep</i> has a zero or negative value.

## YCbCr422ToBGR\_JPEG

*Converts 16-bit per pixel YCbCr image to 24-bit per pixel BGR image.*

---

### Syntax

```
IppStatus ippiYCbCr422ToBGR_JPEG_8u_P3C3R(const Ipp8u* pSrc[3], int srcStep[3], Ipp8u* pDst, int dstStep, IppiSize roiSize);
```

```
IppStatus ippiYCbCr422ToBGR_JPEG_8u_P3C4R(const Ipp8u* pSrc[3], int srcStep[3], Ipp8u* pDst, int dstStep, IppiSize roiSize, Ipp8u aval);
```

### Parameters

<i>pSrc</i>	Array of pointers to ROI in each plane of the source image.
<i>srcStep</i>	Array of distance values in bytes between starts of consecutive lines in each plane of the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>roiSize</i>	Size of the source and destination ROI in pixels.
<i>aval</i>	Constant value to create the fourth channel.

### Description

The function `ippiYCbCr422ToBGR_JPEG` is declared in the `ippj.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts an YCbCr planar image *pSrc*, packed in **4:2:2 sampling** format (see [Figure “Different JPEG Sampling Formats”](#) for more details) to the BGR image *pDst* according to the same formulas as the function `ippiYCbCrToRGB_JPEG` does. The output BGR values are saturated to the range [0..255].

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pSrc</i> or <i>pDst</i> pointer is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.

## RGBToYCbCr411\_JPEG

*Converts an RGB image to 4:1:1 YCbCr color model.*

---

### Syntax

```
IppStatus ippiRGBToYCbCr411_JPEG_8u_C3P3R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);
```

```
IppStatus ippiRGBToYCbCr411_JPEG_8u_C4P3R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);
```

## Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Array of pointers to the ROI in each plane of the destination image.
<i>dstStep</i>	Array of distance values in bytes between starts of consecutive lines in each planes of the destination image.
<i>roiSize</i>	Size of the source and destination ROI in pixels.

## Description

The function `ippiRGBToYCbCr411_JPEG` is declared in the `ippj.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts an RGB image to the YCbCr color model and stores results in *pDst* packed in 4:1:1 sampling format (see [Figure "Different JPEG Sampling Formats"](#) for more details).

The function performs color conversion using the following formulas:

$$Y = 0.299 * R + 0.587 * G + 0.114 * B$$

$$Cb = -0.16874 * R - 0.33126 * G + 0.5 * B + 128$$

$$Cr = 0.5 * R - 0.41869 * G - 0.08131 * B + 128$$

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if any of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> or <i>dstStep</i> has a zero or negative value.

## YCbCr4112ToRGB\_JPEG

*Converts 4:1:1 YCbCr image to the RGB image.*

### Syntax

```
ippStatus ippiYCbCr411ToRGB_JPEG_8u_P3C3R(const Ipp8u* pSrc[3], int srcStep[3], Ipp8u* pDst, int dstStep, IppiSize roiSize);
```

```
ippStatus ippiYCbCr411ToRGB_JPEG_8u_P3C4R(const Ipp8u* pSrc[3], int srcStep[3], Ipp8u* pDst, int dstStep, IppiSize roiSize, Ipp8u aval);
```

## Parameters

<i>pSrc</i>	Array of pointers to ROI in each plane of the source image.
<i>srcStep</i>	Array of distance values in bytes between starts of consecutive lines in each plane of the source image.

<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>roiSize</i>	Size of the source and destination ROI in pixels.
<i>aval</i>	Constant value to create the fourth channel.

### Description

The function `ippiYCbCr422ToRGB_JPEG` is declared in the `ippj.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts an YCbCr planar image *pSrc*, packed in **4:1:1 sampling** format (see [Figure "Different JPEG Sampling Formats"](#) for more details) to the RGB image *pDst* according to the same formulas as the function `ippiYCbCrToRGB_JPEG` does. The output RGB values are saturated to the range `[0..255]`.

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pSrc</i> or <i>pDst</i> pointer is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.

## BGRToYCbCr411\_JPEG

Converts an BGR image to 4:1:1 YCbCr color model.

### Syntax

```
ippStatus ippiBGRToYCbCr411_JPEG_8u_C3P3R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst[3],
int dstStep[3], IppiSize roiSize);
```

```
ippStatus ippiBGRToYCbCr411_JPEG_8u_C4P3R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst[3],
int dstStep[3], IppiSize roiSize);
```

### Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Array of pointers to the ROI in each plane of the destination image.
<i>dstStep</i>	Array of distance values in bytes between starts of consecutive lines in each planes of the destination image.
<i>roiSize</i>	Size of the source and destination ROI in pixels.

### Description

The function `ippiBGRToYCbCr411_JPEG` is declared in the `ippj.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts an BGR image to the YCbCr color model and stores results in *pDst* packed in **4:1:1 sampling** format (see [Figure “Different JPEG Sampling Formats”](#) for more details).

The function performs color conversion using the following formulas:

$$Y = 0.299 * R + 0.587 * G + 0.114 * B$$

$$Cb = -0.16874 * R - 0.33126 * G + 0.5 * B + 128$$

$$Cr = 0.5 * R - 0.41869 * G - 0.08131 * B + 128$$

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if any of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> or <i>dstStep</i> has a zero or negative value.

## YCbCr4112ToBGR\_JPEG

*Converts 4:1:1 YCbCr image to the BGR image.*

### Syntax

```
ippStatus ippiYCbCr411ToBGR_JPEG_8u_P3C3R(const Ipp8u* pSrc[3], int srcStep[3], Ipp8u* pDst, int dstStep, IppiSize roiSize);
```

```
ippStatus ippiYCbCr411ToBGR_JPEG_8u_P3C4R(const Ipp8u* pSrc[3], int srcStep[3], Ipp8u* pDst, int dstStep, IppiSize roiSize, Ipp8u aval);
```

### Parameters

<i>pSrc</i>	Array of pointers to ROI in each plane of the source image.
<i>srcStep</i>	Array of distance values in bytes between starts of consecutive lines in each plane of the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>roiSize</i>	Size of the source and destination ROI in pixels.
<i>aval</i>	Constant value to create the fourth channel.

### Description

The function `ippiYCbCr422ToBGR_JPEG` is declared in the `ippj.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts an YCbCr planar image *pSrc*, packed in **4:1:1 sampling** format (see [Figure “Different JPEG Sampling Formats”](#) for more details) to the BGR image *pDst* according to the same formulas as the function `ippiYCbCrToRGB_JPEG` does. The output BGR values are saturated to the range `[0..255]`.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <code>pSrc</code> or <code>pDst</code> pointer is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <code>roiSize</code> has a field with zero or negative value.

## CMYKToYCK\_JPEG

Converts a CMYK image to the YCK color model.

### Syntax

#### Case 1: Operation on planar data

```
ippStatus ippCMYKToYCK_JPEG_8u_P4R(const Ipp8u* pSrc[4], int srcStep, Ipp8u* pDst[4],
int dstStep, IppiSize roiSize);
```

#### Case 2: Operation on pixel-order data

```
ippStatus ippCMYKToYCK_JPEG_8u_C4P4R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst[4],
int dstStep, IppiSize roiSize);
```

### Parameters

<code>pSrc</code>	Array of pointers to the ROI in the separate planes for the planar source image. Pointer to ROI in the pixel-order source image.
<code>srcStep</code>	Distance in bytes between starts of consecutive lines in the source image.
<code>pDst</code>	Array of pointers to the ROI in the destination image.
<code>dstStep</code>	Distance in bytes between starts of consecutive lines in the destination image.
<code>roiSize</code>	Size of the source and destination ROI in pixels.

### Description

The function `ippCMYKToYCK_JPEG` is declared in the `ippj.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts a CMYK image to the YCK image in two steps. First, conversion is done into RGB format:

$$R = 255 - C$$

$$G = 255 - M$$

$$B = 255 - Y$$

After that, conversion to YCK image is performed as:

$$Y = 0.299*R + 0.587*G + 0.114*B$$

$$Cb = -0.16874*R - 0.33126*G + 0.5*B + 128$$

$$Cr = 0.5*R - 0.41869*G - 0.08131*B + 128$$



The values of K channel are written without modification.

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <code>roiSize</code> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <code>srcStep</code> or <code>dstStep</code> has a zero or negative value.

## YCCKToCMYK\_JPEG

*Converts an YCCK image to the CMYK color model.*

### Syntax

```
ippStatus ippYCCKToCMYK_JPEG_8u_P4R(const Ipp8u* pSrc[4], int srcStep, Ipp8u* pDst[4],
int dstStep, IppiSize roiSize);
```

```
ippStatus ippYCCKToCMYK_JPEG_8u_P4C4R(const Ipp8u* pSrc[4], int srcStep, Ipp8u* pDst,
int dstStep, IppiSize roiSize);
```

### Parameters

<code>pSrc</code>	Array of pointers to the ROI in each plane of the source image.
<code>srcStep</code>	Distance in bytes between starts of consecutive lines in the source image.
<code>pDst</code>	Array of pointers to the ROI in each planes of the destination planar image. Pointer to the ROI in the destination pixel-order image.
<code>dstStep</code>	Distance in bytes between starts of consecutive lines in the destination image.
<code>roiSize</code>	Size of the source and destination ROI in pixels.

### Description

The function `ippYCCKToCMYK_JPEG` is declared in the `ippj.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts an YCCK image to the CMYK image in two steps. First, conversion is done into RGB format as:

$$R = Y + 1.402 \cdot Cr - 179.456$$

$$G = Y - 0.34414 \cdot Cb - 0.71414 \cdot Cr + 135.45984$$

$$B = Y + 1.772 \cdot Cb - 226.816$$

After that, conversion to CMYK image is performed as follows:

$$C = 255 - R$$

$$M = 255 - G$$

$Y = 255 - B$

The values of K channel are written without modification.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if any of the specified pointers is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> or <i>dstStep</i> has a zero or negative value.

## Combined Color Conversion Functions

This section describes Intel IPP functions that perform sampling, color conversion and level shift (transition from unsigned data range to the signed one and vice versa) and are specific for JPEG codec. These functions are listed in the table below.

### Combined Color Conversion Functions

Function Base Name	Description
<code>RGBToYCbCr444LS_MCU</code>	Converts RGB images to YCbCr color model and creates 444 MCU.
<code>RGBToYCbCr422LS_MCU</code>	Converts RGB images to YCbCr color model and creates 422 MCU.
<code>RGBToYCbCr411LS_MCU</code>	Converts RGB images to YCbCr color model and creates 411 MCU.
<code>BGRToYCbCr444LS_MCU</code>	Converts BGR images to YCbCr color model and creates 444 MCU.
<code>BGR565ToYCbCr444LS_MCU, BGR555ToYCbCr444LS_MCU</code>	Convert 16-bit BGR images to YCbCr color model and create 444 MCU.
<code>BGRToYCbCr422LS_MCU</code>	Converts BGR images to YCbCr color model and creates 422 MCU.
<code>BGR565ToYCbCr422LS_MCU, BGR555ToYCbCr422LS_MCU</code>	Convert 16-bit BGR images to YCbCr color model and create 422 MCU.
<code>BGRToYCbCr411LS_MCU</code>	Converts BGR images to YCbCr color model and creates 411 MCU.
<code>BGR565ToYCbCr411LS_MCU, BGR555ToYCbCr411LS_MCU</code>	Convert 16-bit BGR images to YCbCr color model and create 411 MCU.
<code>CMYKToYCK444LS_MCU</code>	Converts CMYK images to YCK color model and creates 4444 MCU.
<code>CMYKToYCK422LS_MCU</code>	Converts CMYK images to YCK color model and creates 4224 MCU.
<code>CMYKToYCK411LS_MCU</code>	Converts CMYK images to YCK color model and creates 4114 MCU.
<code>YCbCr444ToRGBLS_MCU</code>	Creates YCbCr image from 444 MCU and converts it to RGB color model.
<code>YCbCr422ToRGBLS_MCU</code>	Creates YCbCr image from 422 MCU and converts it to RGB color model.
<code>YCbCr411ToRGBLS_MCU</code>	Creates YCbCr image from 411 MCU and converts it to RGB color model.
<code>YCbCr444ToBGRLS_MCU</code>	Creates YCbCr image from 444 MCU and converts it to BGR color model.
<code>YCbCr444ToBGR565LS_MCU, YCbCr444ToBGR555LS_MCU</code>	Create YCbCr image from 444 MCU and convert it to 16-bit BGR image.
<code>YCbCr422ToBGRLS_MCU</code>	Creates YCbCr image from 422 MCU and converts it to BGR color model.

Function Base Name	Description
<a href="#">YCbCr422ToBGR565LS_MCU</a> , <a href="#">YCbCr422ToBGR555LS_MCU</a>	Create YCbCr image from 422 MCU and convert it to 16-bit BGR image.
<a href="#">YCbCr411ToBGR565LS_MCU</a>	Creates YCbCr image from 411 MCU and converts it to BGR color model.
<a href="#">YCbCr411ToBGR555LS_MCU</a>	Create YCbCr image from 411 MCU and convert it to 16-bit BGR image.
<a href="#">YCK444ToCMYKLS_MCU</a>	Creates YCK image from 4444 MCU and converts it to CMYK color model.
<a href="#">YCK422ToCMYKLS_MCU</a>	Creates YCK image from 4224 MCU and converts it to CMYK color model.
<a href="#">YCK411ToCMYKLS_MCU</a>	Creates YCK image from 4114 MCU and converts it to CMYK color model.

Here MCU stands for JPEG Minimum Coded Unit, that is, an interleaved set of blocks of size determined by the sampling factors (see [Figure “MCU Structure for Different JPEG Sampling Formats”](#)), or a single block in a non interleaved scan.

All functions implement the level shift operation in accordance with its definition by [\[ISO10918\]](#), *Annex A.3.1, Level Shift*.

## RGBToYCbCr444LS\_MCU

*Converts an RGB image to the YCbCr color model and creates 444 MCU.*

### Syntax

```
ippStatus ippRGBToYCbCr444LS_MCU_8u16s_C3P3R(const Ipp8u* pSrcRGB, int srcStep, Ipp16s* pDstMCU[3]);
```

### Parameters

<i>pSrcRGB</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDstMCU</i>	Array of 3 pointers to the destination image blocks.

### Description

The function `ippRGBToYCbCr444LS_MCU` is declared in the `ippj.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts an RGB image to the YCbCr image using the same formulas as the `ippRGBToYCbCr_JPEG` function for computing  $Y$ ,  $Cb$ , and  $Cr$  component values.

Additionally, this function converts source data from unsigned `Ipp8u` range  $[0..255]$  to the signed `Ipp16s` range  $[-128..127]$  performing level shift operation (by subtracting 128) and creates 444 MCU (see [Figure “MCU Structure for Different JPEG Sampling Formats”](#)).

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
--------------------------	---------------------

<code>ippStsNullPtrErr</code>	Indicates an error condition if any of the specified pointers is <code>NULL</code> .
<code>ippStsStepErr</code>	Indicates an error condition if <code>srcStep</code> has a zero or negative value.

## RGBToYCbCr422LS\_MCU

*Converts an RGB image to the YCbCr color model and creates 422 MCU.*

---

### Syntax

```
IppStatus ippRGBToYCbCr422LS_MCU_8u16s_C3P3R(const Ipp8u* pSrcRGB, int srcStep, Ipp16s* pDstMCU[3]);
```

### Parameters

<code>pSrcRGB</code>	Pointer to the source image ROI.
<code>srcStep</code>	Distance in bytes between starts of consecutive lines in the source image.
<code>pDstMCU</code>	Array of 3 pointers to the destination image blocks.

### Description

The function `ippRGBToYCbCr422LS_MCU` is declared in the `ippj.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts an RGB image to the YCbCr image using the same formulas as the `ippRGBToYCbCr_JPEG` function for computing *Y*, *Cb*, and *Cr* component values.

Additionally, this function converts source data from unsigned `Ipp8u` range `[0..255]` to the signed `Ipp16s` range `[-128..127]` performing level shift operation (by subtracting 128) and creates 422 MCU (see [Figure "MCU Structure for Different JPEG Sampling Formats"](#)).

Downsampling is performed by plain averaging.

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if any of the specified pointers is <code>NULL</code> .
<code>ippStsStepErr</code>	Indicates an error condition if <code>srcStep</code> has a zero or negative value.

## Example

The code example below shows how to use the function `ippiRGBToYCbCr422LS_MCU_8u16s_C3P3R`.

```
int k=0;
int buf;

Ipp8u SrcRGB[48*8];
for(int i = 0; i < 48*8; i++){
    if(k==48) k=0;
    SrcRGB[i] = k++;
}

Ipp16s* DstMCU[3];
DstMCU[0] = ippiMalloc_16s_C1(16,8,&buf);
// no control on return value
DstMCU[1] = ippiMalloc_16s_C1(8,8,&buf);
DstMCU[2] = ippiMalloc_16s_C1(8,8,&buf);

ippiRGBToYCbCr422LS_MCU_8u16s_C3P3R(SrcRGB, 48, DstMCU);

result:

DstMCU[0]
-128 -125 -122 -119 -116 -113 -110 -107 -128 -125 -122 -119 -116 -113 -110 -107
-128 -125 -122 -119 -116 -113 -110 -107 -128 -125 -122 -119 -116 -113 -110 -107
-128 -125 -122 -119 -116 -113 -110 -107 -128 -125 -122 -119 -116 -113 -110 -107
-128 -125 -122 -119 -116 -113 -110 -107 -128 -125 -122 -119 -116 -113 -110 -107
-104 -101 -98 -95 -92 -89 -86 -83 -104 -101 -98 -95 -92 -89 -86 -83
-104 -101 -98 -95 -92 -89 -86 -83 -104 -101 -98 -95 -92 -89 -86 -83
-104 -101 -98 -95 -92 -89 -86 -83 -104 -101 -98 -95 -92 -89 -86 -83
-104 -101 -98 -95 -92 -89 -86 -83 -104 -101 -98 -95 -92 -89 -86 -83

DstMCU[1]
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
```

```
0 0 0 0 0 0 0 0
```

```
0 0 0 0 0 0 0 0
```

```
    DstMCU[2]
```

```
-1 -1 -1 -1 -1 -1 -1 -1
```

```
-1 -1 -1 -1 -1 -1 -1 -1
```

```
-1 -1 -1 -1 -1 -1 -1 -1
```

```
-1 -1 -1 -1 -1 -1 -1 -1
```

```
-1 -1 -1 -1 -1 -1 -1 -1
```

```
-1 -1 -1 -1 -1 -1 -1 -1
```

```
-1 -1 -1 -1 -1 -1 -1 -1
```

```
-1 -1 -1 -1 -1 -1 -1 -1
```

## RGBToYCbCr411LS\_MCU

*Converts an RGB image to the YCbCr color model and creates 411 MCU.*

---

### Syntax

```
IppStatus ippiRGBToYCbCr411LS_MCU_8u16s_C3P3R(const Ipp8u* pSrcRGB, int srcStep, Ipp16s* pDstMCU[3]);
```

### Parameters

<i>pSrcRGB</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDstMCU</i>	Array of 3 pointers to the destination image blocks.

### Description

The function `ippiRGBToYCbCr411LS_MCU` is declared in the `ippj.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)). This function converts an RGB image to the YCbCr image using the same formulas as the `ippiRGBToYCbCr_JPEG` function for computing *Y*, *Cb*, and *Cr* component values.

Additionally, this function converts source data from unsigned `Ipp8u` range `[0..255]` to the signed `Ipp16s` range `[-128..127]` performing level shift operation (by subtracting 128) and creates 411 MCU (see [Figure "MCU Structure for Different JPEG Sampling Formats"](#)).

Downsampling is performed by plain averaging.

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if any of the specified pointers is <code>NULL</code> .

`ippStsStepErr` Indicates an error condition if `srcStep` has a zero or negative value.

## BGRToYCbCr444LS\_MCU

*Converts a BGR image to the YCbCr color model and creates 444 MCU.*

---

### Syntax

```
IppStatus ippBGRToYCbCr444LS_MCU_8u16s_C3P3R(const Ipp8u* pSrcBGR, int srcStep, Ipp16s* pDstMCU[3]);
```

### Parameters

`pSrcBGR` Pointer to the source image ROI.

`srcStep` Distance in bytes between starts of consecutive lines in the source image.

`pDstMCU` Array of 3 pointers to the destination image blocks.

### Description

The function `ippBGRToYCbCr444LS_MCU` is declared in the `ippj.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)). This function converts a BGR image to the YCbCr image using the same formulas as the `ippiRGBToYCbCr_JPEG` function for computing *Y*, *Cb*, and *Cr* component values.

Additionally, this function converts source data from unsigned `Ipp8u` range [0..255] to the signed `Ipp16s` range [-128..127] performing level shift operation (by subtracting 128) and creates 444 MCU (see [Figure "MCU Structure for Different JPEG Sampling Formats"](#)).

### Return Values

`ippStsNoErr` Indicates no error.

`ippStsNullPtrErr` Indicates an error condition if any of the specified pointers is `NULL`.

`ippStsStepErr` Indicates an error condition if `srcStep` has a zero or negative value.

## BGR565ToYCbCr444LS\_MCU, BGR555ToYCbCr444LS\_MCU

*Convert a BGR image to the YCbCr color model and create 444 MCU.*

---

### Syntax

```
IppStatus ippBGR565ToYCbCr444LS_MCU_16u16s_C3P3R(const Ipp16u* pSrcBGR, int srcStep, Ipp16s* pDstMCU[3]);
```

```
IppStatus ippBGR555ToYCbCr444LS_MCU_16u16s_C3P3R(const Ipp16u* pSrcBGR, int srcStep, Ipp16s* pDstMCU[3]);
```

## Parameters

<i>pSrcBGR</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDstMCU</i>	Array of 3 pointers to the destination image blocks.

## Description

The functions `ippiBGR565ToYCbCr444LS_MCU` and `ippiBGR555ToYCbCr444LS_MCU` are declared in the `ippj.h` file. They operate with ROI (see [Regions of Interest in Intel IPP](#)).

These functions convert a source BGR image to the YCbCr 444 MCU (see [Figure "MCU Structure for Different JPEG Sampling Formats"](#)). The source image *pSrcBGR* has a reduced bit depth of 16 bits per pixel (see [Figure "16-bit pixel formats"](#)), and it can be in one of two possible formats : BGR565 (5 bits for blue, 6 bits for green, 5 bits for red), or BGR555 (5 bits for blue, green, red).

Source data are converted to the signed `Ipp16s` range `[-128..127]` by transforming them at first to the unsigned `Ipp8u` range with following level shift operation (by subtracting 128). *Y*, *Cb*, and *Cr* component values are computed using the same formulas as the `ippiRGBToYCbCr_JPEG` function.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if any of the specified pointers is <code>NULL</code> .
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> has a zero or negative value.

## BGRToYCbCr422LS\_MCU

*Converts an BGR image to the YCbCr color model and creates 422 MCU.*

---

## Syntax

```
IppStatus ippiBGRToYCbCr422LS_MCU_8u16s_C3P3R(const Ipp8u* pSrcBGR, int srcStep, Ipp16s* pDstMCU[3]);
```

## Parameters

<i>pSrcBGR</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDstMCU</i>	Array of 3 pointers to the destination image blocks.

## Description

The function `ippiBGRToYCbCr422LS_MCU` is declared in the `ippj.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).



This function converts an BGR image to the YCbCr image using the same formulas as the [ippiRGBToYCbCr\\_JPEG](#) function for computing  $Y$ ,  $Cb$ , and  $Cr$  component values.

Additionally, this function converts source data from unsigned `Ipp8u` range  $[0..255]$  to the signed `Ipp16s` range  $[-128..127]$  performing level shift operation (by subtracting 128) and creates 422 MCU (see [Figure "MCU Structure for Different JPEG Sampling Formats"](#)).

Downsampling is performed by plain averaging.

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if any of the specified pointers is <code>NULL</code> .
<code>ippStsStepErr</code>	Indicates an error condition if <code>srcStep</code> has a zero or negative value.

## BGR565ToYCbCr422LS\_MCU, BGR555ToYCbCr422LS\_MCU

*Convert an BGR image to the YCbCr color model and create 422 MCU.*

### Syntax

```
IppStatus ippiBGR565ToYCbCr422LS_MCU_16u16s_C3P3R(const Ipp16u* pSrcBGR, int srcStep, Ipp16s* pDstMCU[3]);
```

```
IppStatus ippiBGR555ToYCbCr422LS_MCU_16u16s_C3P3R(const Ipp16u* pSrcBGR, int srcStep, Ipp16s* pDstMCU[3]);
```

### Parameters

<code>pSrcBGR</code>	Pointer to the source image ROI.
<code>srcStep</code>	Distance in bytes between starts of consecutive lines in the source image.
<code>pDstMCU</code>	Array of 3 pointers to the destination image blocks.

### Description

The functions `ippiBGR565ToYCbCr422LS_MCU` and `ippiBGR555ToYCbCr422LS_MCU` are declared in the `ippj.h` file. They operate with ROI (see [Regions of Interest in Intel IPP](#)).

These functions convert a source BGR image to the YCbCr 422 MCU (see [Figure "MCU Structure for Different JPEG Sampling Formats"](#)). The source image `pSrcBGR` has a reduced bit depth of 16 bits per pixel (see [Figure "16-bit pixel formats"](#)), and it can be in one of two possible formats : BGR565 (5 bits for blue, 6 bits for green, 5 bits for red), or BGR555 (5 bits for blue, green, red).

Source data are converted to the signed `Ipp16s` range  $[-128..127]$  by transforming them at first to the unsigned `Ipp8u` range with following level shift operation (by subtracting 128).  $Y$ ,  $Cb$ , and  $Cr$  component values are computed using the same formulas as the [ippiRGBToYCbCr\\_JPEG](#) function.

Downsampling is performed by plain averaging.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if any of the specified pointers is <code>NULL</code> .
<code>ippStsStepErr</code>	Indicates an error condition if <code>srcStep</code> has a zero or negative value.

## BGRToYCbCr411LS\_MCU

*Converts an BGR image to the YCbCr color model and creates 411 MCU.*

---

### Syntax

```
ippStatus ippIBGRToYCbCr411LS_MCU_8u16s_C3P3R(const Ipp8u* pSrcBGR, int srcStep, Ipp16s* pDstMCU[3]);
```

### Parameters

<code>pSrcBGR</code>	Pointer to the source image ROI.
<code>srcStep</code>	Distance in bytes between starts of consecutive lines in the source image.
<code>pDstMCU</code>	Array of 3 pointers to the destination image blocks.

### Description

The function `ippIBGRToYCbCr411LS_MCU` is declared in the `ippj.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts an BGR image to the YCbCr image using the same formulas as the `ippiRGBToYCbCr_JPEG` function for computing *Y*, *Cb*, and *Cr* component values.

Additionally, this function converts source data from unsigned `Ipp8u` range `[0..255]` to the signed `Ipp16s` range `[-128..127]` performing level shift operation (by subtracting 128) and creates 411 MCU (see [Figure "MCU Structure for Different JPEG Sampling Formats"](#)).

Downsampling is performed by plain averaging.

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if any of the specified pointers is <code>NULL</code> .
<code>ippStsStepErr</code>	Indicates an error condition if <code>srcStep</code> has a zero or negative value.

## GR565ToYCbCr411LS\_MCU, BGR555ToYCbCr411LS\_MCU

*Convert an BGR image to the YCbCr color model and create 411 MCU.*

---

### Syntax

```
IppStatus ippBGR565ToYCbCr411LS_MCU_16u16s_C3P3R(const Ipp16u* pSrcBGR, int srcStep,
Ipp16s* pDstMCU[3]);
```

```
IppStatus ippBGR555ToYCbCr411LS_MCU_16u16s_C3P3R(const Ipp16u* pSrcBGR, int srcStep,
Ipp16s* pDstMCU[3]);
```

### Parameters

<i>pSrcBGR</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDstMCU</i>	Array of 3 pointers to the destination image blocks.

### Description

The functions `ippBGR565ToYCbCr411LS_MCU` and `ippBGR555ToYCbCr411LS_MCU` are declared in the `ippj.h` file. They operate with ROI (see [Regions of Interest in Intel IPP](#)).

These functions convert a source BGR image to the YCbCr 411 MCU (see [Figure "MCU Structure for Different JPEG Sampling Formats"](#)). The source image *pSrcBGR* has a reduced bit depth of 16 bits per pixel (see [Figure "16-bit pixel formats"](#)), and it can be in one of two possible formats : BGR565 (5 bits for blue, 6 bits for green, 5 bits for red), or BGR555 (5 bits for blue, green, red).

Source data are converted to the signed `Ipp16s` range  $[-128..127]$  by transforming them at first to the unsigned `Ipp8u` range with following level shift operation (by subtracting 128). *Y*, *Cb*, and *Cr* component values are computed using the same formulas as the `ippiRGBToYCbCr_JPEG` function.

Downsampling is performed by plain averaging.

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if any of the specified pointers is <code>NULL</code> .
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> has a zero or negative value.

## CMYKToYCCK444LS\_MCU

*Converts a CMYK image to the YCCK color model and creates 4444 MCU.*

---

### Syntax

```
IppStatus ippCMYKToYCCK444LS_MCU_8u16s_C4P4R(const Ipp8u* pSrcCMYK, int srcStep, Ipp16s*
pDstMCU[4]);
```

## Parameters

<i>pSrcCMYK</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDstMCU</i>	Array of 4 pointers to the destination image blocks.

## Description

The function `ippiCMYKToYCKK444LS_MCU` is declared in the `ippj.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts a CMYK image to the YCKK image in the same way as the function `ippiCMYKToYCKK_JPEG` does.

Additionally, this function converts source data from unsigned `Ipp8u` range `[0..255]` to the signed `Ipp16s` range `[-128..127]` performing level shift operation (by subtracting 128) and creates 4444 MCU.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if any of the specified pointers is <code>NULL</code> .
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> has a zero or negative value.

## CMYKToYCKK422LS\_MCU

*Converts a CMYK image to the YCKK color model and creates 4224 MCU.*

---

## Syntax

```
IppStatus ippiCMYKToYCKK422LS_MCU_8u16s_C4P4R(const Ipp8u* pSrcCMYK, int srcStep, Ipp16s* pDstMCU[4]);
```

## Parameters

<i>pSrcCMYK</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDstMCU</i>	Array of 4 pointers to the destination image blocks.

## Description

The function `ippiCMYKToYCKK422LS_MCU` is declared in the `ippj.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts a CMYK image to the YCKK image in the same way as the function `ippiCMYKToYCKK_JPEG` does.

Additionally, this function converts source data from unsigned `Ipp8u` range `[0..255]` to the signed `Ipp16s` range `[-128..127]` performing level shift operation (by subtracting 128) and creates 4224 MCU.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if any of the specified pointers is <code>NULL</code> .
<code>ippStsStepErr</code>	Indicates an error condition if <code>srcStep</code> has a zero or negative value.

## CMYKToYCCK411LS\_MCU

*Converts a CMYK image to the YCCK color model and creates 4114 MCU.*

---

### Syntax

```
ippStatus ippCMYKToYCCK411LS_MCU_8u16s_C4P4R(const Ipp8u* pSrcCMYK, int srcStep, Ipp16s* pDstMCU[4]);
```

### Parameters

<code>pSrcCMYK</code>	Pointer to the source image ROI.
<code>srcStep</code>	Distance in bytes between starts of consecutive lines in the source image.
<code>pDstMCU</code>	Array of 4 pointers to the destination image blocks.

### Description

The function `ippiCMYKToYCCK411LS_MCU` is declared in the `ippj.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)). This function converts a CMYK image to the YCCK image in the same way as the function `ippiCMYKToYCCK_JPEG` does.

Additionally, this function converts data from unsigned `Ipp8u` range `[0..255]` to the signed `Ipp16s` range `[-128..127]` performing level shift operation (by subtracting 128) and creates 4114 MCU.

Downsampling is performed by plain averaging.

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if any of the specified pointers is <code>NULL</code> .
<code>ippStsStepErr</code>	Indicates an error condition if <code>srcStep</code> has a zero or negative value.

## YCbCr444ToRGBLS\_MCU

*Creates an YCbCr image from 444 MCU and converts it to the RGB color model.*

---

### Syntax

```
ippStatus ippiYCbCr444ToRGBLS_MCU_16s8u_P3C3R(const Ipp16s* pSrcMCU[3], Ipp8u* pDstRGB, int dstStep);
```

## Parameters

<i>pSrcMCU</i>	Array of 3 pointers to the source image blocks.
<i>pDstRGB</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.

## Description

The function `ippiYCbCr444ToRGBLS_MCU` is declared in the `ippj.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)). This function creates YCbCr 8x8 image from 444 MCU and converts it to the RGB format using the same formulas as the `ippiYCbCrToRGB_JPEG` function for computing *R*, *G*, and *B* component values. Additionally, this function converts data from the signed `Ipp16s` range `[-128..127]` to the unsigned `Ipp8u` range `[0..255]` performing level shift operation (by adding 128).

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if any of the specified pointers is <code>NULL</code> .
<code>ippStsStepErr</code>	Indicates an error condition if <i>dstStep</i> has a zero or negative value.

## YCbCr422ToRGBLS\_MCU

*Creates an YCbCr image from 422 MCU and converts it to the RGB color model.*

---

## Syntax

```
IppStatus ippiYCbCr422ToRGBLS_MCU_16s8u_P3C3R(const Ipp16s* pSrcMCU[3], Ipp8u* pDstRGB, int dstStep);
```

## Parameters

<i>pSrcMCU</i>	Array of 3 pointers to the source image blocks.
<i>pDstRGB</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.

## Description

The function `ippiYCbCr422ToRGBLS_MCU` is declared in the `ippj.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)). This function creates YCbCr 16x8 image from 422 MCU and converts it to the RGB format using the same formulas as the `ippiYCbCrToRGB_JPEG` function for computing *R*, *G*, and *B* component values. Additionally, this function converts data from the signed `Ipp16s` range `[-128..127]` to the unsigned `Ipp8u` range `[0..255]` performing level shift operation (by adding 128).

Upsampling is performed by replication of the neighbor value.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if any of the specified pointers is <code>NULL</code> .
<code>ippStsStepErr</code>	Indicates an error condition if <code>dstStep</code> has a zero or negative value.

## YCbCr411ToRGBLS\_MCU

*Creates an YCbCr image from 411 MCU and converts it to the RGB color model.*

---

### Syntax

```
ippStatus ippYCbCr411ToRGBLS_MCU_16s8u_P3C3R(const Ipp16s* pSrcMCU[3], Ipp8u* pDstRGB,
int dstStep);
```

### Parameters

<code>pSrcMCU</code>	Array of 3 pointers to the source image blocks.
<code>pDstRGB</code>	Pointer to the destination image ROI.
<code>dstStep</code>	Distance in bytes between starts of consecutive lines in the destination image.

### Description

The function `ippYCbCr411ToRGBLS_MCU` is declared in the `ippj.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)). This function creates YCbCr 16x16 image from 411 MCU and converts it to the RGB format using the same formulas as the `ippYCbCrToRGB_JPEG` function for computing *R*, *G*, and *B* component values.

Additionally, this function converts data from the signed `Ipp16s` range `[-128..127]` to the unsigned `Ipp8u` range `[0..255]` performing level shift operation (by adding 128).

Upsampling is performed by replication of the neighbor value.

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if any of the specified pointers is <code>NULL</code> .
<code>ippStsStepErr</code>	Indicates an error condition if <code>dstStep</code> has a zero or negative value.

## YCbCr444ToBGRLS\_MCU

*Creates an YCbCr image from 444 MCU and converts it to the BGR color model.*

---

### Syntax

```
ippStatus ippYCbCr444ToBGRLS_MCU_16s8u_P3C3R(const Ipp16s* pSrcMCU[3], Ipp8u* pDstBGR,
int dstStep);
```

## Parameters

<i>pSrcMCU</i>	Array of 3 pointers to the source image blocks.
<i>pDstBGR</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.

## Description

The function `ippiYCbCr444ToBGR565LS_MCU` is declared in the `ippj.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)). This function creates YCbCr 8x8 image from 444 MCU and converts it to the BGR format using the same formulas as the `ippiYCbCrToRGB_JPEG` function for computing *B*, *G*, and *R* component values. Additionally, this function converts data from the signed `Ipp16s` range `[-128..127]` to the unsigned `Ipp8u` range `[0..255]` performing level shift operation (by adding 128).

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if any of the specified pointers is <code>NULL</code> .
<code>ippStsStepErr</code>	Indicates an error condition if <i>dstStep</i> has a zero or negative value.

## YCbCr444ToBGR565LS\_MCU, YCbCr444ToBGR555LS\_MCU

*Create an YCbCr image from 444 MCU and convert it to the BGR color model.*

---

## Syntax

```
IppStatus ippiYCbCr444ToBGR565LS_MCU_16s16u_P3C3R(const Ipp16s* pSrcMCU[3], Ipp16u* pDstBGR, int dstStep);
```

```
IppStatus ippiYCbCr444ToBGR555LS_MCU_16s16u_P3C3R(const Ipp16s* pSrcMCU[3], Ipp16u* pDstBGR, int dstStep);
```

## Parameters

<i>pSrcMCU</i>	Array of 3 pointers to the source image blocks.
<i>pDstBGR</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.

## Description

The functions `ippiYCbCr444ToBGR565LS_MCU` and `ippiYCbCr444ToBGR555LS_MCU` are declared in the `ippj.h` file. They operate with ROI (see [Regions of Interest in Intel IPP](#)). These functions create YCbCr 8x8 image from 444 MCU (see [Figure "MCU Structure for Different JPEG Sampling Formats"](#)) and convert it to the BGR format using the same formulas as the `ippiYCbCrToRGB_JPEG` function for computing *B*, *G*, and *R* component values.



Additionally, the functions convert data from the signed `Ipp16s` range `[-128..127]` to the unsigned `Ipp8u` range `[0..255]` performing level shift operation (by adding 128). The destination image `pDstBGR` has a reduced bit depth of 16 bits per pixel (see [Figure "16-bit pixel formats"](#)), and it can be in one of two possible formats : BGR565 (5 bits for blue, 6 bits for green, 5 bits for red), or BGR555 (5 bits for blue, green, red).

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if any of the specified pointers is <code>NULL</code> .
<code>ippStsStepErr</code>	Indicates an error condition if <code>dstStep</code> has a zero or negative value.

## YCbCr422ToBGRLS\_MCU

*Creates a YCbCr image from 422 MCU and converts it to the BGR color model.*

### Syntax

```
IppStatus ippiYCbCr422ToBGRLS_MCU_16s8u_P3C3R(const Ipp16s* pSrcMCU[3], Ipp8u* pDstBGR,
int dstStep);
```

### Parameters

<code>pSrcMCU</code>	Array of 3 pointers to the source image blocks.
<code>pDstBGR</code>	Pointer to the destination image ROI.
<code>dstStep</code>	Distance in bytes between starts of consecutive lines in the destination image.

### Description

The function `ippiYCbCr422ToBGRLS_MCU` is declared in the `ippj.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)). This function creates YCbCr 16x8 image from 422 MCU (see [Figure "MCU Structure for Different JPEG Sampling Formats"](#)) and converts it to the BGR format using the same formulas as the `ippiYCbCrToRGB_JPEG` function for computing *B*, *G*, and *R* component values. Upsampling is performed by replication of the neighbor value.

Additionally, this function converts data from the signed `Ipp16s` range `[-128..127]` to the unsigned `Ipp8u` range `[0..255]` performing level shift operation (by adding 128).

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if any of the specified pointers is <code>NULL</code> .
<code>ippStsStepErr</code>	Indicates an error condition if <code>dstStep</code> has a zero or negative value.

## YCbCr422ToBGR565LS\_MCU, YCbCr422ToBGR555LS\_MCU

Create an YCbCr image from 422 MCU and convert it to the BGR color model.

---

### Syntax

```
IppStatus ippiYCbCr422ToBGR565LS_MCU_16s16u_P3C3R(const Ipp16s* pSrcMCU[3], Ipp16u* pDstBGR,
int dstStep);
```

```
IppStatus ippiYCbCr422ToBGR555LS_MCU_16s16u_P3C3R(const Ipp16s* pSrcMCU[3], Ipp16u* pDstBGR,
int dstStep);
```

### Parameters

<i>pSrcMCU</i>	Array of 3 pointers to the source image blocks.
<i>pDstBGR</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.

### Description

The functions `ippiYCbCr444ToBGR565LS_MCU` and `ippiYCbCr444ToBGR555LS_MCU` are declared in the `ippj.h` file. They operate with ROI (see [Regions of Interest in Intel IPP](#)). These functions create YCbCr 8x8 image from 422 MCU (see [Figure "MCU Structure for Different JPEG Sampling Formats"](#)) and convert it to the BGR format using the same formulas as the `ippiYCbCrToRGB_JPEG` function for computing *B*, *G*, and *R* component values. Upsampling is performed by replication of the neighbor value.

Additionally, the functions convert data from the signed `Ipp16s` range `[-128..127]` to the unsigned `Ipp8u` range `[0..255]` performing level shift operation (by adding 128). The destination image *pDstBGR* has a reduced bit depth of 16 bits per pixel (see [Figure "16-bit pixel formats"](#)), and it can be in one of two possible formats : BGR565 (5 bits for blue, 6 bits for green, 5 bits for red), or BGR555 (5 bits for blue, green, red).

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if any of the specified pointers is <code>NULL</code> .
<code>ippStsStepErr</code>	Indicates an error condition if <i>dstStep</i> has a zero or negative value.

## YCbCr411ToBGR1S\_MCU

Creates an YCbCr image from 411 MCU and converts it to the BGR color model.

---

### Syntax

```
IppStatus ippiYCbCr411ToBGR1S_MCU_16s8u_P3C3R(const Ipp16s* pSrcMCU[3], Ipp8u* pDstBGR,
int dstStep);
```

## Parameters

<i>pSrcMCU</i>	Array of 3 pointers to the source image blocks.
<i>pDstBGR</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.

## Description

The function `ippiYCbCr411ToBGR565LS_MCU` is declared in the `ippj.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)). This function creates YCbCr 16x16 image from 411 MCU and converts it to the BGR format using the same formulas as the `ippiYCbCrToRGB_JPEG` function for computing *B*, *G*, and *R* component values. Upsampling is performed by replication of the neighbor value.

Additionally, this function converts data from the signed `Ipp16s` range `[-128..127]` to the unsigned `Ipp8u` range `[0..255]` performing level shift operation (by adding 128).

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if any of the specified pointers is <code>NULL</code> .
<code>ippStsStepErr</code>	Indicates an error condition if <i>dstStep</i> has a zero or negative value.

## YCbCr411ToBGR565LS\_MCU, YCbCr411ToBGR555LS\_MCU

*Create an YCbCr image from 411 MCU and convert it to the BGR color model.*

## Syntax

```
IppStatus ippiYCbCr411ToBGR565LS_MCU_16s16u_P3C3R(const Ipp16s* pSrcMCU[3], Ipp16u* pDstBGR, int dstStep);
```

```
IppStatus ippiYCbCr411ToBGR555LS_MCU_16s16u_P3C3R(const Ipp16s* pSrcMCU[3], Ipp16u* pDstBGR, int dstStep);
```

## Parameters

<i>pSrcMCU</i>	Array of 3 pointers to the source image blocks.
<i>pDstBGR</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.

## Description

The functions `ippiYCbCr411ToBGR565LS_MCU` and `ippiYCbCr411ToBGR555LS_MCU` are declared in the `ippj.h` file. They operate with ROI (see [Regions of Interest in Intel IPP](#)). These functions create YCbCr 8x8 image from 411 MCU (see [Figure "MCU Structure for Different JPEG Sampling Formats"](#)) and convert it to the BGR format using the same formulas as the `ippiYCbCrToRGB_JPEG` function for computing *B*, *G*, and *R* component values. Upsampling is performed by replication of the neighbor value.

Additionally, the functions convert data from the signed `Ipp16s` range `[-128..127]` to the unsigned `Ipp8u` range `[0..255]` performing level shift operation (by adding 128). The destination image `pDstBGR` has a reduced bit depth of 16 bits per pixel (see [Figure "16-bit pixel formats"](#)), and it can be in one of two possible formats : BGR565 (5 bits for blue, 6 bits for green, 5 bits for red), or BGR555 (5 bits for blue, green, red).

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if any of the specified pointers is <code>NULL</code> .
<code>ippStsStepErr</code>	Indicates an error condition if <code>dstStep</code> has a zero or negative value.

## YCCK444ToCMYKLS\_MCU

*Creates YCCK image from 4444 MCU and converts it to the CMYK color model.*

---

### Syntax

```
IppStatus ippYCK444ToCMYKLS_MCU_16s8u_P4C4R(const Ipp16s* pSrcMCU[4], Ipp8u* pDstCMYK,
int dstStep);
```

### Parameters

<code>pSrcMCU</code>	Array of 4 pointers to the source image blocks.
<code>pDstCVBYK</code>	Pointer to the destination image ROI.
<code>dstStep</code>	Distance in bytes between starts of consecutive lines in the destination image.

### Description

The function `ippiYCK444ToCMYKLS_MCU` is declared in the `ippj.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)). This function creates YCCK 8x8 image from 4444 MCU and converts it to the CMYK color model in the same way as the function `ippiYCKToCMYK_JPEG` does.

Additionally, this function converts data from the signed `Ipp16s` range `[-128..127]` to the unsigned `Ipp8u` range `[0..255]` performing level shift operation (by adding 128).

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if any of the specified pointers is <code>NULL</code> .
<code>ippStsStepErr</code>	Indicates an error condition if <code>dstStep</code> has a zero or negative value.

## YCK422ToCMYKLS\_MCU

*Creates an YCK image from 4224 MCU and converts it to the CMYK color model.*

---

### Syntax

```
IppStatus ippYCK422ToCMYKLS_MCU_16s8u_P4C4R(const Ipp16s* pSrcMCU[4], Ipp8u* pDstCMYK,
int dstStep);
```

### Parameters

<i>pSrcMCU</i>	Array of 4 pointers to the source image blocks.
<i>pDstCVBYK</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.

### Description

The function `ippYCK422ToCMYKLS_MCU` is declared in the `ippj.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)). This function creates YCK 16x8 image from 4224 MCU and converts it to the CMYK color model in the same way as the function `ippYCKToCMYK_JPEG` does. Upsampling is performed by replication of the neighbor value.

Additionally, this function converts data from the signed `Ipp16s` range `[-128..127]` to the unsigned `Ipp8u` range `[0..255]` performing level shift operation (by adding 128).

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if any of the specified pointers is <code>NULL</code> .
<code>ippStsStepErr</code>	Indicates an error condition if <i>dstStep</i> has a zero or negative value.

## YCK411ToCMYKLS\_MCU

*Creates an YCK image from 4114 MCU and converts it to the CMYK color model.*

---

### Syntax

```
IppStatus ippYCK411ToCMYKLS_MCU_16s8u_P4C4R(const Ipp16s* pSrcMCU[4], Ipp8u* pDstCMYK,
int dstStep);
```

### Parameters

<i>pSrcMCU</i>	Array of 4 pointers to the source image blocks.
<i>pDstCVBYK</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.

## Description

The function `ippiYCKK411ToCMYKLS_MCU` is declared in the `ippj.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)). This function creates YCKK 16x16 image from 4114 MCU and converts it to the CMYK color model in the same way as the function `ippiYCKKToCMYK_JPEG` does. Upsampling is performed by replication of the neighbor value.

Additionally, this function converts data from the signed `Ipp16s` range `[-128..127]` to the unsigned `Ipp8u` range `[0..255]` performing level shift operation (by adding 128).

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if any of the specified pointers is <code>NULL</code> .
<code>ippStsStepErr</code>	Indicates an error condition if <code>dstStep</code> has a zero or negative value.

# Quantization Functions

This section describes Intel IPP functions that perform quantization, dequantization and preparation of quantization tables and are specific for JPEG codec. These functions are listed in [Table "Quantization Functions"](#).

## Quantization Functions

Function Base Name	Description
<code>QuantFwdRawTableInit_JPEG</code>	Modifies raw quantization tables in accordance with quality factor.
<code>QuantFwdTableInit_JPEG</code>	Prepares quantization tables in a format suitable for an encoder.
<code>QuantFwd8x8_JPEG</code>	Performs quantization of an 8x8 block of DCT coefficients.
<code>QuantInvTableInit_JPEG</code>	Prepares quantization tables in a format suitable for a decoder.
<code>QuantFwd8x8_JPEG</code>	Performs dequantization of DCT coefficients in 8x8 block

## QuantFwdRawTableInit\_JPEG

*Modifies raw quantization tables in accordance with the quality factor.*

## Syntax

```
IppStatus ippiQuantFwdRawTableInit_JPEG_8u(Ipp8u* pQuantRawTable, int qualityFactor);
```

## Parameters

<code>pQuantRawTable</code>	Pointer to the raw quantization table.
<code>qualityFactor</code>	JPEG quality factor, must be in the range <code>[1..100]</code> .

## Description

The function `ippiQuantFwdRawTableInit_JPEG` is declared in the `ippj.h` file. This function modifies the initial raw quantization table in accordance with the quality factor using the algorithm specified in IJG JPEG Library, version 6b ([www.ijg.org](http://www.ijg.org)). This function is optional for JPEG codec implementation.

Example “Using Intel IPP for Quantization of 8x8 Block of DCT Coefficients” and Example “Using Combined DCT Functions” show how to use the function `ippiQuantFwdRawTableInit_JPEG_8u`.

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if any of the specified pointers are NULL.

## QuantFwdTableInit\_JPEG

*Prepares quantization tables suitable for fast encoding.*

### Syntax

```
ippStatus ippiQuantFwdTableInit_JPEG_8u16u(const Ipp8u* pQuantRawTable, Ipp16u* pQuantFwdTable);
```

### Parameters

<code>pQuantRawTable</code>	Pointer to the raw quantization table.
<code>pQuantFwdTable</code>	Pointer to the quantization table for the encoder.

### Description

The function `ippiQuantFwdTableInit_JPEG` is declared in the `ippj.h` file. This function prepares quantization table in a format that is suitable for fast encoding. Initial raw quantization tables have zigzag order by definition. This function performs reordering transformation that converts the zigzag sequence of table elements to conventional order (left-to-right, top-to-bottom). To avoid division during quantization, the function `ippiQuantFwdTableInit_JPEG` scales the array by 15 bits. The pointer `pQuantRawTable` points to the array of 64 elements as is required by [ISO10918], Annex B.2.4.1, *Quantization Table Specification Syntax*. The pointer `pQuantFwdTable` points to an array containing 64 values of `Ipp16u` type.

Example “Using Intel IPP for Quantization of 8x8 Block of DCT Coefficients” and Example “Using Combined DCT Functions” show how to use the function `ippiQuantFwdTableInit_JPEG_8u16u`.

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one or both of the specified pointers are NULL.

## QuantFwd8x8\_JPEG

*Performs quantization of an 8x8 block of DCT coefficients.*

### Syntax

```
IppStatus ippiQuantFwd8x8_JPEG_16s_C1I(Ipp16s* pSrcDst, const Ipp16u* pQuantFwdTable);
```

## Parameters

<code>pSrcDst</code>	Pointer to the 8x8 block of DCT coefficients.
<code>pQuantFwdTable</code>	Pointer to the quantization table for the encoder.

## Description

The function `ippiQuantFwd8x8_JPEG` is declared in the `ippj.h` file. This function performs quantization of computed DCT coefficients for an 8x8 block. Quantization is defined in [ISO10918], Annex A.3.4, *DCT Coefficient Quantization and Dequantization*, as

$$Sq(u,v) = \text{round}(S(u,v)/Q(u,v)),$$

where  $Sq(u,v)$  is the quantized DCT coefficient,  $S(u,v)$  is a computed DCT coefficient, and  $Q(u,v)$  is a quantization table element.

Rounding is to the nearest integer. To avoid division, quantization table elements are scaled up by  $2^{15}$ . After that, division operation is replaced by multiplication of DCT coefficients by scaled quantization table elements. Then the products are rescaled back by 15 bits.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one or both specified pointers are NULL.



## Example

The code example below shows how to use the function `ippiQuantFwd8x8_JPEG_16s_C1I`.

```
Ipp8u pQuantRawTable[64]= {16, 11, 12, 14, 12, 10, 16, 14,
                           13, 14, 18, 17, 16, 19, 24, 40,
                           26, 24, 22, 22, 24, 49, 35, 37,
                           29, 40, 58, 51, 61, 60, 57, 51,
                           56, 55, 64, 72, 92, 78, 64, 68,
                           87, 69, 55, 56, 80, 109, 81, 87,
                           95, 98, 103, 104, 103, 62, 77, 113,
                           121, 112, 100, 120, 92, 101, 103, 99 };
```

```
Ipp16s dstFwd[8*8] = { 4, 4, 4, 4, 4, 4, 4, 4,
                        4, 3, 3, 3, -3, 3, -3, 4,
                        4, -3, 2, 0, 0, 2, 3, 4,
                        4, 3, 0, 1, -1, 0, -3, 4,
                        4, -3, 0, -1, 1, 0, 3, 4,
                        4, 3, 2, 0, 0, 2, -3, 4,
                        4, -3, 3, -3, 3, -3, 3, 4,
                        4, 4, 4, 4, 4, 4, 4, 4};
```

```
Ipp16u pQuantFwdTable[64];
```

```
int quality = 75;
```

```
ippiQuantFwdRawTableInit_JPEG_8u ( pQuantRawTable, quality );
```

```
ippiQuantFwdTableInit_JPEG_8u16u ( pQuantRawTable, pQuantFwdTable );
```

```
ippiQuantFwd8x8_JPEG_16s_C1I ( dstFwd, pQuantFwdTable );
```

```
result:
```

```
pQuantRawTable (with quality)
```

```
8 6 6 7 6 5 8 7
```

```
7 7 9 9 8 10 12 20
```

```
13 12 11 11 12 25 18 19
```

```
15 20 29 26 31 30 29 26
```

```
28 28 32 36 46 39 32 34
```

```
44 35 28 28 40 55 41 44
48 49 52 52 52 31 39 57
61 56 50 60 46 51 52 50
```

pQuantFwdTable

```
4096 5461 6554 4096 2731 1638 1260 1057
5461 5461 4681 3277 2521 1130 1092 1170
4681 4681 4096 2731 1638 1130 936 1170
4681 3641 2979 2185 1260 745 819 1057
3641 2979 1725 1170 964 596 630 840
2731 1820 1170 1024 799 630 575 712
1311 1024 840 745 630 537 546 643
910 712 683 669 585 655 630 655
```

dstFwd (after pQuantFwdTable)

```
0 1 1 0 0 0 0 0
1 0 0 0 0 0 0 0
1 0 0 0 0 0 0 0
1 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
```

## QuantInvTableInit\_JPEG

*Prepares quantization tables suitable for fast decoding.*

### Syntax

```
IppStatus ippiQuantInvTableInit_JPEG_8u16u(const Ipp8u* pQuantRawTable, Ipp16u*
pQuantInvTable);
```

### Parameters

<i>pQuantRawTable</i>	Pointer to the raw quantization table.
<i>pQuantInvTable</i>	Pointer to the quantization table for decoding.

## Description

The function `ippiQuantInvTableInit_JPEG` is declared in the `ippj.h` file. This function prepares a quantization table for fast decoding. It also performs reordering transformation that converts the zigzag sequence of table elements to conventional order (left-to-right, top-to-bottom) that is more suitable for a decoder. See [ISO10918], Annex A.3.6, *Zig-zag Sequence* for more information on zigzag order.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one or both of the specified pointers are <code>NULL</code> .

## QuantInv8x8\_JPEG

*Performs dequantization of an 8x8 block of DCT coefficients*

## Syntax

```
ippStatus ippiQuantInv8x8_JPEG_16s_C1I(Ipp16s* pSrcDst, const Ipp16u* pQuantInvTable);
```

## Parameters

<code>pSrcDst</code>	Pointer to the DCT coefficients of the 8x8 block.
<code>pQuantInvTable</code>	Pointer to the quantization table for decoding.

## Description

The function `ippiQuantInv8x8_JPEG` is declared in the `ippj.h` file. This function performs dequantization of the 8x8 block of DCT coefficients. Dequantization operation is defined in [ISO10918], Annex A.3.4, *DCT Coefficient Quantization and Dequantization*.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one or both of the specified pointers are <code>NULL</code> .

## Combined DCT Functions

This section describes functions that perform Discrete Cosine Transform (DCT) plus some additional operations - level shift, quantization (or dequantization) - and are specific for JPEG codec. These functions are listed in the table below.

### Functions that combine Quantization, DCT, and Level Shift

Function Base Name	Description
<code>DCTQuantFwd8x8_JPEG</code>	Performs forward DCT and quantization.
<code>DCTQuantFwd8x8LS_JPEG</code>	Performs forward DCT, quantization and level shift.
<code>DCTQuantInv8x8_JPEG</code>	Performs inverse DCT and dequantization.

Function Base Name	Description
<code>DCTQuantInv8x8LS_JPEG</code>	Performs inverse DCT, dequantization and level shift.
<code>DCTQuantInv8x8LS_1x1_JPEG</code> , <code>CTQuantInv8x8LS_2x2_JPEG</code> , <code>DC-</code> <code>TQuantInv8x8LS_4x4_JPEG</code>	Performs fast inverse DCT, dequantization and level shift for 8x8 block with zero high frequency components.
<code>DCTQuantInv8x8To4x4LS_JPEG</code> , <code>DC-</code> <code>TQuantInv8x8To2x2LS_JPEG</code>	Performs fast inverse DCT, dequantization, level shift and downsampling.

These functions implement component operations in accordance with their definition by [ISO10918], specifically:

- for DCT, as in [ISO10918], Annex A.3.3, *FDCT and IDCT*;
- for quantization/dequantization, as in [ISO10918], Annex A.3.4, *DCT Coefficient Quantization and Dequantization*;
- for level shift, as in [ISO10918], Annex A.3.1, *Level Shift*.



**NOTE.** The IPP functions described in this section use quantization tables in special format that is suitable for fast encoding/decoding. Initial raw quantization tables must be previously transformed using functions `ippiQuantFwdTableInit_JPEG` and `ippiQuantInvTableInit_JPEG`.

## DCTQuantFwd8x8\_JPEG

*Performs forward DCT and quantization of an 8x8 block.*

### Syntax

```
ippStatus ippiDCTQuantFwd8x8_JPEG_16s_C1(const Ipp16s* pSrc, Ipp16s* pDst, const Ipp16u*
pQuantFwdTable);
```

```
ippStatus ippiDCTQuantFwd8x8_JPEG_16s_C1I(Ipp16s* pSrcDst, const Ipp16u* pQuantFwdTable);
```

### Parameters

<code>pSrc</code>	Pointer to the 8x8 block in the source image.
<code>pDst</code>	Pointer to the destination 8x8 block of the quantized DCT coefficients.
<code>pSrcDst</code>	Pointer to the source and destination 8x8 block for in-place operation.
<code>pQuantFwdTable</code>	Pointer to the quantization table for the encoder.

### Description

The function `ippiDCTQuantFwd8x8_JPEG` is declared in the `ippj.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function performs a forward DCT and quantization of DCT coefficients.

The function uses a quantization table suitable for fast encoding. This table must be prepared using the function `ippiQuantFwdTableInit_JPEG`.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if any of the specified pointers is <code>NULL</code> .

## DCTQuantFwd8x8LS\_JPEG

*Performs level shift, forward DCT and quantization of an 8x8 block.*

---

## Syntax

```
IppStatus ippIDCTQuantFwd8x8LS_JPEG_8u16s_C1R(const Ipp8u* pSrc, int srcStep, Ipp16s* pDst, const Ipp16u* pQuantFwdTable);  
  
IppStatus ippIDCTQuantFwd8x8LS_JPEG_16u16s_C1R(const Ipp16u* pSrc, int srcStep, Ipp16s* pDst, const Ipp32f* pQuantFwdTable);
```

## Parameters

<code>pSrc</code>	Pointer to the 8x8 block in the source image.
<code>srcStep</code>	Distance in bytes between starts of consecutive lines in the source image.
<code>pDst</code>	Pointer to the destination 8x8 block of the quantized DCT coefficients.
<code>pQuantFwdTable</code>	Pointer to the quantization table for the encoder.

## Description

The function `ippIDCTQuantFwd8x8LS_JPEG` is declared in the `ippj.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function performs a forward DCT, quantization of DCT coefficients, and data conversion. Data from the unsigned `Ipp8u` range `[0..255]` are converted to the signed `Ipp16s` range `[-128..127]`.

The function uses a quantization table suitable for fast encoding. This table must be prepared using the function `ippiQuantFwdTableInit_JPEG`.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if any of the specified pointers is <code>NULL</code> .

**Example**

The code example below shows how to use the function `ippiDCTQuantFwd8x8LS_JPEG_8u16s_C1R`.

```
Ipp8u pQuantRawTable[64] = { 16, 11, 12, 14, 12, 10, 16, 14,
                             13, 14, 18, 17, 16, 19, 24, 40,
                             26, 24, 22, 22, 24, 49, 35, 37,
                             29, 40, 58, 51, 61, 60, 57, 51,
                             56, 55, 64, 72, 92, 78, 64, 68,
                             87, 69, 55, 56, 80, 109, 81, 87,
                             95, 98, 103, 104, 103, 62, 77, 113,
                             121, 112, 100, 120, 92, 101, 103, 99};
```

```
Ipp8u src[8*8] = { 4, 4, 4, 4, 4, 4, 4, 4,
                   4, 3, 3, 3, 3, 3, 3, 4,
                   4, 3, 2, 2, 2, 2, 3, 4,
                   4, 3, 2, 1, 1, 2, 3, 4,
                   4, 3, 2, 1, 1, 2, 3, 4,
                   4, 3, 2, 2, 2, 2, 3, 4,
                   4, 3, 3, 3, 3, 3, 3, 4,
                   4, 4, 4, 4, 4, 4, 4, 4};
```

```
Ipp16s dst[64];
```

```
Ipp16u pQuantFwdTable[64];
```

```
int quality = 75;
```

```
ippiQuantFwdRawTableInit_JPEG_8u ( pQuantRawTable, quality );
```

```
ippiQuantFwdTableInit_JPEG_8u16u ( pQuantRawTable, pQuantFwdTable );
```

```
ippiDCTQuantFwd8x8LS_JPEG_8u16s_C1R ( src, 8, dst, pQuantFwdTable );
```

result:

pQuantRawTable (with quality)

```
8 6 6 7 6 5 8 7
7 7 9 9 8 0 12 20
13 12 11 11 12 25 18 19
15 20 29 26 31 30 29 26
```

```

28 28 32 36 46 39 32 34
44 35 28 28 40 55 41 44
48 49 52 52 52 31 39 57
61 56 50 60 46 51 52 50

```

pQuantFwdTable

```

4096 5461 6554 4096 2731 1638 1260 1057
5461 5461 4681 3277 2521 1130 1092 1170
4681 4681 4096 2731 1638 1130 936 1170
4681 3641 2979 2185 1260 745 819 1057
3641 2979 1725 1170 964 596 630 840
2731 1820 1170 1024 799 630 575 712
1311 1024 840 745 630 537 546 643
910 712 683 669 585 655 630 655

```

dst

```

-125 0 1 0 0 0 0 0
0 0 0 0 0 0 0 0
1 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0

```

## DCTQuantInv8x8\_JPEG

*Performs dequantization and inverse DCT of an 8x8 block.*

### Syntax

```
IppStatus ippiDCTQuantInv8x8_JPEG_16s_C1(const Ipp16s* pSrc, Ipp16s* pDst, const Ipp16u*
pQuantInvTable);
```

```
IppStatus ippiDCTQuantInv8x8_JPEG_16s_C1I(Ipp16s* pSrcDst, const Ipp16u* pQuantInvTable);
```

### Parameters

*pSrc*                      Pointer to the 8x8 block of quantized DCT coefficients.

<i>pDst</i>	Pointer to the 8x8 block in the destination image.
<i>pSrcDst</i>	Pointer to the source and destination 8x8 block for in-place operation.
<i>pQuantInvTable</i>	Pointer to the quantization table for the decoder.

### Description

The function `ippiDCTQuantInv8x8_JPEG` is declared in the `ippj.h` file. This function performs an inverse DCT and dequantization of DCT coefficients.

The function uses a quantization table suitable for fast decoding. This table must be prepared using the function [ippiQuantInvTableInit\\_JPEG](#).

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition when any of the specified pointers is NULL.

## DCTQuantInv8x8LS\_JPEG

*Performs dequantization, inverse DCT and level shift of an 8x8 block.*

---

### Syntax

```
ippStatus ippiDCTQuantInv8x8LS_JPEG_16s8u_C1R(const Ipp16s* pSrc, Ipp8u* pDst, int dstStep,
const Ipp16u* pQuantInvTable);
```

```
ippStatus ippiDCTQuantInv8x8LS_JPEG_16s16u_C1R(const Ipp16s* pSrc, Ipp16u* pDst, int
dstStep, const Ipp32f* pQuantInvTable);
```

### Parameters

<i>pSrc</i>	Pointer to the 8x8 block of quantized DCT coefficients.
<i>pDst</i>	Pointer to the 8x8 block in the destination image.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>pQuantInvTable</i>	Pointer to the quantization table for the decoder.

### Description

The function `ippiDCTQuantInv8x8LS_JPEG` is declared in the `ippj.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function performs an inverse DCT, dequantization of DCT coefficients and data conversion. The function `ippiDCTQuantInv8x8LS_JPEG_16s8u_C1R` converts data from the signed `Ipp16s` range `[-128..127]` to the unsigned `Ipp8u` range `[0..255]`.

The function uses a quantization table suitable for fast decoding. This table must be prepared using the function [ippiQuantInvTableInit\\_JPEG](#).



## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if any of the specified pointers is <code>NULL</code> .

## DCTQuantInv8x8LS\_1x1\_JPEG, DCTQuantInv8x8LS\_2x2\_JPEG, DCTQuantInv8x8LS\_4x4\_JPEG

*Performs dequantization, inverse DCT and level shift of an 8x8 block with zero high frequency components.*

## Syntax

```
ippStatus ippIDCTQuantInv8x8LS_4x4_JPEG_16s8u_C1R(const Ipp16s* pSrc, Ipp8u* pDst, int
dstStep, const Ipp16u* pQuantInvTable);
```

```
ippStatus ippIDCTQuantInv8x8LS_2x2_JPEG_16s8u_C1R(const Ipp16s* pSrc, Ipp8u* pDst, int
dstStep, const Ipp16u* pQuantInvTable);
```

```
ippStatus ippIDCTQuantInv8x8LS_1x1_JPEG_16s8u_C1R(const Ipp16s* pSrc, Ipp8u* pDst, int
dstStep, const Ipp16u* pQuantInvTable);
```

## Parameters

<code>pSrc</code>	Pointer to the 8x8 block of quantized DCT coefficients.
<code>pDst</code>	Pointer to the 8x8 block in the destination image.
<code>dstStep</code>	Distance in bytes between starts of consecutive lines in the destination image.
<code>pQuantInvTable</code>	Pointer to the quantization table for the decoder.

## Description

These functions are declared in the `ippj.h` file. They operate with ROI (see [Regions of Interest in Intel IPP](#)).

These functions perform fast inverse DCT of an 8x8 block of coefficients containing non-zero components exclusively in the specified top-left part of the block. All coefficients outside the specified parts are equal to 0. The function `ippIDCTQuantInv8x8LS_4x4` operates with coefficients in the 4x4 top-left quadrant of the block, `ippIDCTQuantInv8x8LS_2x2` operates with coefficients in the 2x2 top-left part of the block, `ippIDCTQuantInv8x8LS_1x1` operates with only DC non-zero coefficient. All functions perform dequantization of DCT coefficients and data conversion. The functions convert data from the signed `Ipp16s` range `[-128..127]` to the unsigned `Ipp8u` range `[0..255]`.

The function uses a quantization table suitable for fast decoding. This table must be prepared using the function `ippiQuantInvTableInit_JPEG`.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if any of the specified pointers is <code>NULL</code> .

## DCTQuantInv8x8To4x4LS\_JPEG, DCTQuantInv8x8To2x2LS\_JPEG

*Performs dequantization, inverse DCT, level shift, and downsampling of an 8x8 block.*

### Syntax

```
IppStatus ippiDCTQuantInv8x8To4x4LS_JPEG_16s8u_C1R(const Ipp16s* pSrc, Ipp8u* pDst, int
dstStep, const Ipp16u* pQuantInvTable);
```

```
IppStatus ippiDCTQuantInv8x8To2x2LS_JPEG_16s8u_C1R(const Ipp16s* pSrc, Ipp8u* pDst, int
dstStep, const Ipp16u* pQuantInvTable);
```

### Parameters

<i>pSrc</i>	Pointer to the 8x8 block of quantized DCT coefficients.
<i>pDst</i>	Pointer to the block in the destination image.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>pQuantInvTable</i>	Pointer to the quantization table for the decoder.

### Description

These functions are declared in the `ippj.h` file. They operate with ROI (see [Regions of Interest in Intel IPP](#)).

These functions perform an inverse DCT, dequantization of DCT coefficients, downsampling and data conversion. The function `ippiDCTQuantInv8x8To4x4LS_JPEG` performs downsampling with factor 1:2, that is the output block contains 4x4 elements. The function `ippiDCTQuantInv8x8To2x2LS_JPEG` performs downsampling with factor 1:4, that is the output block contains 2x2 elements.

The functions convert data from the signed `Ipp16s` range `[-128..127]` to the unsigned `Ipp8u` range `[0..255]`.

The function uses a quantization table suitable for fast decoding. This table must be prepared using the function `ippiQuantInvTableInit_JPEG`.

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if any of the specified pointers is <code>NULL</code> .

## Level Shift Functions

This section describes the Intel IPP functions that perform level shift operation and are specific for JPEG codec. These functions are listed in the table below.

### Level Shift Functions

Function Base Name	Description
<code>Sub128_JPEG</code>	Performs level shift to the signed representation.
<code>Add128_JPEG</code>	Performs level shift to the unsigned representation.

The level shift operation is implemented in accordance with its definition in [\[ISO10918\]](#), Annex A.3.1, *Level Shift*.

## Sub128\_JPEG

*Converts data from the unsigned 8u range to the signed 16s range.*

---

### Syntax

```
IppStatus ippiSub128_JPEG_8u16s_C1R(const Ipp8u* pSrc, int srcStep, Ipp16s* pDst, int dstStep, IppiSize roiSize);
```

### Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>roiSize</i>	Size of the source and destination ROI in pixels.

### Description

The function `ippiSub128_JPEG` is declared in the `ippj.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts every sample of the source image ROI from the unsigned `Ipp8u` range `[0..255]` to the signed `Ipp16s` range `[-128..127]`.

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> or <i>dstStep</i> has a zero or negative value.

## Add128\_JPEG

*Restores samples to the unsigned representation.*

---

### Syntax

```
IppStatus ippiAdd128_JPEG_16s8u_C1R(const Ipp16s* pSrc, int srcStep, Ipp8u* pDst, int dstStep, IppiSize roiSize);
```

### Parameters

<i>pSrc</i>	Pointer to the source image ROI.
-------------	----------------------------------

<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>roiSize</i>	Size of the source and destination ROI in pixels.

## Description

The function `ippiAdd128_JPEG` is declared in the `ippj.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts every sample of source image from the signed `Ipp16s` range `[-128..127]` to the unsigned `Ipp8u` range `[0..255]`.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one or both of the specified pointers are <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> or <i>dstStep</i> has a zero or negative value.

## Sampling Functions

This section describes Intel IPP functions that perform sampling and are specific for JPEG codec. These functions are listed in the table below.

### Sampling Functions

Function Base Name	Description
<a href="#">SampleDownH2V1_JPEG</a>	Performs 2:1 horizontal sampling and 1:1 vertical sampling of an image.
<a href="#">SampleDownH2V2_JPEG</a>	Performs 2:1 horizontal sampling and 2:1 vertical sampling of an image.
<a href="#">SampleDownRowH2V1_Box_JPEG</a>	Performs 2:1 horizontal sampling and 1:1 vertical sampling of an image row.
<a href="#">SampleDownRowH2V2_Box_JPEG</a>	Performs 2:1 horizontal sampling and 2:1 vertical sampling of an image row.
<a href="#">SampleUpH2V1_JPEG</a>	Performs 1:2 horizontal sampling and 1:1 vertical sampling of an image.
<a href="#">SampleUpH2V2_JPEG</a>	Performs 1:2 horizontal sampling and 1:2 vertical sampling of an image.
<a href="#">SampleUpRowH2V1_Triangle_JPEG</a>	Performs 1:2 horizontal sampling and 1:1 vertical sampling of an image row.
<a href="#">SampleUpRowH2V2_Triangle_JPEG</a>	Performs 1:2 horizontal sampling and 1:2 vertical sampling of an image row.
<a href="#">SampleDown444LS_MCU</a>	Creates 444 MCU with level shift
<a href="#">SampleDown422LS_MCU</a>	Creates 422 MCU with level shift

Function Base Name	Description
<a href="#">SampleDown411LS_MCU</a>	Creates 411 MCU with level shift
<a href="#">SampleUp444LS_MCU</a>	Creates pixel-order image from 444 MCU with level shift
<a href="#">SampleUp422LS_MCU</a>	Creates pixel-order image from 422 MCU with level shift
<a href="#">SampleUp411LS_MCU</a>	Creates pixel-order image from 411 MCU with level shift

## SampleDownH2V1\_JPEG

*Performs 2:1 horizontal sampling and 1:1 vertical sampling of an image.*

### Syntax

```
IppStatus ippiSampleDownH2V1_JPEG_8u_C1R(const Ipp8u* pSrc, int srcStep, IppiSize srcSize,
Ipp8u* pDst, int dstStep, IppiSize dstSize);
```

### Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>srcSize</i>	Size of the source ROI in pixels.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>dstSize</i>	Size of the destination ROI in pixels.

### Description

The function `ippiSampleDownH2V1_JPEG` is declared in the `ippj.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function performs 2:1 horizontal sampling and 1:1 vertical sampling of an image. Downsampling is performed as a simple “box” filter.

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one or both of the specified pointers are NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> or <i>dstStep</i> has a zero or negative value

## SampleDownH2V2\_JPEG

*Performs 2:1 horizontal sampling and 2:1 vertical sampling of an image.*

---

### Syntax

```
IppStatus ippISampleDownH2V2_JPEG_8u_C1R(const Ipp8u* pSrc, int srcStep, IppiSize srcSize,
Ipp8u* pDst, int dstStep, IppiSize dstSize);
```

### Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>srcSize</i>	Size of the source ROI in pixels.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>dstSize</i>	Size of the destination ROI in pixels.

### Description

The function `ippISampleDownH2V2_JPEG` is declared in the `ippj.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#) ).

This function performs 2:1 horizontal sampling and 2:1 vertical sampling of an image. Downsampling is performed as a simple “box” filter.

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one or both of the specified pointers are <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> or <i>dstStep</i> has a zero or negative value.

## SampleDownRowH2V1\_Box\_JPEG

*Performs 2:1 horizontal sampling and 1:1 vertical sampling of an image row.*

---

### Syntax

```
IppStatus ippISampleDownRowH2V1_Box_JPEG_8u_C1(const Ipp8u* pSrc, int srcWidth, Ipp8u*
pDst);
```

## Parameters

<i>pSrc</i>	Pointer to the source image row.
<i>pDst</i>	Pointer to the destination image row.
<i>srcWidth</i>	Width of the source row in pixels.

## Description

The function `ippiSampleDownRowH2V1_Box_JPEG` is declared in the `ippj.h` file. This function performs 2:1 horizontal sampling and 1:1 vertical sampling of an image row. Downsampling is performed as a simple “box” filter.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one or both of the specified pointers are <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <i>srcWidth</i> has a zero or negative value.

## Example

The code example below shows how to use the function `ippiSampleDownRowH2V1_Box_JPEG_8u_C1`.

```
void func_sampling()
{
    Ipp8u pSrc[8*8];
    IppiSize srcSize = {8, 8};
    int srcStep = 8*sizeof(Ipp8u);
    int srcWidth = 8;

    Ipp8u pDst[8*8];

    ippiImageJaehne_8u_C1R( pSrc, srcStep, srcSize);

    for(int r = 0; r < 8; r++)
    {
        ippiSampleDownRowH2V1_Box_JPEG_8u_C1( &pSrc[8*r], srcWidth, &pDst[4*r]);
    }
}
```

Result:

pSrc (8x8)								pDst (8x4)			
0	67	165	209	209	165	67	0	34	187	187	33
67	209	255	250	250	255	209	67	138	252	253	138
165	255	226	188	188	226	255	165	210	207	207	210
209	250	188	140	140	188	250	209	230	164	164	229
209	250	188	140	140	188	250	209	230	164	164	229
165	255	226	188	188	226	255	165	210	207	207	210
67	209	255	250	250	255	209	67	138	252	253	138
0	67	165	209	209	165	67	0	34	187	187	33



## SampleDownRowH2V2\_Box\_JPEG

*Performs 2:1 horizontal sampling and 2:1 vertical sampling of an image row.*

---

### Syntax

```
IppStatus ippiSampleDownRowH2V2_Box_JPEG_8u_C1(const Ipp8u* pSrc1, const Ipp8u* pSrc2,
int srcWidth, Ipp8u* pDst);
```

### Parameters

<i>pSrc1</i>	Pointer to the source image row.
<i>pSrc2</i>	Pointer to the subsequent source image row.
<i>pDst</i>	Pointer to the destination image row.
<i>srcWidth</i>	Width of the source row in pixels.

### Description

The function `ippiSampleDownRowH2V2_Box_JPEG` is declared in the `ippj.h` file. This function performs 2:1 horizontal sampling and 2:1 vertical sampling of an image row. Downsampling is performed as a simple “box” filter.

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one or all of the specified pointers are <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <i>srcWidth</i> has a zero or negative value.

## SampleUpH2V1\_JPEG

*Performs 1:2 horizontal sampling and 1:1 vertical sampling of an image.*

---

### Syntax

```
IppStatus ippiSampleUpH2V1_JPEG_8u_C1R(const Ipp8u* pSrc, int srcStep, IppiSize srcSize,
Ipp8u* pDst, int dstStep, IppiSize dstSize);
```

### Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>srcSize</i>	Size of the source ROI in pixels.
<i>pDst</i>	Pointer to the destination image ROI.

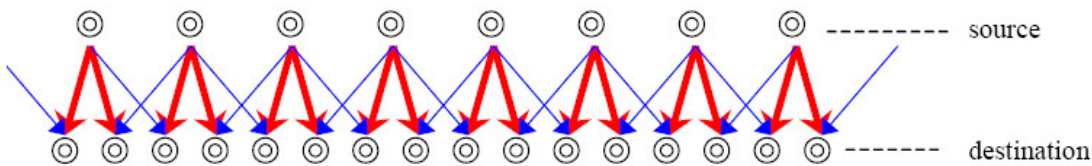
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>dstSize</i>	Size of the destination ROI in pixels.

### Description

The function `ippiSampleUpH2V1_JPEG` is declared in the `ippj.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function performs 1:2 horizontal sampling and 1:1 vertical sampling of an image. Sampling is performed following the scheme of a “triangle” filter as shown in [Figure “Triangle Sampling Scheme”](#):

### Triangle Sampling Scheme



Here thick red lines denote source sample weight  $\frac{3}{4}$  and thin blue lines denote source sample weight  $\frac{1}{4}$  in their contribution to the destination sample value.



**NOTE.** This function requires that some samples outside the source image ROI are defined. Specifically, the left-most column must be present and the right-most column may be required.

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one or both of the specified pointers are NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> or <i>dstStep</i> has a zero or negative value

## SampleUpH2V2\_JPEG

*Performs 1:2 horizontal sampling and 1:2 vertical sampling of an image.*

---

### Syntax

```
IppStatus ippISampleUpH2V2_JPEG_8u_C1R(const Ipp8u* pSrc, int srcStep, IppiSize srcSize, Ipp8u* pDst, int dstStep, IppiSize dstSize);
```

### Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>srcSize</i>	Size of the source ROI in pixels.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>dstSize</i>	Size of the destination ROI in pixels.

### Description

The function `ippISampleUpH2V2_JPEG` is declared in the `ippj.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function performs 1:2 horizontal sampling and 1:2 vertical sampling of an image. Sampling is performed following the scheme of a “triangle” filter as shown in [Figure “Triangle Sampling Scheme”](#).



---

**NOTE.** This function requires that some samples outside the source image ROI are defined. Specifically, the left-most column and the top-most row must be present, whereas the right-most column and the bottom-most row may be required.

---

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one or both of the specified pointers are NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> or <i>dstStep</i> has a zero or negative value.

## SampleUpRowH2V1\_Triangle\_JPEG

*Performs 1:2 horizontal sampling and 1:1 vertical sampling of an image row.*

---

### Syntax

```
IppStatus ippiSampleUpRowH2V1_Triangle_JPEG_8u_C1(const Ipp8u* pSrc, int srcWidth, Ipp8u* pDst);
```

### Parameters

<i>pSrc</i>	Pointer to the source image row.
<i>pDst</i>	Pointer to the destination image row.
<i>srcWidth</i>	Width of the source row in pixels.

### Description

The function `ippiSampleUpRowH2V1_Triangle_JPEG` is declared in the `ippj.h` file. This function performs 1:2 horizontal sampling and 1:1 vertical sampling of an image row. Sampling is performed following the scheme of a “triangle” filter as shown in [Figure “Triangle Sampling Scheme”](#). This function does not require samples outside the source image row.

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one or both of the specified pointers are NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>srcWidth</i> has a zero or negative value.

## SampleUpRowH2V2\_Triangle\_JPEG

*Performs 1:2 horizontal sampling and 1:2 vertical sampling of an image row.*

---

### Syntax

```
IppStatus ippiSampleUpRowH2V2_Triangle_JPEG_8u_C1(const Ipp8u* pSrc1, const Ipp8u* pSrc2, int srcWidth, Ipp8u* pDst);
```

### Parameters

<i>pSrc1</i>	Pointer to the source image row.
<i>pSrc2</i>	Pointer to the subsequent source image row.
<i>pDst</i>	Pointer to the destination image row.
<i>srcWidth</i>	Width of the source row in pixels.

## Description

The function `ippiSampleUpRowH2V2_Triangle_JPEG` is declared in the `ippj.h` file. This function performs 1:2 horizontal sampling and 1:2 vertical sampling of an image row. Sampling is performed following the scheme of a “triangle” filter as shown in [Figure “Triangle Sampling Scheme”](#). This function does not require samples outside the source image rows.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one or all specified pointers are NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <code>srcWidth</code> has a zero or negative value.

## SampleDown444LS\_MCU

Creates 444 MCU with level shift from pixel-order data.

## Syntax

```
ippStatus ippiSampleDown444LS_MCU_8u16s_C3P3R(const Ipp8u* pSrc, int srcStep, Ipp16s* pDstMCU[3]);
```

## Parameters

<code>pSrc</code>	Pointer to the source image ROI.
<code>srcStep</code>	Distance in bytes between starts of consecutive lines in the source image.
<code>pDstMCU</code>	Array of 3 pointers to the destination image blocks.

## Description

The function `ippiSampleDown444LS_MCU` is declared in the `ippj.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts interleaved data to the 444 MCU that consists of three 8x8 blocks - one block for each channel (see [Figure “MCU Structure for Different JPEG Sampling Formats”](#)).

Additionally, this function converts data from unsigned `Ipp8u` range [0..255] to the signed `Ipp16s` range [-128..127] performing level shift operation (by subtracting 128).

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if any of the specified pointers are NULL.
<code>ippStsStepErr</code>	Indicates an error condition if <code>srcStep</code> has a zero or negative value.

## SampleDown422LS\_MCU

*Creates 422 MCU with level shift from pixel-order data.*

### Syntax

```
IppStatus ippiSampleDown422LS_MCU_8u16s_C3P3R(const Ipp8u* pSrc, int srcStep, Ipp16s* pDstMCU[3]);
```

### Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDstMCU</i>	Pointer to the array of 3 pointers to the destination blocks.

### Description

The function `ippiSampleDown422LS_MCU` is declared in the `ippj.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts interleaved data to the full MCU with 4:2:2 downsampling. Thus, final MCU consists of four blocks - two 8x8 blocks for channel 1, one 8x8 block for channel 2, and one 8x8 block for channel 3 (see [Figure "MCU Structure for Different JPEG Sampling Formats"](#)).

Downsampling is performed by averaging the corresponding pixel values.

Additionally, this function converts data from unsigned `Ipp8u` range `[0..255]` to the signed `Ipp16s` range `[-128..127]` performing level shift operation (by subtracting 128).

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if any of the specified pointers are NULL.
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> has a zero or negative value.

## SampleDown411LS\_MCU

*Creates 411 MCU with level shift from pixel-order data.*

### Syntax

```
IppStatus ippiSampleDown411LS_MCU_8u16s_C3P3R(const Ipp8u* pSrc, int srcStep, Ipp16s* pDstMCU[3]);
```

### Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.

*pDstMCU*

Pointer to the array of 3 pointers to the destination blocks.

## Description

The function `ippiSampleDown411LS_MCU` is declared in the `ippj.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts interleaved data to the full MCU with 4:1:1 downsampling. Thus, final MCU consists of six blocks - four 8x8 blocks for channel 1, one 8x8 block for channel 2, and one 8x8 block for channel 3 (see [Figure "MCU Structure for Different JPEG Sampling Formats"](#)).

Downsampling is performed by averaging the corresponding pixel values.

Additionally, this function converts data from unsigned `Ipp8u` range `[0..255]` to the signed `Ipp16s` range `[-128..127]` performing level shift operation (by subtracting 128).

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if any of the specified pointers are NULL.
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> has a zero or negative value.

## SampleUp444LS\_MCU

*Creates pixel-order image from 444 MCU and performs level shift.*

## Syntax

```
IppStatus ippiSampleUp444LS_MCU_16s8u_P3C3R(const Ipp16s* pSrcMCU[3], Ipp8u* pDst, int dstStep);
```

## Parameters

<i>pSrcMCU</i>	Pointer to the array of 3 pointers to the source blocks.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.

## Description

The function `ippiSampleUp444LS_MCU` is declared in the `ippj.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function creates pixel-order image from 444 MCU (see [Figure "MCU Structure for Different JPEG Sampling Formats"](#)).

Additionally, this function converts data from the signed `Ipp16s` range `[-128..127]` to the unsigned `Ipp8u` range `[0..255]` performing level shift operation (by adding 128).

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if any of the specified pointers is <code>NULL</code> .
<code>ippStsStepErr</code>	Indicates an error condition if <code>dstStep</code> has a zero or negative value.

## SampleUp422LS\_MCU

*Creates pixel-order image from 422 MCU and performs level shift.*

---

### Syntax

```
IppStatus ippISampleUp422LS_MCU_16s8u_P3C3R(const Ipp16s* pSrcMCU[3], Ipp8u* pDst, int dstStep);
```

### Parameters

<code>pSrcMCU</code>	Pointer to the array of 3 pointers to the source blocks.
<code>pDst</code>	Pointer to the destination image ROI.
<code>dstStep</code>	Distance in bytes between starts of consecutive lines in the destination image.

### Description

The function `ippISampleUp422LS_MCU` is declared in the `ippj.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function creates pixel-order image from 422 MCU (see [Figure "MCU Structure for Different JPEG Sampling Formats"](#)). Upsampling is performed as a simple box filter.

Additionally, this function converts data from the signed `Ipp16s` range `[-128..127]` to the unsigned `Ipp8u` range `[0..255]` performing level shift operation (by adding 128).

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if any of the specified pointers is <code>NULL</code> .
<code>ippStsStepErr</code>	Indicates an error condition if <code>dstStep</code> has a zero or negative value.

## SampleUp411LS\_MCU

*Creates pixel-order image from 411 MCU and performs level shift.*

---

### Syntax

```
IppStatus ippISampleUp411LS_MCU_16s8u_P3C3R(const Ipp16s* pSrcMCU[3], Ipp8u* pDst, int dstStep);
```



## Parameters

<i>pSrcMCU</i>	Pointer to the array of 3 pointers to the source blocks.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.

## Description

The function `ippiSampleUp411LS_MCU` is declared in the `ippj.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function creates pixel-order image from 411 MCU (see [Figure "MCU Structure for Different JPEG Sampling Formats"](#)). Upsampling is performed as a simple box filter.

Additionally, this function converts data from the signed `Ipp16s` range `[-128..127]` to the unsigned `Ipp8u` range `[0..255]` performing level shift operation (by adding 128).

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if any of the specified pointers is <code>NULL</code> .
<code>ippStsStepErr</code>	Indicates an error condition if <i>dstStep</i> has a zero or negative value.

# Planar-to-Pixel and Pixel-to-Planar Conversion Functions

This section describes the functions that convert planar (not-interleaved) data to pixel-order (interleaved) data and vice versa. [Table "Pixel-to-Planar and Planar-to-Pixel Conversion Functions"](#) lists these functions described in more detail later in this section.

## Pixel-to-Planar and Planar-to-Pixel Conversion Functions

Function Base Name	Description
<a href="#">Split422LS_MCU</a>	Creates 422 MCU from 422 interleaved data with level shift.
<a href="#">Join422LS_MCU</a>	Creates 422 interleaved data from 422 MCU with level shift.

## Split422LS\_MCU

*Creates 422 MCU from 422 interleaved data with level shift.*

## Syntax

```
IppStatus ippiSplit422LS_MCU_8u16s_C2P3R(const Ipp8u* pSrc, int SrcStep, Ipp16s* pDstMCU[3]);
```

## Parameters

<i>pSrc</i>	Pointer to the source image ROI.
-------------	----------------------------------

<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDstMCU</i>	Pointer to the array of 3 pointers to the destination blocks.

### Description

The function `ippiSplit422LS_MCU` is declared in the `ippj.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function creates 422 MCU from 422 pixel-order data (see [Figure "Plane Size and Layout - 3-planes Images"](#)), that is, converts pixel-order data to planar form without re-sampling. Final MCU consists of four blocks - two 8x8 blocks for channel 1, one 8x8 block for channel 2, and one 8x8 block for channel 3 (see [Figure "MCU Structure for Different JPEG Sampling Formats"](#)).

Additionally, this function converts data from unsigned `Ipp8u` range `[0..255]` to the signed `Ipp16s` range `[-128..127]` performing level shift operation (by subtracting 128).

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if any of the specified pointers are NULL.
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> has a zero or negative value.

## Join422LS\_MCU

*Creates 422 interleaved data from 422 MCU with level shift.*

---

### Syntax

```
IppStatus ippiJoin422LS_MCU_16s8u_P3C2R(const Ipp16s* pSrcMCU[3], Ipp8u* pDst, int dstStep);
```

### Parameters

<i>pSrcMCU</i>	Pointer to the array of 3 pointers to the source blocks.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.

### Description

The function `ippiJoin422LS_MCU` is declared in the `ippj.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function creates 422 interleaved data (see [Table "Pixel-order Image Formats"](#)) from 422 MCU (see [Figure "MCU Structure for Different JPEG Sampling Formats"](#)), that is, converts planar data to pixel-order form without re-sampling. Output data is a repetitive sequence of pixels Ch1Ch2Ch1Ch3...

Additionally, this function converts data from the signed `Ipp16s` range `[-128..127]` to the unsigned `Ipp8u` range `[0..255]` performing level shift operation (by adding 128).

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if any of the specified pointers are NULL.
<code>ippStsStepErr</code>	Indicates an error condition if <i>dstStep</i> has a zero or negative value.

## Huffman Codec Functions

This section describes Huffman encoding and decoding functions that are specific for JPEG codec. These functions are listed in the table below.

### Huffman Codec Functions

Function Base Name	Description
<b>Encoder Functions</b>	
<code>EncodeHuffmanRawTableInit_JPEG</code>	Creates raw Huffman tables using Huffman symbols statistics.
<code>EncodeHuffmanSpecGetBufSize_JPEG</code>	Returns the length of the <code>IppiEncodeHuffmanSpec</code> structure.
<code>EncodeHuffmanSpecInit_JPEG</code>	Creates Huffman table in format that is suitable for an encoder.
<code>EncodeHuffmanSpecInitAlloc_JPEG</code>	Allocates memory and creates Huffman table in format that is suitable for an encoder.
<code>EncodeHuffmanSpecFree_JPEG</code>	Frees memory allocated by the <code>ippiEncodeHuffmanSpecInitAlloc_JPEG_8u</code> function.
<code>EncodeHuffmanStateGetBufSize_JPEG</code>	Returns the length of the <code>IppiEncodeHuffmanState</code> structure.
<code>EncodeHuffmanStateInit_JPEG</code>	Initializes the <code>IppiEncodeHuffmanState</code> structure.
<code>EncodeHuffmanStateInitAlloc_JPEG</code>	Allocates memory and initializes the <code>IppiEncodeHuffmanState</code> structure.
<code>EncodeHuffmanStateFree_JPEG</code>	Frees memory allocated by the <code>ippiEncodeHuffmanStateInitAlloc_JPEG</code> function.
<code>EncodeHuffman8x8_JPEG</code>	Performs Huffman baseline encoding of an 8x8 block of quantized DCT coefficients.
<code>EncodeHuffman8x8_Direct_JPEG</code>	Directly performs Huffman baseline encoding of an 8x8 block of quantized DCT coefficients.
<code>GetHuffmanStatistics8x8_JPEG</code>	Computes Huffman symbols statistics for the baseline encoding.
<code>GetHuffmanStatistics8x8_DCFirst_JPEG</code>	Computes Huffman symbols statistics for the progressive encoding (DC coefficient).
<code>GetHuffmanStatistics8x8_ACFirst_JPEG</code>	Computes Huffman symbols statistics for the progressive encoding (AC coefficients, the first scan).
<code>GetHuffmanStatistics8x8_ACRrefine_JPEG</code>	Computes Huffman symbols statistics for the progressive encoding (AC coefficients, subsequent scans).
<code>EncodeHuffman8x8_DCFirst_JPEG</code>	Performs progressive encoding of the DC coefficient from an 8x8 block of the quantized DCT coefficients (first scan).
<code>EncodeHuffman8x8_DCRefine_JPEG</code>	Performs progressive encoding of the DC coefficient from an 8x8 block of the quantized DCT coefficients (subsequent scans).
<code>EncodeHuffman8x8_ACFirst_JPEG</code>	Performs progressive encoding of the AC coefficients from an 8x8 block of the quantized DCT coefficients (first scan).

Function Base Name	Description
<code>EncodeHuffman8x8_ACRefine_JPEG</code>	Performs progressive encoding of the AC coefficients from an 8x8 block of the quantized DCT coefficients (subsequent scans).
<b>Decoder Functions</b>	
<code>DecodeHuffmanSpecGetBufSize_JPEG</code>	Returns the length of the <code>IppiDecodeHuffmanSpec</code> structure.
<code>DecodeHuffmanSpecInit_JPEG</code>	Creates Huffman table in format that is suitable for a decoder.
<code>DecodeHuffmanSpecInitAlloc_JPEG</code>	Allocates memory and creates Huffman table in format that is suitable for a decoder.
<code>DecodeHuffmanSpecFree_JPEG</code>	Frees memory allocated by the <code>ippiDecodeHuffmanSpecInitAlloc_JPEG</code> function.
<code>DecodeHuffmanStateGetBufSize_JPEG</code>	Returns the length of the <code>IppiDecodeHuffmanState</code> structure.
<code>DecodeHuffmanStateInit_JPEG</code>	Initializes the <code>IppiDecodeHuffmanState</code> structure.
<code>DecodeHuffmanStateInitAlloc_JPEG</code>	Allocates memory and initializes the <code>IppiDecodeHuffmanState</code> structure.
<code>DecodeHuffmanStateFree_JPEG</code>	Frees memory allocated by the <code>ippiDecodeHuffmanStateInitAlloc_JPEG</code> function.
<code>DecodeHuffman8x8_JPEG</code>	Performs Huffman baseline decoding of 8x8 block of the quantized DCT coefficients.
<code>DecodeHuffman8x8_Direct_JPEG</code>	Directly performs Huffman baseline decoding of 8x8 block of the quantized DCT coefficients.
<code>DecodeHuffman8x8_DCFirst_JPEG</code>	Performs progressive decoding of the DC coefficient from an 8x8 block of the quantized DCT coefficients (first scan).
<code>DecodeHuffman8x8_DCRefine_JPEG</code>	Performs progressive decoding of the DC coefficient from an 8x8 block of the quantized DCT coefficients (subsequent scans).
<code>DecodeHuffman8x8_ACFirst_JPEG</code>	Performs progressive decoding of the AC coefficients from an 8x8 block of the quantized DCT coefficients (first scan).
<code>DecodeHuffman8x8_ACRefine_JPEG</code>	Performs progressive decoding of the AC coefficients from an 8x8 block of the quantized DCT coefficients (subsequent scans).

## EncodeHuffmanRawTableInit\_JPEG

*Creates raw Huffman tables using Huffman symbols statistics.*

### Syntax

```
IppStatus ippiEncodeHuffmanRawTableInit_JPEG_8u(const int pStatistics[256], Ipp8u* pListBits, Ipp8u* pListVals);
```

### Parameters

<code>pStatistics</code>	Pointer to the buffer that contains Huffman symbol statistics.
<code>pListBits</code>	Pointer to the <i>Bits</i> list.
<code>pListVals</code>	Pointer to the <i>Vals</i> list.

## Description

The function `ippiEncodeHuffmanRawTableInit_JPEG` is declared in the `ippj.h` file. This function builds raw Huffman tables using the previously computed Huffman symbols statistics. An array for statistics is indexed with Huffman symbol value. The values in the array are frequencies of a given symbol.

The *Bits* and *Vals* lists are specified in [ISO10918], Annex B, Figure B.7 .

To generate *Bits* and *Vals* lists, the function uses the procedure that is described in [ISO10918], Annex K.2, *A Procedure for Generating the Lists, Which Specify a Huffman Code Table*.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one or all specified pointers are NULL.

## EncodeHuffmanSpecGetBufSize\_JPEG

Returns the length of the `IppiEncodeHuffmanSpec` structure.

---

## Syntax

```
IppStatus ippiEncodeHuffmanSpecGetBufSize_JPEG_8u(int* pSize);
```

## Parameters

<i>pSize</i>	Pointer to a variable that will receive the length of the <code>IppiEncodeHuffmanSpec</code> structure.
--------------	---

## Description

The function `ippiEncodeHuffmanSpecGetBufSize_JPEG` is declared in the `ippj.h` file. This function computes the length of the `IppiEncodeHuffmanSpec` structure.



**NOTE.** The fields of the structure `IppiEncodeHuffmanSpec` are hidden from the user.

---

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if the pointer is NULL.

## EncodeHuffmanSpecInit\_JPEG

*Creates Huffman table in a format that is suitable for an encoder.*

---

### Syntax

```
IppStatus ippiEncodeHuffmanSpecInit_JPEG_8u(const Ipp8u* pListBits, const Ipp8u* pListVals,
IppiEncodeHuffmanSpec* pEncHuffSpec);
```

### Parameters

<i>pListBits</i>	Pointer to <i>Bits</i> list.
<i>pListVals</i>	Pointer to <i>Vals</i> list.
<i>pEncHuffSpec</i>	Pointer to the Huffman table for the encoder.

### Description

The function `ippiEncodeHuffmanSpecInit_JPEG` is declared in the `ippj.h` file. This function creates Huffman table in a format that is suitable for the encoder. The *Bits* and *Vals* lists are specified in [ISO10918], Annex C, Huffman table specification. To generate the table, the function uses the procedure that is described in [ISO10918], Annex C.2, Conversion of Huffman Tables Specified in Interchange Format to Tables of Codes and Code Lengths.

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one or all specified pointers are NULL.
<code>ippStsJPEGHuffTableErr</code>	Indicates an error condition during the initialization of the table. It may happen when raw tables contain inadmissible values.

## EncodeHuffmanSpecInitAlloc\_JPEG

*Allocates memory and creates Huffman table in a format that is suitable for an encoder.*

---

### Syntax

```
IppStatus ippiEncodeHuffmanSpecInitAlloc_JPEG_8u(const Ipp8u* pListBits, const Ipp8u*
pListVals, IppiEncodeHuffmanSpec** ppEncHuffSpec);
```

### Parameters

<i>pListBits</i>	Pointer to the <i>Bits</i> list.
<i>pListVals</i>	Pointer to the <i>Vals</i> list.
<i>ppEncHuffSpec</i>	Pointer to the returned pointer to the Huffman table for the coder.

## Description

The function `ippiEncodeHuffmanSpecInitAlloc_JPEG` is declared in the `ippj.h` file. This function allocates memory and creates Huffman table in a format that is suitable for the encoder. The *Bits* and *Vals* lists are specified in [ISO10918], Annex C, *Huffman table specification*. To generate the table, the function uses the procedure that is described in [ISO10918], Annex C.2, *Conversion of Huffman Tables Specified in Interchange Format to Tables of Codes and Code Lengths*.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one or all specified pointers are NULL.
<code>ippStsJPEGHuffTableErr</code>	Indicates an error condition when the table is initialized. It may happen when raw tables contain inadmissible values.

## EncodeHuffmanSpecFree\_JPEG

*Frees memory allocated by the `ippiEncodeHuffmanSpecInitAlloc_JPEG_8u` function.*

## Syntax

```
IppStatus ippiEncodeHuffmanSpecFree_JPEG_8u (IppiEncodeHuffmanSpec* pEncHuffSpec);
```

## Parameters

*pEncHuffSpec*                      Pointer to the Huffman table for the encoder.

## Description

The function `ippiEncodeHuffmanSpecFree_JPEG` is declared in the `ippj.h` file. This function frees memory allocated by the `ippiEncodeHuffmanSpecInitAlloc_JPEG` function for the Huffman table.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
--------------------------	---------------------

## EncodeHuffmanStateGetBufSize\_JPEG

*Returns the length of the `IppiEncodeHuffmanState` structure.*

## Syntax

```
IppStatus ippiEncodeHuffmanStateGetBufSize_JPEG_8u(int* pSize);
```

## Parameters

*pSize*                              Pointer to a variable that will receive the length of the `IppiEncodeHuffmanState` structure.

## Description

The function `ippiEncodeHuffmanStateGetBufSize_JPEG` is declared in the `ippj.h` file. This function computes the length (in bytes) of the `IppiEncodeHuffmanState` structure.



**NOTE.** The fields of the structure `IppiEncodeHuffmanState` are hidden from the user.

---

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if specified pointer is <code>NULL</code> .

## EncodeHuffmanStateInit\_JPEG

*Initializes the `IppiEncodeHuffmanState` structure.*

---

### Syntax

```
IppStatus ippiEncodeHuffmanStateInit_JPEG_8u( IppiEncodeHuffmanState* pEncHuffState);
```

### Parameters

<code>pEncHuffState</code>	Pointer to the <code>IppiEncodeHuffmanState</code> structure.
----------------------------	---

### Description

The function `ippiEncodeHuffmanStateInit_JPEG` is declared in the `ippj.h` file. This function initializes the `IppiEncodeHuffmanState` to the initial state. This function must be called prior to the JPEG data stream encoding.

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one or all of the specified pointers are <code>NULL</code> .

## EncodeHuffmanStateInitAlloc\_JPEG

*Allocates memory and initializes the `IppiEncodeHuffmanState` structure.*

---

### Syntax

```
IppStatus ippiEncodeHuffmanStateInitAlloc_JPEG_8u(IppiEncodeHuffmanState** ppEncHuffState);
```

### Parameters

<code>ppEncHuffState</code>	Pointer to the <code>IppiEncodeHuffmanState</code> structure.
-----------------------------	---



## Description

The function `ippiEncodeHuffmanStateInitAlloc_JPEG` is declared in the `ippj.h` file. This function allocates memory and initializes the `IppiEncodeHuffmanState` structure to the initial state. This function must be called prior to the JPEG data stream encoding.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one or all of the specified pointers are <code>NULL</code> .

## EncodeHuffmanStateFree\_JPEG

*Frees memory allocated by the `ippiEncodeHuffmanStateInitAlloc_JPEG` function.*

## Syntax

```
IppStatus ippiEncodeHuffmanStateFree_JPEG_8u(IppiEncodeHuffmanState* pEncHuffState);
```

## Parameters

`pEncHuffState`                      Pointer to the `IppiEncodeHuffmanState` structure.

## Description

The function `ippiEncodeHuffmanStateFree_JPEG` is declared in the `ippj.h` file. This function frees memory allocated for the `IppiEncodeHuffmanState` structure.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
--------------------------	---------------------

## EncodeHuffman8x8\_JPEG

*Performs Huffman baseline encoding of an 8x8 block of quantized DCT coefficients.*

## Syntax

```
IppStatus ippiEncodeHuffman8x8_JPEG_16s1u_C1(const Ipp16s* pSrc, Ipp8u* pDst, int  
dstLenBytes, int* pDstCurrPos, Ipp16s* pLastDC, const IppiEncodeHuffmanSpec* pDcTable,  
const IppiEncodeHuffmanSpec* pAcTable, IppiEncodeHuffmanState* pEncHuffState, int  
bFlushState);
```

## Parameters

<code>pSrc</code>	Pointer to the 8x8 block of the quantized DCT coefficients.
<code>pDst</code>	Pointer to the buffer for the output bit stream.
<code>dstLenBytes</code>	Length in bytes of the buffer for the bit stream.

<i>pDstCurrPos</i>	Shift in bytes of the current byte in the output buffer.
<i>pLastDC</i>	Pointer to the DC coefficient of the previous 8x8 block.
<i>pDcTable</i>	Pointer to the Huffman table for the DC coefficients.
<i>pAcTable</i>	Pointer to the Huffman table for the AC coefficients.
<i>pEncHuffmanState</i>	Pointer to the <code>IppiEncodeHuffmanState</code> structure.
<i>bFlushState</i>	Set to 1 for the last 8x8 block in the scan.

## Description

The function `ippiEncodeHuffman8x8_JPEG` is declared in the `ippj.h` file. This function performs encoding of an 8x8 block of the quantized DCT coefficients using *pDcTable* and *pAcTable* tables. Encoding procedure is specified in [ISO10918], Annex F.1.2, Baseline Huffman Encoding Procedures. Only full bytes are written to the output buffer. The `IppiEncodeHuffmanState` structure collects the bits that do not make up a complete byte. To force adding the bits accumulated in `IppiEncodeHuffmanState` to the output buffer, set the parameter *bFlushState* to 1 for the last 8x8 block in the scan or restart interval being encoded. In all other cases it must be set to zero.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if at least one of the specified pointers is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>dstLenBytes</i> has a zero or negative value, or <i>pDstCurrPos</i> is outside the limit of <i>dstLenBytes</i> .
<code>ippStsJPEGDCTRangeErr</code>	Indicates an error condition if a DCT coefficient is out of the allowed range (-1023..1023).

## EncodeHuffman8x8\_Direct\_JPEG

*Directly performs Huffman baseline encoding of an 8x8 block of quantized DCT coefficients.*

---

## Syntax

```
IppStatus ippiEncodeHuffman8x8_Direct_JPEG_16slu_C1(const Ipp16s* pSrc, Ipp8u* pDst, int*
pDstBitsLen, Ipp16s* pLastDC, const IppiEncodeHuffmanSpec* pDcTable, const
IppiEncodeHuffmanSpec* pAcTable);
```

## Parameters

<i>pSrc</i>	Pointer to the 8x8 block of the quantized DCT coefficients.
<i>pDst</i>	Pointer to the buffer for the output bit stream.
<i>pDstBitsLen</i>	Length in bits of the buffer for the bit stream.
<i>pLastDC</i>	Pointer to the DC coefficient of the previous 8x8 block of the same color component.
<i>pDcTable</i>	Pointer to the Huffman table for the DC coefficients.
<i>pAcTable</i>	Pointer to the Huffman table for the AC coefficients.

## Description

The function `ippiEncodeHuffman8x8_Direct_JPEG` is declared in the `ippj.h` file. This function directly performs encoding of an 8x8 block of the quantized DCT coefficients using `pDcTable` and `pAcTable` tables. The function does not require any additional structure for operation. Encoding procedure is specified in [ISO10918], Annex F.1.2, Baseline Huffman Encoding Procedures.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one or all of the specified pointers are NULL.

## GetHuffmanStatistics8x8\_JPEG

*Computes Huffman symbols statistics for the baseline encoding.*

---

## Syntax

```
IppStatus ippiGetHuffmanStatistics8x8_JPEG_16s_C1(const Ipp16s* pSrc, int  
pDcStatistics[256], int pAcStatistics[256], Ipp16s* pLastDC);
```

## Parameters

<code>pSrc</code>	Pointer to the 8x8 block of the quantized DCT coefficients.
<code>pDcStatistics</code>	Pointer to the Huffman symbols statistics buffer for the DC coefficients.
<code>pAcStatistics</code>	Pointer to the Huffman symbols statistics buffer for the AC coefficients.
<code>pLastDC</code>	Pointer to the DC coefficient of the previous 8x8 block.

## Description

The function `ippiGetHuffmanStatistics8x8_JPEG` is declared in the `ippj.h` file. This function computes the statistics of Huffman symbols for an 8x8 block of the quantized DCT coefficients for the baseline encoding.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one or both of the specified pointers are NULL.
<code>ippStsJPEGDCTRangeErr</code>	Indicates an error condition if a DCT coefficient is out of the allowed range $(-1023..1023)$ .

## GetHuffmanStatistics8x8\_DCFirst\_JPEG

*Computes Huffman symbols statistics for the progressive encoding (DC coefficients).*

---

### Syntax

```
IppStatus ippiGetHuffmanStatistics8x8_DCFirst_JPEG_16s_C1(const Ipp16s* pSrc, int
pDcStatistics[256], Ipp16s* pLastDC, int Al);
```

### Parameters

<i>pSrc</i>	Pointer to the 8x8 block of the quantized DCT coefficients.
<i>pDcStatistics</i>	Pointer to the Huffman symbols statistics buffer for the DC coefficients.
<i>pLastDC</i>	Pointer to the DC coefficient of the previous 8x8 block.
<i>Al</i>	Successive approximation bit position low, it specifies the actual point transform.

### Description

The function `ippiGetHuffmanStatistics8x8_DCFirst_JPEG` is declared in the `ippj.h` file. This function computes the Huffman symbols statistics for an 8x8 block of the quantized DCT coefficients for progressive encoding, the first scan, DC coefficients.

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one or both of the specified pointers are NULL.
<code>ippStsJPEGDCTRangeErr</code>	Indicates an error condition if a DCT coefficient is out of the allowed range $(-1023..1023)$ .

## GetHuffmanStatistics8x8\_ACFirst\_JPEG

*Computes Huffman symbols statistics for the progressive encoding (AC coefficients, the first scan).*

---

### Syntax

```
IppStatus ippiGetHuffmanStatistics8x8_ACFirst_JPEG_16s_C1(const Ipp16s* pSrc, int
pAcStatistics[256], int Ss, int Se, int Al, IppiEncodeHuffmanState* pEncHuffState, int
bFlushState);
```

### Parameters

<i>pSrc</i>	Pointer to the 8x8 block of the quantized DCT coefficients.
<i>pAcStatistics</i>	Pointer to the Huffman symbols statistics buffer for the AC coefficients.
<i>Ss</i>	Spectral selection start index.

<i>Se</i>	Spectral selection end index.
<i>Al</i>	Successive approximation bit positions low, it specifies the actual point transform.
<i>pEncHuffState</i>	Pointer to the <code>IppiEncodeHuffmanState</code> structure.
<i>bFlushState</i>	Set to 1 for the last 8x8 block in the scan.

## Description

The function `ippiGetHuffmanStatistics8x8_ACFirst_JPEG` is declared in the `ippj.h` file. This function computes the Huffman symbols statistics for an 8x8 block of the quantized DCT coefficients for progressive encoding, the first scan, AC coefficients.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one or all of the specified pointers are NULL.
<code>ippStsJPEGDCTRangeErr</code>	Indicates an error condition if a DCT coefficient is out of the allowed range $(-1023..1023)$ .

## GetHuffmanStatistics8x8\_ACRefine\_JPEG

*Computes Huffman symbols statistics for the progressive encoding (AC coefficients, subsequent scans).*

## Syntax

```
ippStatus ippiGetHuffmanStatistics8x8_ACRefine_JPEG_16s_C1(const Ipp16s* pSrc, int
pAcStatistics [256], int Ss, int Se, int Al, IppiEncodeHuffmanState* pEncHuffState, int
bFlushState);
```

## Parameters

<i>pSrc</i>	Pointer to the 8x8 block of the quantized DCT coefficients.
<i>pAcStatistics</i>	Pointer to the Huffman symbols statistics buffer for the AC coefficients.
<i>Ss</i>	Spectral selection start index.
<i>Se</i>	Spectral selection end index.
<i>Al</i>	Successive approximation bit positions low, it specifies the actual point transform.
<i>pEncHuffState</i>	Pointer to the <code>IppiEncodeHuffmanState</code> structure.
<i>bFlushState</i>	Set to 1 for the last 8x8 block in the scan.

## Description

The function `ippiGetHuffmanStatistics8x8_ACRefine_JPEG` is declared in the `ippj.h` file. This function computes the Huffman symbols statistics for an 8x8 block of the quantized DCT coefficients for progressive encoding, the subsequent scans, AC coefficients.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one or all of the specified pointers are NULL.
<code>ippStsJPEGOutOfBufErr</code>	Indicates an error condition if the buffer limits are exceeded.
<code>ippStsJPEGDCTRangeErr</code>	Indicates an error condition if a DCT coefficient is out of the allowed range (-1023..1023).

## EncodeHuffman8x8\_DCFirst\_JPEG

*Performs progressive encoding of the DC coefficient from an 8x8 block of the quantized DCT coefficients (first scan).*

---

## Syntax

```
ippStatus ippEncodeHuffman8x8_DCFirst_JPEG_16slu_C1(const Ipp16s* pSrc, Ipp8u* pDst, int
dstLenBytes, int* pDstCurrPos, Ipp16s* pLastDC, int Al, IppiEncodeHuffmanSpec* pDcTable,
IppiEncodeHuffmanState* pEncHuffState, int bFlushState);
```

## Parameters

<code>pSrc</code>	Pointer to the 8x8 block of the quantized DCT coefficients.
<code>pDst</code>	Pointer to the buffer for the output bit stream.
<code>dstLenBytes</code>	Length in bytes of the buffer for the bit stream.
<code>pDstCurrPos</code>	Shift in bytes of the current byte in the output buffer.
<code>pLastDC</code>	Pointer to the DC coefficient of the previous 8x8 block.
<code>Al</code>	Successive approximation bit positions low, it specifies the actual point transform.
<code>pDcTable</code>	Pointer to the Huffman table for DC coefficients.
<code>pEncHuffState</code>	Pointer to the <code>IppiEncodeHuffmanState</code> structure.
<code>bFlushState</code>	Set it to 1 for the last 8x8 block in the scan

## Description

The function `ippEncodeHuffman8x8_DCFirst_JPEG` is declared in the `ippj.h` file. This function performs encoding of the DC coefficient from an 8x8 block of the quantized coefficients (progressive encoding, the first scan), using `pDcTable` table. The encoding procedure conforms to [ISO10918], Annex G.1.2, *Progressive Encoding Procedures with Huffman*. Only full bytes are written to the output buffer. The `IppiEncodeHuffmanState` structure collects the bits that do not make up a complete byte. To force adding the bits accumulated in `IppiEncodeHuffmanState` to the output buffer, set the parameter `bFlushState` to 1 for the last 8x8 block in the scan or restart interval being encoded. In all other cases it must be set to zero.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
--------------------------	---------------------

<code>ippStsNullPtrErr</code>	Indicates an error condition if one or all of the specified pointers are NULL.
<code>ippStsJPEGOOutOfBufErr</code>	Indicates an error condition if the buffer limits are exceeded.
<code>ippStsJPEGDCTRangeErr</code>	Indicates an error condition if the DCT coefficient is out of the allowed range $(-1023..1023)$ .

## EncodeHuffman8x8\_DCRefine\_JPEG

*Performs progressive encoding of the DC coefficient from an 8x8 block of the quantized DCT coefficients (subsequent scans).*

### Syntax

```
ippStatus ippiEncodeHuffman8x8_DCRefine_JPEG_16slu_C1(const Ipp16s* pSrc, Ipp8u* pDst,
int dstLenBytes, int* pDstCurrPos, int Al, IppiEncodeHuffmanState* pEncHuffState, int
bFlushState);
```

### Parameters

<code>pSrc</code>	Pointer to the 8x8 block of the quantized DCT coefficients.
<code>pDst</code>	Pointer to the buffer for the output bit stream.
<code>dstLenBytes</code>	Length in bytes of the buffer for the bit stream.
<code>pDstCurrPos</code>	Shift in bytes of the current byte in the output buffer.
<code>Al</code>	Successive approximation bit positions low, it specifies the actual point transform.
<code>pEncHuffState</code>	Pointer to the <code>IppiEncodeHuffmanState</code> structure.
<code>bFlushState</code>	Set it to 1 for the last 8x8 block in the scan

### Description

The `ippiEncodeHuffman8x8_DCRefine_JPEG` is declared in the `ippj.h` file. This function performs encoding of the DC coefficient from an 8x8 block of the quantized DCT coefficients (progressive encoding, the subsequent scans), using `pDcTable` table. The encoding procedure conforms to [ISO10918], Annex G.1.2, *Progressive Encoding Procedures with Huffman*. Only full bytes are written to the output buffer. The `IppiEncodeHuffmanState` structure collects the bits that do not make up a complete byte. To force adding the bits accumulated in `IppiEncodeHuffmanState` to the output buffer, set the parameter `bFlushState` to 1 for the last 8x8 block in the scan or restart interval being encoded. In all other cases it must be set to zero.

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one or all of the specified pointers are NULL.
<code>ippStsJPEGOOutOfBufErr</code>	Indicates an error condition if the buffer limits are exceeded.
<code>ippStsJPEGDCTRangeErr</code>	Indicates an error condition if the DCT coefficient is out of the allowed range $(-1023..1023)$ .

## EncodeHuffman8x8\_ACFirst\_JPEG

*Performs progressive encoding of the AC coefficients from an 8x8 block of the quantized DCT coefficients (first scan).*

---

### Syntax

```
IppStatus ippiEncodeHuffman8x8_ACFirst_JPEG_16slu_C1(const Ipp16s* pSrc, Ipp8u* pDst, int
dstLenBytes, int* pDstCurrPos, int Ss, int Se, int Al, IppiEncodeHuffmanSpec* pAcTable,
IppiEncodeHuffmanState* pEncHuffState, int bFlushState);
```

### Parameters

<i>pSrc</i>	Pointer to the 8x8 block of the quantized DCT coefficients
<i>pDst</i>	Pointer to the buffer for the output bit stream.
<i>dstLenBytes</i>	Length in bytes of the buffer for the bit stream.
<i>pDstCurrPos</i>	Shift in bytes of the current byte in the output buffer.
<i>Ss</i>	Spectral selection start index.
<i>Se</i>	Spectral selection end index.
<i>Al</i>	Successive approximation bit positions low, it specifies the actual point transform.
<i>pAcTable</i>	Pointer to the Huffman table for AC coefficients.
<i>pEncHuffState</i>	Pointer to the <i>IppiEncodeHuffmanState</i> structure.
<i>bFlushState</i>	Set it to 1 for the last 8x8 block in the scan.

### Description

The function `ippiEncodeHuffman8x8_ACFirst_JPEG` is declared in the `ippj.h` file. This function performs encoding of the AC coefficients from an 8x8 block of the quantized DCT coefficients (progressive encoding, the first scan), using *pAcTable* table. The encoding procedure conforms to [ISO10918], Annex G.1.2, *Progressive Encoding Procedures with Huffman*. Only full bytes are written to the output buffer. The *IppiEncodeHuffmanState* structure collects the bits that do not make up a complete byte. To force adding the bits accumulated in *IppiEncodeHuffmanState* to the output buffer, set the parameter *bFlushState* to 1 for the last 8x8 block in the scan or restart interval being encoded. In all other cases it must be set to zero.

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one or all of the specified pointers are NULL.
<code>ippStsJPEGOutOfBufErr</code>	Indicates an error condition if the buffer limits are exceeded.
<code>ippStsJPEGDCTRangeErr</code>	Indicates an error condition if the DCT coefficient is out of the allowed range (-1023..1023).



## EncodeHuffman8x8\_ACRefine\_JPEG

*Performs progressive encoding of the AC coefficients from an 8x8 block of the quantized DCT coefficients.*

### Syntax

```
IppStatus ippiEncodeHuffman8x8_ACRefine_JPEG_16slu_C1(const Ipp16s* pSrc, Ipp8u* pDst,
int dstLenBytes, int* pDstCurrPos, int Ss, int Se, int Al, IppiEncodeHuffmanSpec* pAcTable,
IppiEncodeHuffmanState* pEncHuffState, int bFlushState);
```

### Parameters

<i>pSrc</i>	Pointer to the 8x8 block of the quantized DCT coefficients.
<i>pDst</i>	Pointer to the buffer for the output bit stream.
<i>dstLenBytes</i>	Length in bytes of the buffer for the bit stream.
<i>pDstCurrPos</i>	Shift in bytes of the current byte in the output buffer.
<i>Ss</i>	Spectral selection start index.
<i>Se</i>	Spectral selection end index.
<i>Al</i>	Successive approximation bit positions low, it specifies the actual point transform.
<i>pAcTable</i>	Pointer to the Huffman table for AC coefficients.
<i>pEncHuffState</i>	Pointer to the <code>IppiEncodeHuffmanState</code> structure.
<i>bFlushState</i>	Set it to 1 for the last 8x8 block in the scan.

### Description

The function `ippiEncodeHuffman8x8_ACRefine_JPEG` is declared in the `ippj.h` file. This function performs encoding of the AC coefficient from an 8x8 block of the quantized DCT coefficients (progressive encoding, the subsequent scans), using *pAcTable* table. The encoding procedure conforms to [ISO10918], Annex G.1.2, Progressive Encoding Procedures with Huffman. Only full bytes are written to the output buffer. The `IppiEncodeHuffmanState` structure collects the bits that do not make up a complete byte. To force adding the bits accumulated in `IppiEncodeHuffmanState` to the output buffer, set the parameter *bFlushState* to 1 for the last 8x8 block in the scan or restart interval being encoded. In all other cases it must be set to zero.

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one or all of the specified pointers are NULL.
<code>ippStsJPEGOutOfBufErr</code>	Indicates an error condition if the buffer limits are exceeded.
<code>ippStsJPEGDCTRangeErr</code>	Indicates an error condition if the DCT coefficient is out of the allowed range $(-1023..1023)$ .

## DecodeHuffmanSpecGetBufSize\_JPEG

Returns the length of the `IppiDecodeHuffmanSpec` structure.

---

### Syntax

```
IppStatus ippiDecodeHuffmanSpecGetBufSize_JPEG_8u(int* pSize);
```

### Parameters

<i>pSize</i>	Pointer to a variable that will receive the length of the <code>IppiDecodeHuffmanSpec</code> structure.
--------------	---

### Description

The function `ippiDecodeHuffmanSpecGetBufSize_JPEG` is declared in the `ippj.h` file. This function returns the length of the `IppiDecodeHuffmanSpec` structure.



---

**NOTE.** The fields of the structure `IppiDecodeHuffmanSpec` are hidden from the user.

---

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if the pointer is <code>NULL</code> .

## DecodeHuffmanSpecInit\_JPEG

Creates Huffman table in a format that is suitable for a decoder

---

### Syntax

```
IppStatus ippiDecodeHuffmanSpecInit_JPEG_8u(const Ipp8u* pListBits, const Ipp8u* pListVals, IppiDecodeHuffmanSpec* pDecHuffSpec);
```

### Parameters

<i>pListBits</i>	Pointer to <i>Bits</i> list.
<i>pListVals</i>	Pointer to <i>Vals</i> list.
<i>pDecHuffSpec</i>	Pointer to the Huffman table for the decoder.

### Description

The function `ippiDecodeHuffmanSpecInit_JPEG` is declared in the `ippj.h` file. This function creates Huffman table in a format that is suitable for a decoder. The *Bits* and *Vals* lists are specified in [ISO10918], Annex C, *Huffman Table Specification*. To generate the table, the function uses the procedure that is described in [ISO10918], Annex C.2, *Conversion of Huffman Tables Specified in Interchange Format to Tables of Codes and Code Lengths*.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one or all specified pointers are NULL.
<code>ippStsJPEGHuffTableErr</code>	Indicates an error condition during the initialization of the table. It may happen when raw tables contain inadmissible values.

## DecodeHuffmanSpecInitAlloc\_JPEG

*Allocates memory and creates Huffman table in a format that is suitable for a decoder.*

### Syntax

```
IppStatus ippidecodeHuffmanSpecInitAlloc_JPEG_8u(const Ipp8u* pListBits, const Ipp8u* pListVals, IppiDecodeHuffmanSpec** ppDecHuffSpec);
```

### Parameters

<code>pListBits</code>	Pointer to <i>Bits</i> list.
<code>pListVals</code>	Pointer to <i>Vals</i> list.
<code>ppDecHuffSpec</code>	Pointer to the returned pointer to the Huffman table for the decoder.

### Description

The function `ippidecodeHuffmanSpecInitAlloc_JPEG` is declared in the `ippj.h` file. This function allocates memory and creates the Huffman table in a format that is suitable for a decoder. The function expects that *Bits* and *Vals* lists are supplied in the format specified in [ISO10918], Annex C, *Huffman Table Specification*. To generate the table, the function uses the procedure that is described in [ISO10918], Annex C.2, *Conversion of Huffman Tables Specified in Interchange Format to Tables of Codes and Code Lengths*.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one or all of the specified pointers are NULL.
<code>ippStsJPEGHuffTableErr</code>	Indicates an error condition during the initialization of the table. It may happen when raw tables contain inadmissible values.

## DecodeHuffmanSpecFree\_JPEG

*Frees memory allocated by `ippidecodeHuffmanSpecInitAlloc_JPEG` function.*

### Syntax

```
IppStatus ippidecodeHuffmanSpecFree_JPEG_8u(IppiDecodeHuffmanSpec* pDecHuffSpec);
```

## Parameters

*pDecHuffSpec*                      Pointer to the Huffman table for the decoder.

## Description

The function `ippiDecodeHuffmanSpecFree_JPEG` is declared in the `ippj.h` file. This function frees memory allocated by the `ippiDecodeHuffmanSpecInitAlloc_JPEG` function for the Huffman table.

## Return Values

`ippStsNoErr`                      Indicates no error.

## DecodeHuffmanStateGetBufSize\_JPEG

*Returns the length of `IppiDecodeHuffmanState` structure.*

---

## Syntax

```
IppStatus ippiDecodeHuffmanStateGetBufSize_JPEG_8u(int* pSize);
```

## Parameters

*pSize*                              Pointer to a variable that will receive the length of the `IppiDecodeHuffmanState` structure.

## Description

The function `ippiDecodeHuffmanStateGetBufSize_JPEG` is declared in the `ippj.h` file. This function returns the length in bytes of the `IppiDecodeHuffmanState` structure.




---

**NOTE.** The fields of the structure `IppiDecodeHuffmanState` are hidden from the user.

---

## Return Values

`ippStsNoErr`                      Indicates no error.  
`ippStsNullPtrErr`                Indicates an error condition if the specified pointer is `NULL`.

## DecodeHuffmanStateInit\_JPEG

*Initializes `IppiDecodeHuffmanState` structure.*

---

## Syntax

```
IppStatus ippiDecodeHuffmanStateInit_JPEG_8u(IppiDecodeHuffmanState* pDecHuffState);
```

## Parameters

*pDecHuffState*                      Pointer to the `IppiDecodeHuffmanState` structure.

## Description

The function `ippiDecodeHuffmanStateInit_JPEG` is declared in the `ippj.h` file. This function initializes the `IppiDecodeHuffmanState` structure to the initial state and must be called prior to the JPEG bit stream decoding.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one or all of the specified pointers are <code>NULL</code> .

## DecodeHuffmanStateInitAlloc\_JPEG

*Allocates memory and initializes  
`IppiDecodeHuffmanState` structure.*

---

## Syntax

```
IppStatus ippiDecodeHuffmanStateInitAlloc_JPEG_8u(IppiDecodeHuffmanState** ppDecHuffState);
```

## Parameters

*ppDecHuffState*                      Pointer to the `IppiDecodeHuffmanState` structure.

## Description

The function `ippiDecodeHuffmanStateInit_JPEG` is declared in the `ippj.h` file. This function allocates memory and initializes the `IppiDecodeHuffmanState` structure to the initial state. This function must be called prior to the JPEG bit stream decoding.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one or all of the specified pointers are <code>NULL</code> .

## DecodeHuffmanStateFree\_JPEG

*Frees memory allocated by  
`ippiDecodeHuffmanStateInitAlloc_JPEG` function.*

---

## Syntax

```
IppStatus ippiDecodeHuffmanStateFree_JPEG_8u(IppiDecodeHuffmanState* pDecHuffState);
```

## Parameters

*pDecHuffState*                      Pointer to the `IppiDecodeHuffmanState` structure.

## Description

The function `ippiDecodeHuffmanStateFree_JPEG` is declared in the `ippj.h` file. This function frees memory allocated by the `ippiDecodeHuffmanStateInitAlloc_JPEG_8u` function for the `IppiDecodeHuffmanState` structure.

## Return Values

`ippStsNoErr` Indicates no error.

## DecodeHuffman8x8\_JPEG

*Performs Huffman baseline decoding of 8x8 block of the quantized DCT Coefficients.*

---

## Syntax

```
IppStatus ippiDecodeHuffman8x8_JPEG_1u16s_C1(const Ipp8u* pSrc, int srcLenBytes, int*
pSrcCurrPos, Ipp16s* pDst, Ipp16s* pLastDC, int* pMarker, const IppiDecodeHuffmanSpec*
pDcTable, const IppiDecodeHuffmanSpec* pAcTable, IppiDecodeHuffmanState* pDecHuffState);
```

## Parameters

<code>pSrc</code>	Pointer to the source buffer with the JPEG bit stream.
<code>srcLenBytes</code>	Length in bytes of the buffer for the bit stream.
<code>pSrcCurrPos</code>	Shift in bytes of the current byte in the buffer.
<code>pDst</code>	Pointer to the 8x8 block of the quantized DCT coefficients.
<code>pLastDC</code>	Pointer to the DC coefficient of the previous 8x8 block.
<code>pMarker</code>	Pointer to a variable that will receive JPEG marker detected during decoding.
<code>pDcTable</code>	Pointer to the Huffman table for the DC coefficients.
<code>pAcTable</code>	Pointer to the Huffman table for the AC coefficients.
<code>pDecHuffState</code>	Pointer to the <code>IppiDecodeHuffmanState</code> structure.

## Description

The function `ippiDecodeHuffman8x8_JPEG` is declared in the `ippj.h` file. This function decodes an 8x8 block of the quantized DCT coefficients using `pDcTable` and `pAcTable` tables. The decoding procedure conforms to [ISO10918], Annex F.2.2, Baseline Huffman Decoding Procedures. If a JPEG marker is detected during decoding, the function stops decoding and writes the marker to a location indicated by `pMarker`.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicate an error condition if <code>srcLenBytes</code> has a zero or negative value, or <code>pSrcCurrPos</code> is out of <code>srcLenBytes</code> limit.

<code>ippStsJPEGDCTRangeErr</code>	Indicates an error condition if the DCT coefficient is out of the allowed range $(-1023..1023)$ .
<code>ippStsJPEGOutOfBufErr</code>	Indicates an error condition if the buffer limits are exceeded.
<code>ippStsJPEGMarkerWarn</code>	Indicates a warning if a JPEG marker is detected.

## DecodeHuffman8x8\_Direct\_JPEG

*Directly performs Huffman baseline decoding of an 8x8 block of the quantized DCT coefficients.*

### Syntax

```
ippStatus ippDecodeHuffman8x8_Direct_JPEG_lu16s_C1(const Ipp8u* pSrc, int* pSrcBitsLen,
Ipp16s* pDst, Ipp16s* pLastDC, int* pMarker, Ipp32u* pPrefetchedBits, int*
pNumValidPrefetchedBits, const IppiDecodeHuffmanSpec* pDcTable, const IppiDecodeHuffmanSpec*
pAcTable);
```

### Parameters

<code>pSrc</code>	Pointer to the source buffer with the JPEG bit stream.
<code>pSrcBitsLen</code>	Length in bits of the buffer for the bit stream.
<code>pDst</code>	Pointer to the 8x8 block of the quantized DCT coefficients.
<code>pLastDC</code>	Pointer to the DC coefficient of the previous 8x8 block of the same color component.
<code>pMarker</code>	Pointer to a variable that will receive JPEG marker detected during decoding.
<code>pPrefetchedBits</code>	Pointer to the prefetch buffer which contains the previously decoded bits.
<code>pNumValidPrefetchedBits</code>	Number of valid bits in the prefetch buffer.
<code>pDcTable</code>	Pointer to the Huffman table for the DC coefficients.
<code>pAcTable</code>	Pointer to the Huffman table for the AC coefficients.

### Description

The function `ippDecodeHuffman8x8_Direct_JPEG` is declared in the `ippj.h` file. This function directly decodes an 8x8 block of the quantized DCT coefficients using `pDcTable` and `pAcTable` tables. The function does not require any additional structure for operation. The decoding procedure conforms to [ISO10918], Annex F.2.2, *Baseline Huffman Decoding Procedures*. If a JPEG marker is detected during decoding, the function stops decoding and writes the marker to a location indicated by `pMarker`.

`pLastDC` should be set to 0 through the function initialization or after each restart interval. `pMarker` should be set to 0 through the function initialization or after the found marker has been processed.

`pNumValidPrefetchedBits` should be set to 0 in the following cases: 1) through the function initialization; 2) after each restart interval; 3) after each found marker has been processed.

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
--------------------------	---------------------

<code>ippStsNullPtrErr</code>	Indicates an error condition if one or all of the specified pointers are NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <code>pSrcBitsLen</code> or <code>pNumValidPrefetchedBits</code> has negative value.
<code>ippStsJPEGDCTRangeErr</code>	Indicates an error condition if the DCT coefficient is out of the allowed range $(-1023..1023)$ .
<code>ippStsJPEGOutOfBufErr</code>	Indicates an error condition if the buffer limits are exceeded.

## DecodeHuffman8x8\_DCFirst\_JPEG

*Performs progressive decoding of the DC coefficient from an 8x8 block of the quantized DCT coefficients (first scan).*

---

### Syntax

```
ippStatus ippDecodeHuffman8x8_DCFirst_JPEG_1u16s_C1(const Ipp8u* pSrc, int srcLenBytes,
int* pSrcCurrPos, Ipp16s* pDst, Ipp16s* pLastDC, int* pMarker, int Al,
IppiDecodeHuffmanSpec* pDcTable, IppiDecodeHuffmanState* pDecHuffState);
```

### Parameters

<code>pSrc</code>	Pointer to the buffer with the JPEG bit stream.
<code>srcLenBytes</code>	Length in bytes of the buffer for the bit stream.
<code>pSrcCurrPos</code>	Shift in bytes of the current byte in the buffer.
<code>pDst</code>	Pointer to the 8x8 block of the quantized DCT coefficients.
<code>pLastDC</code>	Pointer to the DC coefficient of the previous 8x8 block.
<code>pMarker</code>	Pointer to a variable that will receive JPEG marker detected during decoding.
<code>Al</code>	Successive approximation bit positions low, it specifies the actual point transform.
<code>pDcTable</code>	Pointer to the Huffman table for the DC coefficients.
<code>pDecHuffState</code>	Pointer to the <code>IppiDecodeHuffmanState</code> structure.

### Description

The function `ippDecodeHuffman8x8_DCFirst_JPEG` is declared in the `ippj.h` file. This function decodes the DC coefficient from an 8x8 block of the quantized DCT coefficients, progressive mode, the first scan, using `pDcTable` table. The decoding procedure conforms to [ISO10918], Annex G.2, *Progressive Decoding of the DCT*. If a JPEG marker is detected during decoding, the function stops decoding and writes the marker to a location indicated by `pMarker`.

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one or all of the specified pointers are NULL.



<code>ippStsJPEGDCTRangeErr</code>	Indicates an error condition if the DCT coefficient is out of the allowed range $(-1023..1023)$ .
<code>ippStsJPEGOutOfBufErr</code>	Indicates an error condition if the buffer limits are exceeded.
<code>ippStsJPEGMarkerWarn</code>	Indicates a warning if a JPEG marker is detected.

## DecodeHuffman8x8\_DCRefine\_JPEG

*Performs progressive decoding of the DC coefficient from an 8x8 block of the quantized DCT coefficients.*

### Syntax

```
ippStatus ippDecodeHuffman8x8_DCRefine_JPEG_1ul6s_C1(const Ipp8u* pSrc, int srcLenBytes,
int* pSrcCurrPos, Ipp16s* pDst, int* pMarker, int Al, IppiDecodeHuffmanState*
pDecHuffState);
```

### Parameters

<code>pSrc</code>	Pointer to the buffer with the JPEG bit stream.
<code>srcLenBytes</code>	Length in bytes of the buffer for the bit stream.
<code>pSrcCurrPos</code>	Shift in bytes of the current byte in the buffer.
<code>pDst</code>	Pointer to the 8x8 block of the quantized DCT coefficients.
<code>pMarker</code>	Pointer to a variable that will receive a JPEG marker detected during decoding.
<code>Al</code>	Successive approximation bit positions low, it specifies the actual point transform.
<code>pDecHuffState</code>	Pointer to the <code>IppiDecodeHuffmanState</code> structure.

### Description

The function `ippDecodeHuffman8x8_DCRefine_JPEG` is declared in the `ippj.h` file. This function decodes the DC coefficient from an 8x8 block of the quantized DCT coefficients, progressive mode, the subsequent scans, using `pDcTable` table. The decoding procedure conforms to [ISO10918], Annex G.2, *Progressive Decoding of the DCT*. If a JPEG marker is detected during decoding, the function stops decoding and writes the marker to a location indicated by `pMarker`.

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one or all of the specified pointers are NULL.
<code>ippStsJPEGDCTRangeErr</code>	Indicates an error condition if the DCT coefficient is out of the allowed range $(-1023..1023)$ .
<code>ippStsJPEGOutOfBufErr</code>	Indicates an error condition if the buffer limits are exceeded.
<code>ippStsJPEGMarkerWarn</code>	Indicates a warning if a JPEG marker is detected.

## DecodeHuffman8x8\_ACFirst\_JPEG

*Performs progressive decoding of the AC coefficients from an 8x8 block of the quantized DCT coefficients (first scan).*

---

### Syntax

```
IppStatus ippiDecodeHuffman8x8_ACFirst_JPEG_1u16s_C1(const Ipp8u* pSrc, int srcLenBytes,
int* pSrcCurrPos, Ipp16s* pDst, int* pMarker, int Ss, int Se, int Al, IppiDecodeHuffmanSpec*
pAcTable, IppiDecodeHuffmanState* pDecHuffState);
```

### Parameters

<i>pSrc</i>	Pointer to the buffer with the JPEG bit stream.
<i>srcLenBytes</i>	Length in bytes of the buffer for the bit stream.
<i>pSrcCurrPos</i>	Shift in bytes of the current byte in the buffer.
<i>pDst</i>	Pointer to the 8x8 block of the quantized DCT coefficients.
<i>pMarker</i>	Pointer to a variable that will receive JPEG marker detected during decoding.
<i>Ss</i>	Spectral selection start index.
<i>Se</i>	Spectral selection end index.
<i>Al</i>	Successive approximation bit positions low, it specifies the actual point transform.
<i>pAcTable</i>	Pointer to the Huffman table for the AC coefficients.
<i>pDecHuffState</i>	Pointer to the <code>IppiDecodeHuffmanState</code> structure.

### Description

The function `ippiDecodeHuffman8x8_ACFirst_JPEG` is declared in the `ippj.h` file. This function decodes AC coefficients from an 8x8 block of the quantized DCT coefficients, progressive mode, the first scan, using *pAcTable* table. The decoding procedure conforms to [ISO10918], Annex G.2, *Progressive Decoding of the DCT*. If a JPEG marker is detected during decoding, the function stops decoding and writes the marker to a location indicated by *pMarker*.

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one or all of the specified pointers are NULL.
<code>ippStsJPEGDCTRangeErr</code>	Indicates an error condition if the DCT coefficient is out of the allowed range (-1023..1023).
<code>ippStsJPEGOutOfBufErr</code>	Indicates an error condition if the buffer limits are exceeded.
<code>ippStsJPEGMarkerWarn</code>	Indicates a warning if a JPEG marker is detected.

## DecodeHuffman8x8\_ACRefine\_JPEG

*Performs progressive decoding of the AC coefficients from an 8x8 block of the quantized DCT coefficients (subsequent scans).*

---

### Syntax

```
IppStatus ippDecodeHuffman8x8_ACRefine_JPEG_lu16s_C1(const Ipp8u* pSrc, int srcLenBytes,
int* pSrcCurrPos, Ipp16s* pDst, int* pMarker, int Ss, int Se, int Al, IppiDecodeHuffmanSpec*
pAcTable, IppiDecodeHuffmanState* pDecHuffState);
```

### Parameters

<i>pSrc</i>	Pointer to the buffer with the JPEG bit stream.
<i>srcLenBytes</i>	Length in bytes of the buffer for the bit stream.
<i>pSrcCurrPos</i>	Shift in bytes of the current byte in the buffer.
<i>pDst</i>	Pointer to the 8x8 block of the quantized DCT coefficients.
<i>pMarker</i>	Pointer to a variable that will receive JPEG marker detected during of decoding.
<i>Ss</i>	Spectral selection start index.
<i>Se</i>	Spectral selection end index.
<i>Al</i>	Successive approximation bit positions low, it specifies the actual point transform.
<i>pAcTable</i>	Pointer to the Huffman table for the AC coefficients.
<i>pDecHuffState</i>	Pointer to the <code>IppiDecodeHuffmanState</code> structure.

### Description

The function `ippDecodeHuffman8x8_ACRefine_JPEG` is declared in the `ippj.h` file. This function decodes AC coefficients from an 8x8 block of the quantized DCT coefficients, progressive mode, subsequent scans, using *pAcTable* table. The decoding procedure conforms to [ISO10918], Annex G.2, *Progressive Decoding of the DCT*. If a JPEG marker is detected during decoding, the function stops decoding and writes the marker to a location indicated by *pMarker*.

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one or all of the specified pointers are NULL.
<code>ippStsJPEGDCTRangeErr</code>	Indicates an error condition if the DCT coefficient is out of the allowed range (-1023..1023).
<code>ippStsJPEGOutOfBufErr</code>	Indicates an error condition if the buffer limits are exceeded.
<code>ippStsJPEGMarkerWarn</code>	Indicates a warning if a JPEG marker is detected.

## Functions for Lossless JPEG Coding

This section describes the functions that are specific for the lossless JPEG process with Huffman coding. These functions are listed in the table below.

### Functions for Lossless JPEG Coding

Function Base Name	Description
<a href="#">DiffPredFirstRow_JPEG</a>	Computes the differences between input sample and predictor for the first line
<a href="#">DiffPredRow_JPEG</a>	Computes the differences between input sample and predictor for all lines but the first.
<a href="#">ReconstructPredFirstRow_JPEG</a>	Reconstructs samples from the decoded differences between input samples and predictor for the first line.
<a href="#">ReconstructPredRow_JPEG</a>	Reconstructs samples from the decoded differences between input samples and predictor for all lines but the first.
<a href="#">GetHuffmanStatisticsOne_JPEG</a>	Computes Huffman symbol statistics
<a href="#">EncodeHuffmanOne_JPEG</a>	Performs Huffman encoding of one difference.
<a href="#">DecodeHuffmanOne_JPEG</a>	Decodes one Huffman coded difference.
<a href="#">EncodeHuffmanRow_JPEG</a>	Performs Huffman encoding of one row of differences for each color component in the JPEG scan.
<a href="#">DecodeHuffmanRow_JPEG</a>	Decodes one row for each color component of Huffman coded differences.

The Intel IPP functions use the coding model in accordance with [\[ISO10918\]](#), *Annex H, Lossless Mode of Operation*.

### DiffPredFirstRow\_JPEG

*Computes the differences between input sample and predictor for the first line.*

#### Syntax

```
IppStatus ippiDiffPredFirstRow_JPEG_16s_C1(const Ipp16s* pSrc, Ipp16s* pDst, int width,
int P, int Pt);
```

#### Parameters

<i>pSrc</i>	Pointer to the row of samples.
<i>pDst</i>	Pointer to the row of calculated differences.
<i>width</i>	Row width in elements; has always the same value for all rows.
<i>P</i>	Sample precision derived from JPEG frame header, varies from 2 to 16.
<i>Pt</i>	Point transformation parameter derived from JPEG scan header; its value should be 0 or a positive integer.

#### Description

The function `ippiDiffPredFirstRow_JPEG` is declared in the `ippj.h` file. This function operates with the first row of samples *pSrc* at the start of the scan and at the beginning of each restart interval only. It computes modulo  $2^{16}$  differences between input samples and predictor with the specified sample precision *P* and performs the point transformation with the specified parameter *Pt*. The function uses special predictors for the first line.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if the <i>width</i> parameter has a negative value.
<code>ippStsBadArgErr</code>	Indicates an error condition if <i>P</i> or <i>Pt</i> has an illegal value.

## Example

The code example below shows how to use the function `ippiDiffPredFirstRow_JPEG_16s_C1`.

```
void func_DiffPredFirstRow()
{
    Ipp16s pSrc[4*4] = { 0,  1,  2,  4,
                        8, 16, 32, 64,
                        32, 16,  8,  4,
                        2,  1,  0,  1};

    Ipp16s pDst[4*4];
    int width = 4;
    int P = 8;
    int Pt = 2;
    int predictor = 2;

    ippiDiffPredFirstRow_JPEG_16s_C1(pSrc, pDst, width, P, Pt);

    for(int j = 1; j < 4; j++)
    {
        ippiDiffPredRow_JPEG_16s_C1(pSrc+j*width, pSrc + (j-1)*width,
```

```
pDst+j*width, width, predictor);
    }
}
```

Result:

```
pDst
-32   1   1   2   <- the first row is obtained using ippiDiffPredFirstRow
  8  15  30  60   the rest are obtained using ippiDiffPredRow
 24   0 -24 -60
-30 -15  -8  -3
```

## DiffPredRow\_JPEG

*Computes the differences between input sample and predictor for all lines but the first.*

---

### Syntax

```
IppStatus ippiDiffPredRow_JPEG_16s_C1(const Ipp16s* pSrc, const Ipp16s* pPrevRow, Ipp16s*
pDst, int width, int predictor);
```

### Parameters

<i>pSrc</i>	Pointer to the row of samples.
<i>pPrevRow</i>	Pointer to the adjacent row of samples just above the current row.
<i>pDst</i>	Pointer to the row of calculated differences.
<i>width</i>	Row width in elements; has always the same value for all rows.
<i>predictor</i>	Selected predictor, should be in the range [1, 7].

### Description

The function `ippiDiffPredRow_JPEG` is declared in the `ippj.h` file. This function operates on all rows of samples except the first row at the start of the scan and at the beginning of the restart interval. It computes modulo  $2^{16}$  differences between input samples *pSrc* and the specified predictor *predictor*. The first sample of the row uses the sample from the row *pPrevRow* above as a predictor.

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if the <i>width</i> parameter has a negative value.
<code>ippStsBadArgrErr</code>	Indicates an error condition if <i>predictor</i> has an illegal value.

## ReconstructPredFirstRow\_JPEG

*Reconstructs samples from the decoded differences between input samples and predictor for the first line.*

---

### Syntax

```
IppStatus ippiReconstructPredFirstRow_JPEG_16s_C1(const Ipp16s* pSrc, Ipp16s* pDst, int width, int P, int Pt);
```

### Parameters

<i>pSrc</i>	Pointer to the row of decoded differences.
<i>pDst</i>	Pointer to the row of reconstructed samples.
<i>width</i>	Row width in elements; has always the same value for all rows.
<i>P</i>	Sample precision derived from JPEG frame header, varies from 2 to 16.
<i>Pt</i>	Point transformation parameter derived from JPEG scan header; its value should be 0 or a positive integer.

### Description

The function `ippiReconstructPredFirstRow_JPEG` is declared in the `ippj.h` file. This function operates on the first row of decoded differences *pSrc* at the start of the scan and restart interval only. It reconstructs output samples *pDst* with the specified sample precision *P* adding decoded differences modulo  $2^{16}$  to the predictions. It also performs the point transformation with the specified parameter *Pt*.

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if the <i>width</i> parameter has a negative value.
<code>ippStsBadArgErr</code>	Indicates an error condition if <i>P</i> or <i>Pt</i> has an illegal value.

## ReconstructPredRow\_JPEG

*Reconstructs samples from the decoded differences between input samples and predictor for all lines but the first.*

---

### Syntax

```
IppStatus ippiReconstructPredRow_JPEG_16s_C1(const Ipp16s* pSrc, const Ipp16s* pPrevRow, Ipp16s* pDst, int width, int predictor);
```

### Parameters

<i>pSrc</i>	Pointer to the row of decoded differences.
-------------	--

<i>pPrevRow</i>	Pointer to the adjacent row of reconstructed samples just above the current row.
<i>pDst</i>	Pointer to the row of reconstructed samples.
<i>width</i>	Row width in elements (has always the same value for all rows).
<i>predictor</i>	Selected predictor, should be in the range [1, 7].

### Description

The function `ippiReconstructPredRow_JPEG` is declared in the `ippj.h` file. This function operates on all rows of decoded differences *pSrc* except first rows at the start of the scan and restart interval. It reconstructs output samples *pDst* adding decoded differences modulo  $2^{16}$  to the predictions specified by the *predictor* parameters. The first sample of the row use the reconstructed sample from the row *pPrevRow* above as a predictor.

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if the <i>width</i> parameter has a negative value.
<code>ippStsBadArgErr</code>	Indicates an error condition if <i>predictor</i> has an illegal value.

## GetHuffmanStatisticsOne\_JPEG

*Computes Huffman symbol statistics.*

---

### Syntax

```
IppStatus ippiGetHuffmanStatisticsOne_JPEG_16s_C1(const Ipp16s* pSrc, int
pHuffStatistics[256]);
```

### Parameters

<i>pSrc</i>	Pointer to the row of decoded differences.
<i>pHuffStatistics</i>	Pointer to the calculated statistics buffer.

### Description

The function `ippiGetHuffmanStatisticsOne_JPEG` is declared in the `ippj.h` file. This function computes frequency of Huffman symbols to be encoded. Based on these statistics, an optimal Huffman table can be built by the `ippiEncodeHuffmanRawTableInit_JPEG` function. This table saves space by removing unused symbols and assigning shorter codes for more frequent symbols.

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointers is <code>NULL</code> .



## EncodeHuffmanOne\_JPEG

*Performs Huffman encoding of one difference.*

---

### Syntax

```
IppStatus ippiEncodeHuffmanOne_JPEG_16s1u_C1(const Ipp16s* pSrc, Ipp8u* pDst, int
nDstLenBytes, int* pDstCurrPos, const IppiEncodeHuffmanSpec* pEncHuffTable,
IppiEncodeHuffmanState* pEncHuffState, int bFlushState);
```

### Parameters

<i>pSrc</i>	Pointer to the difference to be encoded. This pointer can be <code>NULL</code> if <i>bFlushState</i> = 1.
<i>pDst</i>	Pointer to the output bitstream buffer.
<i>nDstLenBytes</i>	Number of available bytes in the output buffer.
<i>pDstCurrPos</i>	Pointer to the current byte in the output buffer. This pointer is updated in the function.
<i>pEncHuffTable</i>	Pointer to the <code>IppiEncodeHuffmanSpec</code> structure that contains the Huffman code table. This pointer can be <code>NULL</code> if <i>bFlushState</i> = 1.
<i>pEncHuffState</i>	Pointer to the <code>IppiEncodeHuffmanState</code> structure that contains the Huffman encoder state.
<i>bFlushState</i>	Setting this parameter to 1 forces the function to flush collected bits from the state structure to the bitstream, setting it to 0 forces to perform encoding.

### Description

The function `ippiEncodeHuffmanOne_JPEG` is declared in the `ippj.h` file. This function encodes one difference *pSrc* using Huffman code table *pEncHuffTable*. Only full bytes are written to the output buffer. The `IppiEncodeHuffmanState` structure collects the bits that do not make up a complete byte. To force adding the bits accumulated in the `IppiEncodeHuffmanState` to the output buffer, the parameter *bFlushState* should be set to 1 when the last sample in scan or restart interval is encoded. In all other cases this parameter must be set to zero.

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	When <i>bFlushState</i> =0, indicates an error condition if one of the specified pointers is <code>NULL</code> When <i>bFlushState</i> =1, indicates an error condition if one of the <i>pDst</i> , <i>pDstCurrPos</i> , or <i>pEncHuffState</i> pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if the <i>nDstLenBytes</i> parameter has zero or negative value, or if <i>pDstCurrPos</i> is out of <i>nDstLenBytes</i> limit.

## DecodeHuffmanOne\_JPEG

*Decodes one Huffman coded difference.*

---

### Syntax

```
IppStatus ippDecodeHuffmanOne_JPEG_1u16s_C1(const Ipp8u* pSrc, int nSrcLenBytes, int* pSrcCurrPos, Ipp16s* pDst, int* pMarker, const IppiDecodeHuffmanSpec* pDecHuffTable, IppiDecodeHuffmanState* pDecHuffState);
```

### Parameters

<i>pSrc</i>	Pointer to the input bitstream.
<i>nSrcLenBytes</i>	Number of available bytes in the input buffer.
<i>pSrcCurrPos</i>	Pointer to the current byte in the input buffer. This pointer is updated in the function.
<i>pDst</i>	Pointer to the output buffer to store the decoded difference.
<i>pMarker</i>	Pointer to a variable that will receive JPEG marker detected during decoding.
<i>pDecHuffTable</i>	Pointer to the <code>IppiDecodeHuffmanSpec</code> structure that contains the Huffman code table.
<i>pDecHuffState</i>	Pointer to the <code>IppiDecodeHuffmanState</code> structure that contains the Huffman decoder state.

### Description

The function `ippDecodeHuffmanOne_JPEG` is declared in the `ippj.h` file. This function decodes one Huffman coded difference pointed to in the bitstream by *pSrc* and stores the result in the output buffer *pDst*. The function uses Huffman code table *pDecHuffTable*.

If a JPEG marker is detected during decoding, the function stops decoding and writes the marker to a location indicated by *pMarker*.

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if the <i>nSrcLenBytes</i> parameter has zero or negative value, or if <i>pSrcCurrPos</i> is out of <i>nSrcLenBytes</i> limit.
<code>ippStsJPEGMarkerWarn</code>	Indicates a warning if a JPEG marker is detected.

## EncodeHuffmanRow\_JPEG

*Performs Huffman encoding of one row of differences for each color component in the JPEG scan.*

### Syntax

```
IppStatus ippiEncodeHuffmanRow_JPEG_16s1u_P4C1(const Ipp16s* pSrc[4], int nSrcLen, int
nSrcRows, Ipp8u* pDst, int nDstLenBytes, int* pDstCurrPos, const IppiEncodeHuffmanSpec*
pEncHuffTable[4], IppiEncodeHuffmanState* pEncHuffState, int bFlushState);
```

### Parameters

<i>pSrc</i>	Array of pointers to the differences to be encoded. It can be NULL if <i>bFlushState</i> = 1.
<i>nSrcLen</i>	Number of elements in the input rows (should be the same for all rows).
<i>nSrcRows</i>	Number of rows (one row for every color component in JPEG scan).
<i>pDst</i>	Pointer to the output bitstream buffer.
<i>nDstLenBytes</i>	Number of available bytes in the output buffer.
<i>pDstCurrPos</i>	Pointer to the current byte in the output buffer. This pointer is updated in the function.
<i>pEncHuffTable</i>	Pointer to the <i>IppiEncodeHuffmanSpec</i> structure (one for every color component in JPEG scan) that contains the Huffman code table. This pointer can be NULL if <i>bFlushState</i> = 1 or if there is no corresponding color component.
<i>pEncHuffState</i>	Pointer to the <i>IppiEncodeHuffmanState</i> structure that contains the Huffman encoder state.
<i>bFlushState</i>	Setting this parameter to 1 forces the function to flush collected bits from the state structure to the bitstream, setting it to 0 forces to perform encoding.

### Description

The function *ippiEncodeHuffmanRow\_JPEG* is declared in the *ippj.h* file. This function encodes *nSrcRows* rows (up to 4, one for every color component) containing *nSrcLen* elements of differences pointed by pointers in the array *pSrc[4]* using corresponding Huffman code tables *pEncHuffTable[4]*. If number of components is less than 4, corresponding pointers are not specified.

Only full bytes are written to the output buffer. The *IppiEncodeHuffmanState* structure collects the bits that do not make up a complete byte. To force adding the bits accumulated in the *IppiEncodeHuffmanState* to the output buffer, the parameter *bFlushState* should be set to 1 when the last sample in scan or restart interval is encoded. In all other cases this parameter must be set to zero.

### Return Values

<i>ippStsNoErr</i>	Indicates no error.
--------------------	---------------------

<code>ippStsNullPtrErr</code>	When <code>bFlushState=0</code> , indicates an error condition if one of the specified pointers is NULL, When <code>bFlushState=1</code> , indicates an error condition if one of the <code>pDst</code> , <code>pDstCurrPos</code> or <code>pEncHuffState</code> pointers is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if the <code>nDstLenBytes</code> parameter has zero or negative value, or if <code>pDstCurrPos</code> is out of <code>nDstLenBytes</code> limit.

## DecodeHuffmanRow\_JPEG

*Decodes one row for each color component of Huffman coded differences.*

---

### Syntax

```
ippStatus ippidecodeHuffmanRow_JPEG_1u16s_C1P4(const Ipp8u* pSrc, int nSrcLenBytes, int*
pSrcCurrPos, Ipp16s* pDst[4], int nDstLen, int nDstRows, int* pMarker, const
IppiDecodeHuffmanSpec* pDecHuffTable[4], IppiDecodeHuffmanState* pDecHuffState);
```

### Parameters

<code>pSrc</code>	Pointer to the input bitstream.
<code>nSrcLenBytes</code>	Number of available bytes in the input buffer.
<code>pSrcCurrPos</code>	Pointer to the current byte in the input buffer. This pointer is updated in the function.
<code>pDst</code>	Array of pointers to the output buffer for each color component in the JPEG scan to store the decoded difference.
<code>pDstLen</code>	Number of elements to decode, must be equal to the image width or JPEG restart interval.
<code>pDstRows</code>	Number of output rows, must be equal to the number of color components.
<code>pMarker</code>	Pointer to a variable that will receive JPEG marker detected during decoding.
<code>pDecHuffTable</code>	Array of pointers to the <code>IppiDecodeHuffmanSpec</code> structures (one for each color component) that contains the Huffman decode table.
<code>pDecHuffState</code>	Pointer to the <code>IppiDecodeHuffmanState</code> structure that contains the Huffman decoder state.

### Description

The function `ippidecodeHuffmanRow_JPEG` is declared in the `ippj.h` file. This function decodes `pDstLen` Huffman coded difference for `nDstRows` rows (one for each color component) using corresponding `pDecHuffTable` table from the bitstream pointed by `pSrc`, and places it to the output buffer pointed by the appropriate pointer in the array `pDst`.

If a JPEG marker is detected during decoding, the function stops decoding and writes the marker to a location indicated by `pMarker`.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if the <code>nSrcLenBytes</code> parameter has zero or negative value, or if <code>pSrcCurrPos</code> is out of <code>nSrcLenBytes</code> limit.
<code>ippStsJPEGMarkerWarn</code>	Indicates a warning if a JPEG marker is detected.

## Wavelet Transform Functions

This section describes the wavelet transform functions that are specific for JPEG 2000 image coding system (see [ISO15444]). These functions are listed in the table below.

### Wavelet Transform Functions

Function Base Name	Description
<b>Low-Level Operations</b>	
<code>WTFwdRow_B53_JPEG2K</code>	Performs a row oriented forward wavelet transform with reversible filter.
<code>WTInvRow_B53_JPEG2K</code>	Performs a row oriented inverse wavelet transform with reversible filter.
<code>WTFwdCol_B53_JPEG2K</code>	Performs a forward wavelet transform with reversible filter of image columns.
<code>WTFwdColLift_B53_JPEG2K</code>	Performs a single step of forward wavelet transform with reversible filter of image columns.
<code>WTInvCol_B53_JPEG2K</code>	Performs a column oriented inverse wavelet transform with reversible filter.
<code>WTInvColLift_B53_JPEG2K</code>	Performs a single step of inverse wavelet transform with reversible filter of image columns.
<code>WTFwdRow_D97_JPEG2K</code>	Performs a row oriented forward wavelet transform with irreversible filter.
<code>WTInvRow_D97_JPEG2K</code>	Performs a row oriented inverse wavelet transform with irreversible filter.
<code>WTFwdCol_D97_JPEG2K</code>	Performs a column oriented forward wavelet transform with irreversible filter.
<code>WTFwdColLift_D97_JPEG2K</code>	Performs a single step of forward wavelet transform with irreversible filter of image columns.
<code>WTInvCol_D97_JPEG2K</code>	Performs a column oriented inverse wavelet transform with irreversible filter.
<code>WTInvColLift_D97_JPEG2K</code>	Performs a single step of inverse wavelet transform with irreversible filter of image columns.
<b>Tile-Oriented Transforms</b>	
<code>WTGetBufSize_B53_JPEG2K</code>	Computes the size of the buffer for a tile-oriented wavelet transform
<code>WTFwd_B53_JPEG2K</code>	Performs a tile-oriented forward wavelet transform.
<code>WTInv_B53_JPEG2K</code>	Performs a tile-oriented inverse wavelet transform.
<code>WTGetBufSize_D97_JPEG2K</code>	Computes the size of the buffer for a tile-oriented wavelet transform
<code>WTFwd_D97_JPEG2K</code>	Performs a tile-oriented forward wavelet transform.
<code>WTInv_D97_JPEG2K</code>	Performs a tile-oriented inverse wavelet transform.

## Low-Level Operations

These functions perform one-dimensional reversible (`_B53` modifier) and irreversible (`_D97` modifier) wavelet transforms of image rows or columns for lossless and lossy compressions, respectively, in accordance with [ISO15444], Annex F, *Discrete Wavelet Transformation of Tile-Components*. The transformations are the lifting-based implementations of filtering by 5-3 reversible and 9-7 irreversible wavelet filters.

The filters have odd length: 5 (9) - for lowpass filters, and 3 (7) - for highpass filters.

There are two possible relative positions of lowpass and highpass filters allowed by the standard, as shown in the Figure "Sample Image Changed by the `ippiResize` Function". The relative position is specified by the `phase` argument that has two possible values:

`ippiWTFilterFirstLow` corresponds to positions a),  
`ippiWTFilterFirstHigh` corresponds to positions b).

The wavelet transform functions do not apply any fixed border extension (symmetrical, wraparound or other) to the source image ROI, instead they require valid and accessible border data outside of the source image ROI.

### Possible Relative Positions of Filters

5-3 filter



low pass filter



high pass filter

9-7 filter



low pass filter



high pass filter

a)

b)

## WTFwdRow\_B53\_JPEG2K

*Performs a forward wavelet transform with reversible filter of image rows.*

### Syntax

```
ippiStatus ippiWTFwdRow_B53_JPEG2K_16s_C1R(const Ipp16s* pSrc, int srcStep, Ipp16s* pDstLow,
int dstLowStep, Ipp16s* pDstHigh, int dstHighStep, IppiSize dstRoiSize, IppiWTFilterFirst
phase);
```

```
ippiStatus ippiWTFwdRow_B53_JPEG2K_32s_C1R(const Ipp32s* pSrc, int srcStep, Ipp32s* pDstLow,
int dstLowStep, Ipp32s* pDstHigh, int dstHighStep, IppiSize dstRoiSize, IppiWTFilterFirst
phase);
```

## Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDstLow</i>	Pointer to ROI of the low frequency component of a destination image.
<i>dstLowStep</i>	Distance in bytes between starts of consecutive lines in the low frequency component of a destination image.
<i>pDstHigh</i>	Pointer to ROI of the high frequency component of a destination image.
<i>dstHighStep</i>	Distance in bytes between starts of consecutive lines in the high frequency component of a destination image.
<i>dstRoiSize</i>	Size of the destination image ROI.
<i>phase</i>	Relative position of the high-pass and low-pass filters.

## Description

The function `ippiWTFwdRow_B53_JPEG2K` is declared in the `ippj.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function performs a wavelet decomposition of the source image rows using the 5-3 reversible filter. Both destination ROIs have the same size *dstRoiSize*, while the source image ROI size is uniquely determined from the following relations:

```
srcRoiSize.width = 2 * dstRoiSize.width;  
srcRoiSize.height = dstRoiSize.height.
```

For proper operation, the function needs valid data outside the source image ROI:

if the *phase* argument is equal to `ippWTFilterFirstLow`, the function requires 2 extra pixels on the left and 1 pixel on the right (outside ROI border) for each processed row;

if the *phase* argument is equal to `ippWTFilterFirstHigh`, the function requires 1 extra pixel on the left and 2 pixels on the right (outside ROI border) for each processed row.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if at least one of the specified pointers is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if the width or height of ROI has zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if any of the specified buffer step values is zero or negative.

## WTInvRow\_B53\_JPEG2K

*Performs an inverse wavelet transform with reversible filter of image rows.*

---

### Syntax

```
IppStatus ippiWTInvRow_B53_JPEG2K_16s_C1R(const Ipp16s* pSrcLow, int srcLowStep, const Ipp16s* pSrcHigh, int srcHighStep, IppiSize srcRoiSize, Ipp16s* pDst, int dstStep, IppiWTFilterFirst phase);
```

```
IppStatus ippiWTInvRow_B53_JPEG2K_32s_C1R(const Ipp32s* pSrcLow, int srcLowStep, const Ipp32s* pSrcHigh, int srcHighStep, IppiSize srcRoiSize, Ipp32s* pDst, int dstStep, IppiWTFilterFirst phase);
```

### Parameters

<i>pSrcLow</i>	Pointer to ROI of the low frequency component of a source image.
<i>srcLowStep</i>	Distance in bytes between starts of consecutive lines in the low frequency component of a source image.
<i>pSrcHigh</i>	Pointer to ROI of the high frequency component of a source image.
<i>srcHighStep</i>	Distance in bytes between starts of consecutive lines in the high frequency component of a source image.
<i>srcRoiSize</i>	Size of the source image ROI.
<i>pDst</i>	Pointer to ROI of the destination image.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>phase</i>	<a href="#">Relative position of the high-pass and low-pass filters.</a>

### Description

The function `ippiWTInvRow_B53_JPEG2K` is declared in the `ippj.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function performs a wavelet reconstruction of the source image rows using the 5-3 reversible filter. Both source ROIs have the same size *srcRoiSize*, while the destination image ROI size is uniquely determined from the following relations:

```
dstRoiSize.width = 2 * srcRoiSize.width;
```

```
dstRoiSize.height = srcRoiSize.height.
```

For proper operation, the function needs valid data outside the source image ROI.

For the low frequency component:

if the *phase* argument is equal to `ippWTFilterFirstLow`, the function requires 1 extra pixel on the right (outside ROI border) for each processed row;

if the *phase* argument is equal to `ippWTFilterFirstHigh`, the function requires 1 extra pixel on the left (outside ROI border) for each processed row.



For the high frequency component, the function always requires 1 extra pixel on the left and 1 pixel on the right (outside ROI border) for each processed row.

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if at least one of the specified pointers is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if the width or height of ROI has zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if any of the specified buffer step values is zero or negative.

## WTFwdCol\_B53\_JPEG2K

*Performs a forward wavelet transform with reversible filter of image columns.*

### Syntax

```
IppStatus ippiWTFwdCol_B53_JPEG2K_16s_C1R(const Ipp16s* pSrc, int srcStep, Ipp16s* pDstLow,
int dstLowStep, Ipp16s* pDstHigh, int dstHighStep, IppiSize dstRoiSize, IppiWTFilterFirst
phase);
```

```
IppStatus ippiWTFwdCol_B53_JPEG2K_32s_C1R(const Ipp32s* pSrc, int srcStep, Ipp32s* pDstLow,
int dstLowStep, Ipp32s* pDstHigh, int dstHighStep, IppiSize dstRoiSize, IppiWTFilterFirst
phase);
```

### Parameters

<code>pSrc</code>	Pointer to the source image ROI.
<code>srcStep</code>	Distance in bytes between starts of consecutive lines in the source image.
<code>pDstLow</code>	Pointer to ROI of the low frequency component of a destination image.
<code>dstLowStep</code>	Distance in bytes between starts of consecutive lines in the low frequency component of a destination image.
<code>pDstHigh</code>	Pointer to ROI of the high frequency component of a destination image.
<code>dstHighStep</code>	Distance in bytes between starts of consecutive lines in the high frequency component of a destination image.
<code>dstRoiSize</code>	Size of the destination image ROI.
<code>phase</code>	Relative position of the high-pass and low-pass filters.

### Description

The function `ippiWTFwdCol_B53_JPEG2K` is declared in the `ippj.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function performs a wavelet decomposition of the source image columns using the 5-3 reversible filter. Both destination ROIs have the same size *dstRoiSize*, while the source image ROI size is uniquely determined from the following relations:

```
srcRoiSize.width = dstRoiSize.width;
srcRoiSize.height = 2 * dstRoiSize.height.
```

For proper operation, the function needs valid data outside the source image ROI:

if the *phase* argument is equal to `ippWTFilterFirstLow`, the function requires 2 extra pixels up and 1 pixel down (outside ROI border) for each processed column;

if the *phase* argument is equal to `ippWTFilterFirstHigh`, the function requires 1 extra pixel up and 2 pixels down (outside ROI border) for each processed column.

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if at least one of the specified pointers is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if the width or height of ROI has zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if any of the specified buffer step values is zero or negative.

## WTFwdColLift\_B53\_JPEG2K

*Performs a single step of forward wavelet transform with reversible filter of image columns.*

---

### Syntax

```
IppStatus ippiWTFwdColLift_B53_JPEG2K_16s_C1(const Ipp16s* pSrc0, const Ipp16s* pSrc1,
const Ipp16s* pSrc2, Ipp16s* pDstLow0, const Ipp16s* pSrcHigh0, Ipp16s* pDstHigh1, int
width);
```

```
IppStatus ippiWTFwdColLift_B53_JPEG2K_32s_C1(const Ipp32s* pSrc0, const Ipp32s* pSrc1,
const Ipp32s* pSrc2, Ipp32s* pDstLow0, const Ipp32s* pSrcHigh0, Ipp32s* pDstHigh1, int
width);
```

### Parameters

<i>pSrc0</i>	Pointer to the source image row #0.
<i>pSrc1</i>	Pointer to the source image row #1.
<i>pSrc2</i>	Pointer to the source image row #2.
<i>pSrcHigh0</i>	Pointer to the high frequency component row #0 of a source image.
<i>pDstLow0</i>	Pointer to the low frequency component row #0 of a destination image.

*pDstHigh1* Pointer to the high frequency component row #1 of a destination image.

*width* Width in pixels of rows.

### Description

The function `ippiWTFwdColLift_B53_JPEG2K` is declared in the `ippj.h` file. This function performs a single step of row-scan-based wavelet decomposition of the source image columns using the 5-3 reversible filter.

For each *i*-pixel in the row, this function computes the following values:

$$pDstHigh1[i] = pSrc1[i] - \left\lfloor \frac{pSrc0[i] + pSrc2[i]}{2} \right\rfloor$$

$$pDstLow0[i] = pSrc1[i] + \left\lfloor \frac{pSrcHigh0[i] + pDstHigh1[i] + 2}{4} \right\rfloor$$

Here the notation

$$\left\lfloor x \right\rfloor$$

designates the floor function, that is, the largest integer not exceeding *x*.

### Return Values

`ippStsNoErr` Indicates no error.

`ippStsNullPtrErr` Indicates an error condition if one of the specified pointers is `NULL`.

`ippStsSizeErr` Indicates an error condition if the width of row has zero or negative value.

## WTInvCol\_B53\_JPEG2K

*Performs an inverse wavelet transform with reversible filter of image columns.*

### Syntax

```
IppStatus ippiWTInvCol_B53_JPEG2K_16s_C1R(const Ipp16s* pSrcLow, int srcLowStep, const
Ipp16s* pSrcHigh, int srcHighStep, IppiSize srcRoiSize, Ipp16s* pDst, int dstStep,
IppiWTFilterFirst phase);
```

```
IppStatus ippiWTInvCol_B53_JPEG2K_32s_C1R(const Ipp32s* pSrcLow, int srcLowStep, const
Ipp32s* pSrcHigh, int srcHighStep, IppiSize srcRoiSize, Ipp32s* pDst, int dstStep,
IppiWTFilterFirst phase);
```

## Parameters

<i>pSrcLow</i>	Pointer to ROI of the low frequency component of a source image.
<i>srcLowStep</i>	Distance in bytes between starts of consecutive lines in the low frequency component of a source image.
<i>pSrcHigh</i>	Pointer to ROI of the high frequency component of a source image.
<i>srcHighStep</i>	Distance in bytes between starts of consecutive lines in the high frequency component of a source image.
<i>srcRoiSize</i>	Size of the source image ROI.
<i>pDst</i>	Pointer to ROI of the destination image.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>phase</i>	Relative position of the high-pass and low-pass filters.

## Description

The function `ippiWTInvCol_B53_JPEG2K` is declared in the `ippj.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function performs a wavelet reconstruction of the source image columns using the 5-3 reversible filter. Both source image ROIs have the same size *srcRoiSize*, while the destination image ROI size is uniquely determined from the following relations:

```
dstRoiSize.width = srcRoiSize.width;
dstRoiSize.height = 2 * srcRoiSize.height.
```

For proper operation, the function needs valid data outside the source image ROI.

For the low frequency component:

if the *phase* argument is equal to `ippWTFILTERFIRSTLOW`, the function requires 1 extra pixel down (outside ROI border) for each processed column;

if the *phase* argument is equal to `ippWTFILTERFIRSTHIGH`, the function requires 1 extra pixel up (outside ROI border) for each processed column.

For the high frequency component, the function always requires 1 extra pixel up and 1 pixel down (outside ROI border) for each processed column.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if the width or height of ROI has zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if any of the specified buffer step values is zero or negative.

## WTInvColLift\_B53\_JPEG2K

*Performs a single step of inverse wavelet transform with reversible filter of image columns.*

### Syntax

```
IppStatus ippiWTInvColLift_B53_JPEG2K_16s_C1(const Ipp16s* pSrcLow0, const Ipp16s*
pSrcHigh0, const Ipp16s* pSrcHigh1, const Ipp16s* pSrc0, Ipp16s* pDst1, Ipp16s* pDst2,
int width);
```

```
IppStatus ippiWTInvColLift_B53_JPEG2K_32s_C1(const Ipp32s* pSrcLow0, const Ipp32s*
pSrcHigh0, const Ipp32s* pSrcHigh1, const Ipp32s* pSrc0, Ipp32s* pDst1, Ipp32s* pDst2,
int width);
```

### Parameters

<i>pSrcLow0</i>	Pointer to the source low frequency component row #0.
<i>pSrcHigh0</i>	Pointer to the source high frequency component row #0.
<i>pSrcHigh1</i>	Pointer to the source high frequency component row #1.
<i>pSrc0</i>	Pointer to the reconstructed image row #0.
<i>pDst1</i>	Pointer to the reconstructed image row #1.
<i>pDst2</i>	Pointer to the reconstructed image row #2.
<i>width</i>	Width in pixels of rows.

### Description

The function `ippiWTInvColLift_B53_JPEG2K` is declared in the `ippj.h` file. This function performs a single step of row-scan-based wavelet reconstruction of the source image columns using the 5-3 reversible filter.

For each *i*-pixel in the row, this function computes the following values:

$$pDst2[i] = pSrcLow0[i] - \left\lfloor \frac{pSrcHigh0[i] + pSrcHigh1[i] + 2}{4} \right\rfloor$$

$$pDst1[i] = pSrcHigh0[i] + \left\lfloor \frac{pSrc0[i] + pDst2[i]}{2} \right\rfloor$$

Here the notation

$$\lfloor x \rfloor$$

designates the floor function, that is, the largest integer not exceeding *x*.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if the <i>width</i> is less than or equal to 0.

## WTFwdRow\_D97\_JPEG2K

*Performs a forward wavelet transform with irreversible filter of image rows.*

---

### Syntax

```
IppStatus ippWTFwdRow_D97_JPEG2K<mod>(const Ipp<datatype>* pSrc, int srcStep,
Ipp<datatype>* pDstLow, int dstLowStep, Ipp<datatype>* pDstHigh, int dstHighStep, IppiSize
dstRoiSize, IppiWTFilterFirst phase);
```

Supported values for mod:

16s\_C1R            32s\_C1R            32f\_C1R

### Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDstLow</i>	Pointer to ROI of the low frequency component of a destination image.
<i>dstLowStep</i>	Distance in bytes between starts of consecutive lines in the low frequency component of a destination image.
<i>pDstHigh</i>	Pointer to ROI of the high frequency component of a destination image.
<i>dstHighStep</i>	Distance in bytes between starts of consecutive lines in the high frequency component of a destination image.
<i>dstRoiSize</i>	Size of the destination image ROI.
<i>phase</i>	<a href="#">Relative position of the high-pass and low-pass filters.</a>

### Description

The function `ippWTFwdRow_D97_JPEG2K` is declared in the `ippj.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function performs a wavelet decomposition of the source image rows using the 9-7 irreversible filter. Both destination ROIs have the same size *dstRoiSize*, while the source image ROI size is uniquely determined from the following relations:

```
srcRoiSize.width = 2 * dstRoiSize.width;
srcRoiSize.height = dstRoiSize.height.
```

For proper operation, the function needs valid data outside the source image ROI:

if the *phase* argument is equal to `ippWTFILTERFirstLow`, the function requires 4 extra pixels on the left and 3 pixels on the right (outside ROI border) for each processed row;

if the *phase* argument is equal to `ippWTFILTERFirstHigh`, the function requires 3 extra pixels on the left and 4 pixels on the right (outside ROI border) for each processed row.

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if at least one of the specified pointers is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if the width or height of ROI has zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if any of the specified buffer step values is zero or negative.

## WTInvRow\_D97\_JPEG2K

*Performs an inverse wavelet transform with irreversible filter of image rows.*

### Syntax

```
ippStatus ippiWTInvRow_D97_JPEG2K_<mod>(const Ipp<datatype>* pSrcLow, int srcLowStep,
const Ipp<datatype>* pSrcHigh, int srcHighStep, IppiSize srcRoiSize, Ipp<datatype>* pDst,
int dstStep, IppiWTFILTERFirst phase);
```

Supported values for *mod*:

16s\_C1R      32s\_C1R      32f\_C1R

### Parameters

<i>pSrcLow</i>	Pointer to ROI of the low frequency component of a source image.
<i>srcLowStep</i>	Distance in bytes between starts of consecutive lines in the low frequency component of a source image.
<i>pSrcHigh</i>	Pointer to ROI of the high frequency component of a source image.
<i>srcHighStep</i>	Distance in bytes between starts of consecutive lines in the high frequency component of a source image.
<i>srcRoiSize</i>	Size of the source image ROI.
<i>pDst</i>	Pointer to ROI of the destination image.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>phase</i>	Relative position of the high-pass and low-pass filters.

## Description

The function `ippiWTInvRow_D97_JPEG2K` is declared in the `ippj.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function performs a wavelet reconstruction of the source image rows using the 9-7 irreversible filter. Both source ROIs have the same size `srcRoiSize`, while the destination image ROI size is uniquely determined from the following relations:

```
dstRoiSize.width = 2 * srcRoiSize.width;
```

```
dstRoiSize.height = srcRoiSize.height.
```

For proper operation, the function needs valid data outside the source image ROI.

For the low frequency component:

if the *phase* argument is equal to `ippWTFilterFirstLow`, the function requires 1 extra pixel on the left and 2 extra pixels on the right (outside ROI border) for each processed row;

if the *phase* argument is equal to `ippWTFilterFirstHigh`, the function requires 2 extra pixels on the left and 1 extra pixel on the right (outside ROI border) for each processed row.

For the high frequency component, the function always requires 2 extra pixels on the left and 2 extra pixels on the right (outside ROI border) for each processed row.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if at least one of the specified pointers is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if the width or height of ROI has zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if any of the specified buffer step values is zero or negative.

## WTFwdCol\_D97\_JPEG2K

*Performs a forward wavelet transform with irreversible filter of image columns.*

---

## Syntax

```
IppStatus ippiWTFwdCol_D97_JPEG2K_32f_C1R(const Ipp32f* pSrc, int srcStep, Ipp32f* pDstLow,
int dstLowStep, Ipp32f* pDstHigh, int dstHighStep, IppiSize dstRoiSize, IppiWTFilterFirst
phase);
```

## Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.



<i>pDstLow</i>	Pointer to ROI of the low frequency component of a destination image.
<i>dstLowStep</i>	Distance in bytes between starts of consecutive lines in the low frequency component of a destination image.
<i>pDstHigh</i>	Pointer to ROI of the high frequency component of a destination image.
<i>dstHighStep</i>	Distance in bytes between starts of consecutive lines in the high frequency component of a destination image.
<i>dstRoiSize</i>	Size of the destination image ROI.
<i>phase</i>	Relative position of the high-pass and low-pass filters.

## Description

The function `ippiWTFwdCol_D97_JPEG2K` is declared in the `ippj.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function performs a wavelet decomposition of the source image columns using the 9-7 irreversible filter. Both destination ROIs have the same size *dstRoiSize*, while the source image ROI size is uniquely determined from the following relations:

```
srcRoiSize.width = dstRoiSize.width;
srcRoiSize.height = 2 * dstRoiSize.height.
```

For proper operation, the function needs valid data outside the source image ROI:

if the *phase* argument is equal to `ippWTFilterFirstLow`, the function requires 4 extra pixels up and 3 extra pixels down (outside ROI border) for each processed column;

if the *phase* argument is equal to `ippWTFilterFirstHigh`, the function requires 3 extra pixels up and 4 extra pixels down (outside ROI border) for each processed column.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if at least one of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if the width or height of ROI has zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if any of the specified buffer step values is zero or negative.

## WTFwdColLift\_D97\_JPEG2K

*Performs a single step of forward wavelet transform with irreversible filter of image columns.*

---

### Syntax

```
IppStatus ippiWTFwdColLift_D97_JPEG2K<mod>(const Ipp<datatype>* pSrc0, const Ipp<datatype>* pSrc1, const Ipp<datatype>* pSrc2, Ipp<datatype>* pSrcDstLow0, Ipp<datatype>* pDstLow1, Ipp<datatype>* pSrcDstHigh0, Ipp<datatype>* pSrcDstHigh1, Ipp<datatype>* pDstHigh2, int width);
```

Supported values for mod:

16s\_C1                  32s\_C1                  32f\_C1

### Parameters

<i>pSrc0</i>	Pointer to the source image row #0.
<i>pSrc1</i>	Pointer to the source image row #1.
<i>pSrc2</i>	Pointer to the source image row #2.
<i>pSrcDstLow0</i>	Pointer to the low frequency component row #0.
<i>pDstLow1</i>	Pointer to the low frequency component row #1.
<i>pSrcDstHigh0</i>	Pointer to the high frequency component row #0.
<i>pSrcDstHigh1</i>	Pointer to the high frequency component row #1.
<i>pDstHigh2</i>	Pointer to the high frequency component row #2.
<i>width</i>	Width in pixels of rows.

### Description

The function `ippiWTFwdColLift_D97_JPEG2K` is declared in the `ippj.h` file. This function performs a single step of row-scan-based 2D wavelet decomposition of the source image columns using the 9-7 irreversible filter.

For each *i*-pixel in the row this function computes the following values:

$$\begin{aligned}
pDstHigh2[i] &= pSrc1[i] + \alpha \cdot (pSrc0[i] + pSrc2[i]) \\
pDstLow1[i] &= pSrc0[i] + \beta \cdot (pSrcDstHigh1[i] + pDstHigh2[i]) \\
pSrcDstHigh1[i] &= pSrcDstHigh1[i-1] + \gamma \cdot (pSrcDstLow0[i] + pDstLow1[i]) \\
pSrcDstLow0[i] &= pSrcDstLow0[i-1] + \delta \cdot (pSrcDstHigh0[i] + pSrcDstHigh1[i]) \\
pSrcDstLow0[i] &= pSrcDstLow0[i] \cdot K \\
pSrcDstHigh0[i] &= (pSrcDstHigh0[i]) / K
\end{aligned}$$

where  $\alpha$ ,  $\beta$ ,  $\gamma$ ,  $\delta$ , and  $K$  are the lifting parameters in accordance with [ISO15444], Annex F.3, Inverse Discrete Wavelet Transformation.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if the <i>width</i> is less than or equal to 0.

## WTInvCol\_D97\_JPEG2K

*Performs an inverse wavelet transform with irreversible filter of image columns.*

### Syntax

```
ippStatus ippWTInvCol_D97_JPEG2K_32f_C1R(const Ipp32f* pSrcLow, int srcLowStep, const
Ipp32f* pSrcHigh, int srcHighStep, IppiSize srcRoiSize, Ipp32f* pDst, int dstStep,
IppiWTFilterFirst phase);
```

### Parameters

<i>pSrcLow</i>	Pointer to ROI of the low frequency component of a source image.
<i>srcLowStep</i>	Distance in bytes between starts of consecutive lines in the low frequency component of a source image.
<i>pSrcHigh</i>	Pointer to ROI of the high frequency component of a source image.
<i>srcHighStep</i>	Distance in bytes between starts of consecutive lines in the high frequency component of a source image.
<i>srcRoiSize</i>	Size of the source image ROI.
<i>pDst</i>	Pointer to ROI of the destination image.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>phase</i>	Relative position of the high-pass and low-pass filters.

### Description

The function `ippWTInvCol_D97_JPEG2K` is declared in the `ippj.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function performs a wavelet reconstruction of the source image columns using the 9-7 irreversible filter. Both source image ROIs have the same size *srcRoiSize*, while the destination image ROI size is uniquely determined from the following relations:

```
dstRoiSize.width = srcRoiSize.width;
dstRoiSize.height = 2 * srcRoiSize.height.
```

For proper operation, the function needs valid data outside the source image ROI.

For the low frequency component:

if the *phase* argument is equal to `ippWTFILTERFIRSTLOW`, the function requires 1 extra pixel up and 2 extra pixels down (outside ROI border) for each processed column;

if the *phase* argument is equal to `ippWTFILTERFIRSHIGH`, the function requires 2 extra pixels up and 1 extra pixel down (outside ROI border) for each processed column.

For the high frequency component, the function always requires 2 extra pixels up and 2 extra pixels down (outside ROI border) for each processed column.

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if at least one of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if the width or height of ROI has zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if any of the specified buffer step values is zero or negative.

## WTInvColLift\_D97\_JPEG2K

*Performs a single step of inverse wavelet transform with irreversible filter of image columns.*

---

### Syntax

```
IppStatus ippiWTInvColLift_D97_JPEG2K_<mod>(const Ipp<datatype>* pSrcLow0, const
Ipp<datatype>* pSrcHigh0, const Ipp<datatype>* pSrcHigh1, const Ipp<datatype>* pSrc0,
Ipp<datatype>* pSrcDst1, Ipp<datatype>* pSrcDst2, Ipp<datatype>* pDst3, Ipp<datatype>*
pDst4, int width);
```

Supported values for `mod`:

16s\_C1            32s\_C1            32f\_C1

### Parameters

<code>pSrcLow0</code>	Pointer to the source low frequency component row #0.
<code>pSrcHigh0</code>	Pointer to the source high frequency component row #0.
<code>pSrcHigh1</code>	Pointer to the source high frequency component row #1.
<code>pSrc0</code>	Pointer to the reconstructed image row #0.
<code>pSrcDst1</code>	Pointer to the reconstructed image row #1.
<code>pSrcDst2</code>	Pointer to the reconstructed image row #2.
<code>pDst3</code>	Pointer to the reconstructed image row #3.
<code>pDst4</code>	Pointer to the reconstructed image row #4.
<code>width</code>	Width in pixels of rows.

## Description

The function `ippiWTInvColLift_D97_JPEG2K` is declared in the `ippj.h` file. This function performs a single step of row-scan-based 2D wavelet reconstruction of the source image columns using the 9-7 irreversible filter. For each  $i$ -pixel in the row, this function computes the following values:

$$\begin{aligned} pDst4[i] &= pSrcLow0[i] \cdot K - (\delta / K \cdot (pSrcHigh0[i] + pSrcHigh1[i])) \\ pDst3[i] &= (pSrcHigh0[i]) / K - (\gamma \cdot (pSrcDst2[i] + pDst4[i])) \\ pSrcDst2[i] &= pSrcDst2[i-1] - (\beta \cdot (pSrcDst1[i] + pDst3[i])) \\ pSrcDst1[i] &= pSrcDst1[i-1] - (\alpha \cdot (pSrc0[i] + pSrcDst2[i])) \end{aligned}$$

where  $\alpha$ ,  $\beta$ ,  $\gamma$ ,  $\delta$ , and  $K$  are the lifting parameters in accordance with [ISO15444], Annex F.3, *Inverse Discrete Wavelet Transformation*.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if the <i>width</i> is less than or equal to 0.

## Tile-oriented Transforms

These functions perform high-level tile-oriented reversible ( `_B53` modifier) and irreversible ( `_D97` modifier) wavelet transforms for lossless and lossy compressions, respectively, in accordance with [ISO15444], Annex F, *Discrete Wavelet Transformation of Tile-Components*. The transformations are the lifting-based implementations of filtering by 5-3 reversible and 9-7 irreversible wavelet filters.

The functions operate with tile-component and perform one-level decomposition or reconstruction of the image. Each tile-component is transformed into a set of four two-dimensional subbands.

## WTGetBufSize\_B53\_JPEG2K

*Computes the size of the buffer for wavelet transform with reversible filter.*

## Syntax

```

IppStatus ippiWTGetBufSize_B53_JPEG2K_16s_C1R(const IppiRect* pTileRect, int* pSize);
IppStatus ippiWTGetBufSize_B53_JPEG2K_32s_C1R(const IppiRect* pTileRect, int* pSize);
IppStatus ippiWTGetBufSize_B53_JPEG2K_16s_C1IR(const IppiRect* pTileRect, int* pSize);
IppStatus ippiWTGetBufSize_B53_JPEG2K_32s_C1IR(const IppiRect* pTileRect, int* pSize);

```

## Parameters

<code>pTileRect</code>	Pointer to the tile rectangle structure.
<code>pSize</code>	Pointer to the computed value of the buffer.

## Description

The function `ippiWTFwdBufSize_B53_JPEG2K` is declared in the `ippj.h` file. This function computes the size in bytes of the work buffer that is required for a tile-oriented wavelet transform functions `ippiWTFwd_B53_JPEG2K` and `ippiWTInv_B53_JPEG2K`.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if any of the specified pointers is <code>NULL</code> .

## WTFwd\_B53\_JPEG2K

*Performs a tile-oriented forward wavelet transform with reversible filter.*

---

## Syntax

```
IppStatus ippiWTFwd_B53_JPEG2K_16s_C1R(const Ipp16s* pSrc, int srcStep, const IppiRect* pTileRect, Ipp16s* pDst[4], int dstStep[4], Ipp8u* pBuffer);
```

```
IppStatus ippiWTFwd_B53_JPEG2K_32s_C1R(const Ipp32s* pSrc, int srcStep, const IppiRect* pTileRect, Ipp32s* pDst[4], int dstStep[4], Ipp8u* pBuffer);
```

```
IppStatus ippiWTFwd_B53_JPEG2K_16s_C1IR(Ipp16s* pSrcDstTile, int srcDstStep, const IppiRect* pTileRect, Ipp8u* pBuffer);
```

```
IppStatus ippiWTFwd_B53_JPEG2K_32s_C1IR(Ipp32s* pSrcDstTile, int srcDstStep, const IppiRect* pTileRect, Ipp8u* pBuffer);
```

## Parameters

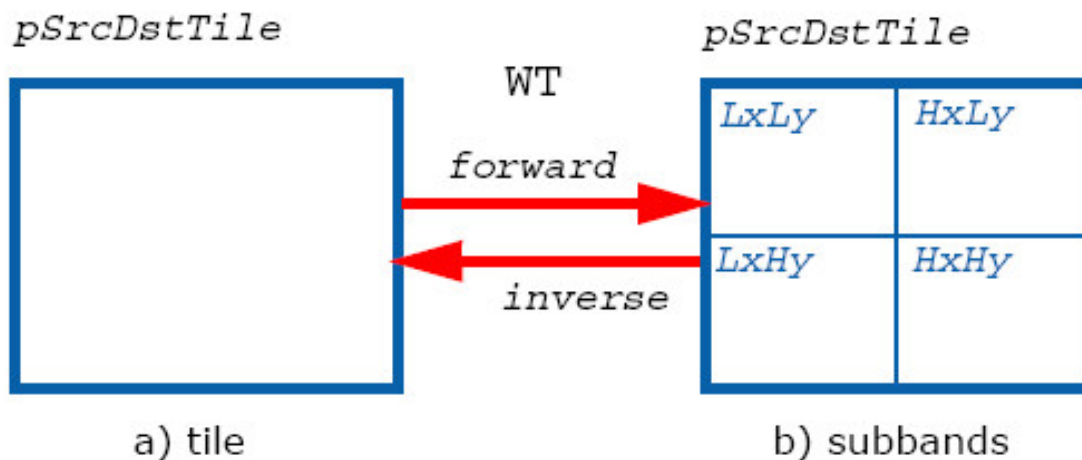
<code>pSrc</code>	Pointer to the source image.
<code>srcStep</code>	Distance in bytes between starts of consecutive lines in the source image.
<code>pTileRect</code>	Pointer to the tile rectangle structure.
<code>pDst</code>	Array of pointers to the subbands of the destination image (in the order <code>LxLy</code> , <code>LxHy</code> , <code>HxLy</code> , <code>HxHy</code> ).
<code>dstStep</code>	Array of distances in bytes between starts of consecutive lines in the the subbands of the destination image.
<code>pSrcDstTile</code>	Pointer to the source and destination image for in-place operation.
<code>srcDstStep</code>	Distance in bytes between starts of consecutive lines in the source and destination image for in-place operation.
<code>pBuffer</code>	Pointer to the work buffer.

## Description

The function `ippiWTFwd_B53_JPEG2K` is declared in the `ippj.h` file. This function performs a forward discrete wavelet transformation of the tile-component using a subband decomposition in accordance with [ISO15444], Annex F, *Discrete Wavelet Transformation of Tile-Components*. The function performs one-level decomposition by downsampling each tile of the source image `pSrc` (`pSrcDstTile` for in-place flavors) into a set of four two-dimensional subbands labelled as `LxLy`, `LxHy`, `HxLy`, `HxHy`. The subband `LxLy` is a downsampled version of source tile that is low-pass filtered horizontally and low-pass filtered vertically. The subband `LxHy` is a downsampled version of source tile that is low-pass filtered horizontally and high-pass filtered vertically. The subband `HxLy` is a downsampled version of source tile that is high-pass filtered horizontally and low-pass filtered vertically. The subband `HxHy` is a downsampled version of source tile that is high-pass filtered horizontally and high-pass filtered vertically. The function uses the 5-3 reversible filter.

For not-in-place flavors, the subbands are stored separately in the array `pDst`. For in-place flavors, the subbands are stored in the `pSrcDstTile` (see Figure “Subband Positions for in-place Operations”).

## Subband Positions for in-place Operations



The size of the required external buffer should be determined by calling the `ippiWTGetBufSize_B53_JPEG2K` function prior to using the wavelet transform function.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if any of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if the width or height of ROI has zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if any of the specified step values is zero or negative.

## WTInv\_B53\_JPEG2K

*Performs a tile-oriented inverse wavelet transform with reversible filter.*

---

### Syntax

```

IppStatus ippiWTInv_B53_JPEG2K_16s_C1R(const Ipp16s* pSrc[4], int srcStep[4], Ipp16s*
pDst, int dstStep, const IppiRect* pTileRect, Ipp8u* pBuffer);

IppStatus ippiWTInv_B53_JPEG2K_32s_C1R(const Ipp32s* pSrc[4], int srcStep[4], Ipp32s*
pDst, int dstStep, const IppiRect* pTileRect, Ipp8u* pBuffer);

IppStatus ippiWTInv_B53_JPEG2K_16s_C1IR(Ipp16s* pSrcDstTile, int srcDstStep, const IppiRect*
pTileRect, Ipp8u* pBuffer);

IppStatus ippiWTInv_B53_JPEG2K_32s_C1IR(Ipp32s* pSrcDstTile, int srcDstStep, const IppiRect*
pTileRect, Ipp8u* pBuffer);

```

### Parameters

<i>pSrc</i>	An array of pointers to the subbands of the source image (in the order LxLy, LxHy, HxLy, HxHy).
<i>srcStep</i>	Array of distances in bytes between starts of consecutive lines in the subbands of the source image.
<i>pDst</i>	Pointer to the destination image.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>pSrcDstTile</i>	Pointer to the source and destination image for in-place operation.
<i>srcDstStep</i>	Distance in bytes between starts of consecutive lines in the source and destination image for in-place operation.
<i>pTileRect</i>	Pointer to the tile rectangle structure.
<i>pBuffer</i>	Pointer to the work buffer.

### Description

The function `ippiWTInv_B53_JPEG2K` is declared in the `ippj.h` file. This function performs an inverse discrete wavelet transformation using a subband reconstruction in accordance with [ISO15444], Annex F, *Discrete Wavelet Transformation of Tile-Components*. The function performs one-level reconstruction of the tile of the destination image *pDst* from the set of four two-dimensional subbands labelled as LxLy, LxHy, HxLy, HxHy. The subband LxLy is a downsampled version of the initial tile that is low-pass filtered horizontally and low-pass filtered vertically. The subband LxHy is a downsampled version of the initial tile that is low-pass filtered horizontally and high-pass filtered vertically. The subband HxLy is a downsampled version of the initial tile that is high-pass filtered horizontally and low-pass filtered vertically. The subband HxHy is a downsampled version of the initial tile that is high-pass filtered horizontally and high-pass filtered vertically. The function uses the 5-3 reversible filter.

For not-in-place flavors, the source subbands are placed separately in the array *pDst*. For in-place flavors, the source subbands are stored in the *pSrcDstTile* (see [Figure "Subband Positions for in-place Operations"](#)).



The size of the required external buffer should be determined by calling the [ippiWTGetBufSize\\_B53\\_JPEG2K](#) function prior to using the wavelet transform function.

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if any of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if the width or height of ROI has zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if any of the specified step values is zero or negative.

## WTGetBufSize\_D97\_JPEG2K

*Computes the size of the buffer wavelet transform with irreversible filter*

---

### Syntax

```
IppStatus ippiWTGetBufSize_D97_JPEG2K_16s_C1R(const IppiRect* pTileRect, int* pSize);
IppStatus ippiWTGetBufSize_D97_JPEG2K_32s_C1R(const IppiRect* pTileRect, int* pSize);
IppStatus ippiWTGetBufSize_D97_JPEG2K_16s_C1IR(const IppiRect* pTileRect, int* pSize);
IppStatus ippiWTGetBufSize_D97_JPEG2K_32s_C1IR(const IppiRect* pTileRect, int* pSize);
```

### Parameters

<code>pTileRect</code>	Pointer to the tile rectangle structure.
<code>pSize</code>	Pointer to the computed value of the buffer.

### Description

The function `ippiWTGetBufSize_D97_JPEG2K` is declared in the `ippj.h` file. This function computes the size in bytes of the work buffer that is required for a tile-oriented irreversible wavelet transform functions [ippiWTFwd\\_D97\\_JPEG2K](#) and [ippiWTInv\\_D97\\_JPEG2K](#).

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if at least one of the specified pointers is <code>NULL</code> .

## WTFwd\_D97\_JPEG2K

*Performs a tile-oriented forward wavelet transform with irreversible filter.*

---

### Syntax

```

IppStatus ippiWTFwd_D97_JPEG2K_16s_C1R(const Ipp16s* pSrc, int srcStep, const IppiRect*
pTileRect, Ipp16s* pDst[4], int dstStep[4], Ipp8u* pBuffer);

IppStatus ippiWTFwd_D97_JPEG2K_32s_C1R(const Ipp32s* pSrc, int srcStep, const IppiRect*
pTileRect, Ipp32s* pDst[4], int dstStep[4], Ipp32s* pBuffer);

IppStatus ippiWTFwd_D97_JPEG2K_16s_C1IR(Ipp16s* pSrcDstTile, int srcDstStep, const IppiRect*
pTileRect, Ipp8u* pBuffer);

IppStatus ippiWTFwd_D97_JPEG2K_32s_C1IR(Ipp32s* pSrcDstTile, int srcDstStep, const IppiRect*
pTileRect, Ipp8u* pBuffer);

```

### Parameters

<i>pSrc</i>	Pointer to the source image.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pTileRect</i>	Pointer to the tile rectangle structure.
<i>pDst</i>	Array of pointers to the subbands of the destination image (in the order LxLy, LxHy, HxLy, HxHy).
<i>dstStep</i>	Array of distances in bytes between starts of consecutive lines in the subbands of the destination image.
<i>pSrcDstTile</i>	Pointer to the source and destination image for in-place operation.
<i>srcDstStep</i>	Distance in bytes between starts of consecutive lines in the source and destination image for in-place operation.
<i>pBuffer</i>	Pointer to the buffer.

### Description

The function `ippiWTFwd_D97_JPEG2K` is declared in the `ippj.h` file. This function performs a forward discrete wavelet transformation of the tile-component using a subband decomposition in accordance with [ISO15444], Annex F, *Discrete Wavelet Transformation of Tile-Components*. The function performs one-level decomposition by downsampling each tile of the source image *pSrc* into a set of four two-dimensional subbands *pDst* labelled as LxLy, LxHy, HxLy, HxHy. The subband LxLy is a downsampled version of the source tile to which the horizontal and vertical low-pass filters were applied. The subband LxHy is a downsampled version of the source tile to which a horizontal low-pass and vertical and high-pass filters were applied. The subband HxLy is a downsampled version of the source tile to which the horizontal and vertical high-pass filters were applied. The subband HxHy is a downsampled version of the source tile to which horizontal high-pass and vertical high-pass filters were applied. The function uses the 9-7 irreversible filter.

For not-in-place flavors, the subbands are stored separately in the array *pDst*. For in-place flavors, the subbands are stored in the *pSrcDstTile* (see Figure “Subband Positions for in-place Operations”).

The size of the required external buffer should be determined by calling the [ippiWTGetBufSize\\_D97\\_JPEG2K](#) function prior to using the wavelet transform function.

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if any of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if the width or height of ROI has zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if any of the specified step values is zero or negative.

## WTInv\_D97\_JPEG2K

*Performs a tile-oriented inverse wavelet transform with irreversible filter.*

### Syntax

```

IppStatus ippiWTInv_D97_JPEG2K_16s_C1R(const Ipp16s* pSrc[4], int srcStep[4], Ipp16s*
pDst, int dstStep, const IppiRect* pTileRect, Ipp8u* pBuffer);

IppStatus ippiWTInv_D97_JPEG2K_32s_C1R(const Ipp32s* pSrc[4], int srcStep[4], Ipp32s*
pDst, int dstStep, const IppiRect* pTileRect, Ipp8u* pBuffer);

IppStatus ippiWTInv_D97_JPEG2K_16s_C1IR(Ipp16s* pSrcDstTile, int srcDstStep, const IppiRect*
pTileRect, Ipp8u* pBuffer);

IppStatus ippiWTInv_D97_JPEG2K_32s_C1IR(Ipp32s* pSrcDstTile, int srcDstStep, const IppiRect*
pTileRect, Ipp8u* pBuffer);

```

### Parameters

<code>pSrc</code>	An array of pointers to the source subbands (in the order LxLy, LxHy, HxLy, HxHy).
<code>srcStep</code>	Array of distances in bytes between starts of consecutive lines in the the source subbands.
<code>pDst</code>	Pointer to the destination image.
<code>dstStep</code>	Distance in bytes between starts of consecutive lines in the the destination image.
<code>pSrcDstTile</code>	Pointer to the source and destination image for in-place operation.
<code>srcDstStep</code>	Distance in bytes between starts of consecutive lines in the source and destination image for in-place operation.
<code>pTileRect</code>	Pointer to the tile rectangle structure.
<code>pBuffer</code>	Pointer to the buffer.

## Description

The function `ippiWTInv_D97_JPEG2K` is declared in the `ippj.h` file. This function performs an inverse discrete wavelet transformation using a subband reconstruction in accordance with [ISO15444], *Annex F, Discrete Wavelet Transformation of Tile-Components*. The function performs one-level reconstruction of the tile of the destination image `pDst` from the set of four two-dimensional subbands `pSrc` labelled as `LxLy`, `LxHy`, `HxLy`, `HxHy`. The subband `LxLy` is a downsampled version of the initial tile to which horizontal and vertical low-pass filters were applied. The subband `LxHy` is a downsampled version of the initial tile to which a horizontal low-pass and vertical high-pass filters were applied. The subband `HxLy` is a downsampled version of the initial tile to which horizontal high-pass and vertical low-pass filters were applied. The subband `HxHy` is a downsampled version of the initial tile to which horizontal and vertical high-pass filters were applied. The function uses the 9-7 irreversible filter.

For not-in-place flavors, the source subbands are placed separately in the array `pDst`. For in-place flavors, the source subbands are placed in the `pSrcDstTile` (see Figure “Subband Positions for in-place Operations”).

The size of the required external buffer should be determined by calling the `ippiWTGetBufSize_D97_JPEG2K` function prior to using the wavelet transform function.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if any of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if the width or height of ROI has zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if any of the specified step values is zero or negative.

# JPEG2000 Entropy Coding and Decoding Functions

This section describes the image processing functions that perform arithmetic encoding and decoding procedures and are specific for JPEG 2000 image coding system. These functions are listed in the table below.

## Entropy Coder Functions

Function Base Name	Description
<code>EncodeInitAlloc_JPEG2K</code>	Allocates memory and initializes an entropy encoder state structure
<code>EncodeFree_JPEG2K</code>	Deallocates memory allocated for an entropy encoding state structure
<code>EncodeLoadCodeBlock_JPEG2K</code>	Loads code block and prepares data for entropy encoding
<code>EncodeStoreBits_JPEG2K</code>	Produces output codestream
<code>EncodeGetTermPassLen_JPEG2K</code>	Returns the length of the specified terminated coding pass.
<code>EncodeGetRate_JPEG2K</code>	Returns estimated target bit rate for the specified coding pass
<code>EncodeGetDist_JPEG2K</code>	Returns estimated distortion of the image for the specified coding pass
<code>DecodeGetBufSize_JPEG2K</code>	Computes the size of the working buffer for decoding routine
<code>DecodeCodeBlock_JPEG2K</code>	Decodes the compressed code block data
<b>Decoder Functions for Progressive Mode</b>	

Function Base Name	Description
<a href="#">DecodeCBProgrGetStateSize_JPEG2K</a>	Computes the size of the external buffer for the decoder state structure.
<a href="#">DecodeCBProgrInit_JPEG2K</a>	Initializes the decoder state structure in the external buffer/
<a href="#">DecodeCBProgrInitAlloc_JPEG2K</a>	Allocates memory and initializes the decoder state structure in the external buffer.
<a href="#">DecodeCBProgrFree_JPEG2K</a>	Frees memory allocated for the decoder state structure.
<a href="#">DecodeCBProgrAttach_JPEG2K</a>	Attaches the buffer with data for progressive decoding.
<a href="#">DecodeCBProgrSetPassCounter_JPEG2K</a>	Sets the value of internal coding pass counter
<a href="#">DecodeCBProgrGetPassCounter_JPEG2K</a>	Returns the value of the internal coding pass counter.
<a href="#">DecodeCBProgrGetCurBitPlane_JPEG2K</a>	Returns the number of the current bit plane
<a href="#">DecodeCBProgrStep_JPEG2K</a>	Performs a single step of decoding.

The implemented adaptive arithmetic encoding is performed as two successive procedures. First, the prepared code block is pre-processed by the function [ippiEncodeLoadCodeBlock\\_JPEG2K](#). After that, the function [ippiEncodeStoreBits\\_JPEG2K](#) performs final stages of encoding and produces the output codestream. In the course of encoding, the state structure *pState* is used, which is allocated and initialized by the function [ippiEncodeInitAlloc\\_JPEG2K](#).

## EncodeInitAlloc\_JPEG2K

*Allocates memory and initializes an entropy encoder state structure.*

### Syntax

```
IppStatus ippiEncodeInitAlloc_JPEG2K(IppiEncodeState_JPEG2K** pState, IppiSize codeBlockMaxSize);
```

### Parameters

<i>pState</i>	Pointer to variable that returns the pointer to allocated and initialized encoder state structure.
<i>codeBlockMaxSize</i>	Maximum size of code block in pixels.

### Description

The function [ippiEncodeInitAlloc\\_JPEG2K](#) is declared in the `ippj.h` file. This function allocates and initializes the entropy encoder state structure that is used in the encoding procedures.

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsSizeErr</code>	Indicates an error condition if the width or height of the code block has zero or negative value.
<code>ippStsMemAllocErr</code>	Indicates an error condition if memory allocation fails.

## EncodeFree\_JPEG2K

*Frees memory allocated for an entropy encoding state structure.*

---

### Syntax

```
IppStatus ippiEncodeFree_JPEG2K(IppiEncodeState_JPEG2K* pState);
```

### Parameters

*pState*                                      Pointer to the allocated and initialized encoder state structure.

### Description

The function `ippiEncodeFree_JPEG2K` is declared in the `ippj.h` file. This function frees memory allocated for the entropy encoder state structure by the function `ippiEncodeInitAlloc_JPEG2K`.

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pState</i> pointer to a state structure is NULL.
<code>ippStsContextMatchErr</code>	Indicates an error condition if a pointer to an invalid state structure is passed.

## EncodeLoadCodeBlock\_JPEG2K

*Loads a code block and prepares data for entropy encoding.*

---

### Syntax

```
IppStatus ippiEncodeLoadCodeBlock_JPEG2K_32s_C1R(const Ipp32s* pSrc, int srcStep, IppiSize codeBlockSize, IppiEncodeState_JPEG2K* pState, IppiWTSubband subband, int magnBits, IppiMQTermination mqTermType, IppiMQRateAppr mqRateAppr, int codeStyleFlags, int* pSfBits, int* pNoFPasses, int* pNoFTermPasses);
```

### Parameters

<i>pSrc</i>	Pointer to the source code block.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the code block.
<i>codeBlockSize</i>	Size of the code block.
<i>pState</i>	Pointer to allocated and initialized encoder state structure.
<i>subband</i>	Wavelet transform subband contained the given code block. Possible values are listed in the <a href="#">Table "Subband Parameter for the JPEG2000 Entropy Coder Functions"</a> .
<i>magnBits</i>	Non-fractional bit count in integer representation.

<i>codeStyleFlags</i>	Specifies the options for encoding procedure. The possible values are listed in the <a href="#">Table "CodeStyleFlag Argument for the JPEG2000 Entropy Coder Functions"</a> .
<i>mqTermType</i>	Termination mode for MQ-coder. Possible values are listed in the <a href="#">Table "MqTermType Argument for the JPEG2000 Entropy Coder Functions"</a> .
<i>mqRateAppr</i>	Specifies the bit rate estimation model. Only one value, <code>ippMQRateApprGood</code> , that sets the non-optimal approximation model is available now.
<i>pSfBits</i>	Pointer to the variable that returns the number of significant bit planes.
<i>pNOFPasses</i>	Pointer to the variable that returns the number of coding passes.
<i>pNOFTermPasses</i>	Pointer to the variable that returns the number of terminated coding passes.

## Description

The function `ippiEncodeLoadCodeBlock_JPEG2K` is declared in the `ippj.h` file. This function performs the first encoding procedure, that is, loads the specified code block, computes required parameters of the MQ-coder, and prepares data for the second encoding procedure. The wavelet transform subband should be specified in the *subband* parameter. Possible values are listed in the [Table "Subband Parameter for the JPEG2000 Entropy Coder Functions"](#).

### Subband Parameter for the JPEG2000 Entropy Coder Functions

Value	Description
<code>ippWTSubbandLxLy</code>	Subband obtained by low-pass filtering along x and y directions
<code>ippWTSubbandLxHy</code>	Subband obtained by low-pass filtering along x and high-pass filtering along y direction
<code>ippWTSubbandHxLy</code>	Subband obtained by high-pass filtering along x and low-pass filtering along y direction
<code>ippWTSubbandHxHy</code>	Subband obtained by high-pass filtering along x and y directions

### CodeStyleFlag Argument for the JPEG2000 Entropy Coder Functions

Value	Description
<code>IPP_JPEG2K_VERTICALLY_CAUSAL_CONTEXT</code>	Sets the creation of vertically causal context
<code>IPP_JPEG2K_SELECTIVE_MQ_BYPASS</code>	Sets the selective MQ encoding bypass, which is raw encoding for some coding passes
<code>IPP_JPEG2K_TERMINATE_ON_EVERY_PASS</code>	Sets the termination of MQ coder after every coding pass
<code>IPP_JPEG2K_RESETCTX_ON_EVERY_PASS</code>	Sets the reset of MQ coder context after each coding pass
<code>IPP_JPEG2K_USE_SEGMENTATION_SYMBOLS</code>	Uses the segmentation symbol sequence for error resilience
<code>IPP_JPEG2K_LOSSLESS_MODE</code>	Indicates that the lossless wavelet transforms are used in the rate-distortion estimation
<code>IPP_JPEG2K_DEC_CONCEAL_ERRORS</code>	Allows the error concealment in the last bit-plane where the errors were detected
<code>IPP_JPEG2K_DEC_DO_NOT_CLEAR_CB</code>	Does not clear the code block data before decoding
<code>IPP_JPEG2K_DEC_DO_NOT_RESET_LOW_BITS</code>	Does not reset low-order bits after decoding

Value	Description
IPP_JPEG2K_DEC_DO_NOT_CLEAR_SFBUFFER	Does not clear the buffer before decoding to keep the previous significance state
IPP_JPEG2K_DEC_CHECK_PRED_TERM	Verifies during decoding if the predictable termination is correct; if not - "damage in codeblock" error code is generated. It should be used in predictable termination mode.

### MqTermType Argument for the JPEG2000 Entropy Coder Functions

Value	Description
ippMQTermSimple	Simple termination - some extra bytes are added
ippMQTermNearOptimal	Near optimal termination mode
ippMQTermPredictable	Termination mode for predictable error resilience

The variable *sfBits* returns the number of significant bit planes only. For example, if all source pixels are non-negative and their maximum value is 0xA (binary 1010), then *sfBits* should return 4 significant bits. The higher bits are not coded.

Only significant non-fractional bits in integer representation are coded. *magnBits* specifies the number of non-fractional bits. For example, if *magnBits* = 11, the 20 (31-11) least significant bits will not be coded.

Negative integers in the code block should be presented in direct code with the sign in the most significant bit (31<sup>st</sup> in the zero-based numeration, when the least significant bit has a zero index). This code may be effectively obtained using the specially designed function [ippiComplement](#).

### Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when any of the specified pointers is <code>NULL</code> .
ippStsSizeErr	Indicates an error condition if the width or height of the code block has zero or negative value.
ippStsContextMatchErr	Indicates an error condition if a pointer to an invalid state structure is passed.
ippStsStepErr	Indicates an error condition if <i>srcStep</i> has zero or negative value.

## EncodeStoreBits\_JPEG2K

*Produces output codestream.*

### Syntax

```
IppStatus ippiEncodeStoreBits_JPEG2K_lu(Ipp8u* pDst, int* pDstLen, IppiEncodeState_JPEG2K* pState, int* pIsNotFinish);
```

### Parameters

<i>pDst</i>	Pointer to the destination data buffer.
<i>pDstLen</i>	Pointer to the destination buffer length.
<i>pState</i>	Pointer to the allocated and initialized encoder state structure.
<i>pIsNotFinish</i>	Pointer to the variable indicating that the encoding is completed.



## Description

The function `ippiEncodeStoreBits_JPEG2K` is declared in the `ippj.h` file. This function performs the second encoding procedure and produces the output codestream.

## Application Notes

The parameter `dstLen` is used both for reading and writing. It should be passed to the function with a valid buffer length, and then it will return the length of used (filled) buffer.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when any of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if the width or height of the destination buffer has zero or negative value.
<code>ippStsContextMatchErr</code>	Indicates an error condition if a pointer to an invalid state structure is passed.

## Example

The code example below shows how this function may be used.

```
int isNotFinish = 1;
while(isNotFinish)
{
    int len = BUFFER_LEN;
    ippiEncodeStoreBits_JPEG2K_lu(buffer, &len, state, &isNotFinish);
    // You can write compressed data to the output stream, for example:
    // fwrite(buffer, sizeof(Ipp8u), len, file);
}
```

## EncodeGetTermPassLen\_JPEG2K

*Returns the length of the specified terminated coding pass.*

## Syntax

```
IppStatus ippiEncodeGetTermPassLen_JPEG2K(IppiEncodeState_JPEG2K* pState, int passNumber,
int* pPassLen);
```

## Parameters

<code>pState</code>	Pointer to the allocated and initialized encoder state structure.
<code>passNumber</code>	Number of the terminated coding pass.
<code>pPassLen</code>	Pointer to the variable that returns the length of the terminated coding pass.

## Description

The function `ippiEncodeGetTermPassLen_JPEG2K` is declared in the `ippj.h` file. This function returns the length `pPassLen` of the terminated coding pass of a given number `passNumber` that is stored in the `pState` state structure.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when any of the specified pointers is <code>NULL</code> .
<code>ippStsContextMatchErr</code>	Indicates an error condition if a pointer to an invalid state structure is passed.
<code>ippStsJPEG2KBadPassNumber</code>	Indicates an error condition if <code>passNumber</code> exceeds the allowed boundaries.

## EncodeGetRate\_JPEG2K

Returns estimated bit rate for the specified coding pass.

## Syntax

```
IppStatus ippiEncodeGetRate_JPEG2K(IppiEncodeState_JPEG2K* pState, int passNumber, int* pRate);
```

## Parameters

<code>pState</code>	Pointer to the allocated and initialized encoder state structure.
<code>passNumber</code>	Number of the specified coding pass.
<code>pRate</code>	Pointer to the variable that returns the estimated rate for the specified coding pass.

## Description

The function `ippiEncodeGetRate_JPEG2K` is declared in the `ippj.h` file. This function returns the estimated target bit rate `pRate` for the coding pass of a given number `passNumber`. The estimated bit rate is computed during the encoding procedure and then stored in the `pState` state structure. Therefore it is available only after completing both encoding procedures.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when any of the specified pointers is <code>NULL</code> .
<code>ippStsContextMatchErr</code>	Indicates an error condition if a pointer to an invalid state structure is passed.
<code>ippStsJPEG2KBadPassNumber</code>	Indicates an error condition if <code>passNumber</code> exceeds the allowed boundaries.

## EncodeGetDist\_JPEG2K

Returns estimated distortion of the image for the specified coding pass.

---

### Syntax

```
IppStatus ippiEncodeGetDist_JPEG2K(IppiEncodeState_JPEG2K* pState, int passNumber, Ipp64f* pDist);
```

### Parameters

<i>pState</i>	Pointer to the allocated and initialized encoder state structure.
<i>passNumber</i>	Number of the specified coding pass.
<i>pDist</i>	Pointer to the variable that returns the value of the estimated distortion.

### Description

The function `ippiEncodeGetDist_JPEG2K` is declared in the `ippj.h` file. This function returns the cumulative weighted mean square error (MSE) distortion *pDist* of the reconstructed image for the coding pass of a given number *passNumber*. The distortions are computed in accordance with [ISO15444], Annex J, section J.14, Rate Control during the encoding procedure and then stored in the *pState* structure. Therefore, it is available only after completing both encoding procedures.

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when any of the specified pointers is <code>NULL</code> .
<code>ippStsContextMatchErr</code>	Indicates an error condition if a pointer to an invalid state structure is passed.
<code>ippStsJPEG2KBadPassNumber</code>	Indicates an error condition if <i>passNumber</i> exceeds the allowed boundaries.

## DecodeGetBufSize\_JPEG2K

Computes the size of the work buffer for decoding routine.

---

### Syntax

```
IppStatus ippiDecodeGetBufSize_JPEG2K(IppiSize codeBlockMaxSize, int* pSize);
```

### Parameters

<i>codeBlockMaxSize</i>	Maximum size of code block in pixels.
<i>pSize</i>	Pointer to the variable that returns the size of the work buffer.

## Description

The function `ippiDecodeGetBufSize_JPEG2K` is declared in the `ippj.h` file. This function computes the size (in bytes) of the work buffer that corresponds to the specified maximum size of a code block (in pixels).

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when any of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if the width or height of the code block has zero or negative value.

## DecodeCodeBlock\_JPEG2K

*Decodes the compressed code block data.*

---

## Syntax

```
ippStatus ippiDecodeCodeBlock_JPEG2K_1u32s_C1R(const Ipp8u* pSrc, Ipp32s* pDst, int dstStep,
IppiSize codeBlockSize, IppiWTSubband subband, int sfBits, int nOfPasses, const int*
pTermPassLen, int nOfTermPasses, int codeStyleFlags, int* pErrorBitPlane, Ipp8u* pBuffer);
```

## Parameters

<code>pSrc</code>	Pointer to the source compressed code block.
<code>pDst</code>	Pointer to the destination for decoded data.
<code>dstStep</code>	Distance in bytes between starts of consecutive lines in the destination buffer.
<code>codeBlockSize</code>	Size of the code block.
<code>subband</code>	Wavelet transform subband contained the given code block. Possible values are listed in the <a href="#">Table "Subband Parameter for the JPEG2000 Entropy Coder Functions"</a> .
<code>sfBits</code>	Number of significant bit planes.
<code>nOfPasses</code>	Number of coding passes.
<code>pTermPassLen</code>	Pointer to the array that contains the length of each terminated coding pass.
<code>nOfTermPasses</code>	Number of terminated coding passes.
<code>codeStyleFlags</code>	Options for the decoding procedure. Possible values are listed in the <a href="#">Table "CodeStyleFlag Argument for the JPEG2000 Entropy Coder Functions"</a> .
<code>pErrorBitPlane</code>	Pointer to the bit plane that contains the first error returned when the damaged code block occurs.
<code>pBuffer</code>	Pointer to the work buffer.

## Description

The function `ippiDecodeCodeBlock_JPEG2K` is declared in the `ippj.h` file. This function decodes the compressed code block `pSrc` of data using adaptive arithmetic decoding procedure and stores the decoded data in the `pDst` buffer. Negative integers in the code block should be presented in the direct code with the sign in the most significant bit (31<sup>st</sup> in the zero-based numeration, when the least significant bit has a zero index). This code may be effectively obtained using the specially designed function `ippiComplement`.



**NOTE.** `errorBitPlane` should be set to `NULL` if the corresponding information is not required.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when any of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if the width or height of the code block has zero or negative value.
<code>ippStsJPEG2KDamagedCodeBlock</code>	Indicates an error condition if a code block contains damaged data.
<code>ippStsStepErr</code>	Indicates an error condition if <code>dstStep</code> has zero or negative value.

## DecodeCBProgrGetStateSize\_JPEG2K

*Computes the size of the external buffer for the decoder state structure.*

## Syntax

```
IppStatus ippiDecodeCBProgrGetStateSize_JPEG2K(IppiSize codeBlockMaxSize, int* pStateSize);
```

## Parameters

<code>codeBlockMaxSize</code>	Maximum size of the codeblock.
<code>pStateSize</code>	Pointer to the variable that returns the size of the buffer for the decoder state structure.

## Description

The function `ippiDecodeCBProgrGetStateSize_JPEG2K` is declared in the `ippj.h` file. This function computes the size of the external buffer for the decoder state structure. This parameter is depending on the maximum size of the codeblock `codeBlockMaxSize`.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error if <code>pStateSize</code> pointer is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <code>codeBlockMaxSize</code> has a field with zero or negative value.

## DecodeCBProgrInit\_JPEG2K

*Initializes the decoder state structure in the external buffer.*

---

### Syntax

```
IppStatus ippiDecodeCBProgrInit_JPEG2K(IppiDecodeCBProgrState_JPEG2K* pState);
```

### Parameters

*pState*                                      Pointer to the decoder state structure.

### Description

The function `ippiDecodeCBProgrInit_JPEG2K` is declared in the `ippj.h` file. This function initialize the decoder state structure for progressive mode *pState* in the external buffer which size should be computed previously by calling the function `ippiDecodeCBProgrGetStateSize_JPEG2K`.

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error if <i>pState</i> pointer is NULL.
<code>ippStsContextMatchErr</code>	Indicates an error condition if a pointer to the invalid structure is passed.

## DecodeCBProgrInitAlloc\_JPEG2K

*Allocates memory and initializes the decoder state structure in the external buffer.*

---

### Syntax

```
IppStatus ippiDecodeCBProgrInitAlloc_JPEG2K(IppiDecodeCBProgrState_JPEG2K** ppState, IppiSize codeBlockMaxSize);
```

### Parameters

<i>ppState</i>	Pointer to the pointer to the decoder state structure.
<i>codeBlockMaxSize</i>	Maximum size of the codeblock.

### Description

The function `ippiDecodeCBProgrInitAlloc_JPEG2K` is declared in the `ippj.h` file. This function allocates memory and initialize the decoder state structure for progressive mode *pState*.

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error if <i>pState</i> pointer is NULL.

`ippStsSizeErr` Indicates an error condition if `codeBlockSize` has a field with zero or negative value.

## DecodeCBProgrFree\_JPEG2K

Frees memory allocated for the decoder state structure.

### Syntax

```
IppStatus ippidecodeCBProgrFree_JPEG2K(IppiDecodeCBProgrState_JPEG2K* pState);
```

### Parameters

`pState` Pointer to the allocated and initialized decoder state structure.

### Description

The function `ippidecodeCBProgrFree_JPEG2K` is declared in the `ippj.h` file. This function frees memory allocated by the function `ippidecodeCBProgrInitAlloc_JPEG2K` for the decoder state structure.

### Return Values

`ippStsNoErr` Indicates no error.

`ippStsNullPtrErr` Indicates an error if `pState` pointer is `NULL`.

`ippStsContextMatchErr` Indicates an error condition if a pointer to the invalid structure is passed.

## DecodeCBProgrAttach\_JPEG2K

Attaches the buffer with data for progressive decoding.

### Syntax

```
IppStatus ippidecodeCBProgrAttach_JPEG2K_32s_C1R(Ipp32s* pDst, int dstStep, IppiSize codeBlockSize, IppiDecodeCBProgrState_JPEG2K* pState, IppiWTSubband subband, int sfBits, int codeStyleFlags);
```

### Parameters

`pDst` Pointer to the destination buffer.

`dstStep` Step in bytes through the destination buffer.

`codeBlockSize` Size of the codeblock.

`pState` Pointer to the decoder state structure.

`subband` Wavelet transform subband contained the given code block. Possible values are listed in the [Table "Subband Parameter for the JPEG2000 Entropy Coder Functions"](#).

`sfBits` Number of significant bits in code-block.

*codeStyleFlags*

Specifies the options for encoding procedure. The possible values are listed in the [Table “CodeStyleFlag Argument for the JPEG2000 Entropy Coder Functions”](#).

## Description

The function `ippiDecodeCBProgrAttach_JPEG2K` is declared in the `ippj.h` file. This function attaches a destination buffer *pDst* for the code-block rectangle and fix its parameters to perform the progressive decoding.



**CAUTION.** The image data buffer should not be moved in memory and should not be freed during whole decoding procedure, that is until the last step of decoding progression and damage correction will be completed.

---

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error if one of the specified pointers is <code>NULL</code> .
<code>ippStsStepErr</code>	Indicates an error condition if the value of <i>dstStep</i> is zero or negative.
<code>ippStsContextMatchErr</code>	Indicates an error condition if a pointer to the invalid structure is passed.

## DecodeCBProgrSetPassCounter\_JPEG2K

*Sets the value of internal coding pass counter.*

---

### Syntax

```
IppStatus ippiDecodeCBProgrSetPassCounter_JPEG2K(int nOfPasses,
IppiDecodeCBProgrState_JPEG2K* pState);
```

### Parameters

<i>nOfPasses</i>	Number of coding passes.
<i>pState</i>	Pointer to the decode state structure.

### Description

The function `ippiDecodeCBProgrSetPassCounter_JPEG2K` is declared in the `ippj.h` file. This function sets the value of internal coding pass counter.

The single coding pass is “sub-atom” for coding progression, but their counts in a single terminated segment (for example, the segment filled to the byte boundary) is available from the JPEG 2000 stream or file. During decoding step, pass counter is decremented by one or some other number depending on coding options. If pass counter is set to zero, decoding process will be blocked internally. Externally the pass counter value can be obtained by calling `ippiDecodeCBProgrGetPassCounter_JPEG2K` function.

The coding pass counter value can be set before any progression step as well as before series of steps.



## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error if <i>pState</i> pointer is NULL.
<code>ippStsContextMatchErr</code>	Indicates an error condition if a pointer to the invalid structure is passed.

## DecodeCBProgrGetPassCounter\_JPEG2K

Returns the value of the internal coding pass counter.

### Syntax

```
IppStatus ippidecodeCBProgrGetPassCounter_JPEG2K(IppiDecodeCBProgrState_JPEG2K* pState,  
int* pNOFResidualPasses);
```

### Parameters

<i>pState</i>	Pointer to the decode state structure.
<i>pNOFResidualPasses</i>	Pointer to the number of residual coding passes.

### Description

The function `ippidecodeCBProgrGetPassCounter_JPEG2K` is declared in the `ippj.h` file. This function returns the value of internal coding pass counter that is received from the decoding state structure *pState*.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error if <i>pState</i> pointer is NULL.
<code>ippStsContextMatchErr</code>	Indicates an error condition if a pointer to the invalid structure is passed.

## DecodeCBProgrGetCurBitPlane\_JPEG2K

Returns the number of the current bit plane.

### Syntax

```
IppStatus ippidecodeCBProgrGetCurBitPlaneNum_JPEG2K(IppiDecodeCBProgrState_JPEG2K* pState,  
int* pBitPlaneNum);
```

### Parameters

<i>pState</i>	Pointer to the decode state structure.
<i>pBitPlaneNum</i>	Pointer to the variable containing current bit plane number.

## Description

The function `ippiDecodeCBProgrGetCurBitPlane_JPEG2K` is declared in the `ippj.h` file. This function returns the current bit plane number. Initially the bit plane counter is set to  $(sfBits - 1)$  value (see the function `ippiDecodeCBProgrAttach_JPEG2K`). During successful decoding step, the bit plane counter is decremented by one or some other number depending on coding options.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error if <code>pState</code> pointer is <code>NULL</code> .
<code>ippStsContextMatchErr</code>	Indicates an error condition if a pointer to the invalid structure is passed.

## DecodeCBProgrStep\_JPEG2K

*Performs a single step of decoding.*

---

## Syntax

```
ippStatus ippiDecodeCBProgrStep_JPEG2K(const Ipp8u* pSrc, int srcLen,
ippiDecodeCBProgrState_JPEG2K* pState);
```

## Parameters

<code>pSrc</code>	Pointer to the source data.
<code>srcLen</code>	Length of the segment.
<code>pState</code>	Pointer to the decode state structure.

## Description

The function `ippiDecodeCBProgrStep_JPEG2K` is declared in the `ippj.h` file. This function performs one step of the decoding progression.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error if one of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error if length of terminated segment is specified incorrectly.
<code>ippStsContextMatchErr</code>	Indicates an error condition if a pointer to the invalid structure is passed.

## Component Transform Functions

This section describes the functions that perform component transformation specific for JPEG 2000 image coding system. All functions implement the component transformations in accordance with its definition by [ISO15444], Annex G, DC Level Shifting and Multiple Component Transformations.

The table below lists these functions described in more detail later in this section.

### Component Transform Functions

Function Base Name	Description
<a href="#">RCTFwd_JPEG2K</a>	Performs forward reversible component transformation.
<a href="#">RCTInv_JPEG2K</a>	Performs inverse reversible component transformation.
<a href="#">ICTFwd_JPEG2K</a>	Performs forward irreversible component transformation.
<a href="#">ICTInv_JPEG2K</a>	Performs inverse irreversible component transformation.

## RCTFwd\_JPEG2K

*Performs forward reversible component transformation.*

### Syntax

#### Case 1: Operation on pixel-order data

```
IppStatus ippiRCTFwd_JPEG2K_32s_C3P3R(const Ipp32s* pSrc, int srcStep, Ipp32s* pDst[3],
int dstStep, IppiSize roiSize);
```

#### Case 2: In-place operation on planar data

```
IppStatus ippiRCTFwd_JPEG2K_32s_P3IR(Ipp32s* pSrcDst[3], int srcDstStep, IppiSize roiSize);
```

### Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the array of pointers to ROIs in the planes of the destination image.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>pSrcDst</i>	Pointer to the array of pointers to the source and destination image ROIs for the in-place operation.
<i>dstSrcStep</i>	Distance in bytes between starts of consecutive lines in the source and destination image buffer for the in-place operation.
<i>roiSize</i>	Size of the source and destination ROI in pixels.

### Description

The function `ippiRCTFwd_JPEG2K` is declared in the `ippj.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function performs forward reversible component transformation (RCT) of the image buffer *pSrc*. The RCT is applied to all image component samples *I0*, *I1*, *I2*, corresponding to the first, second, and third components, and produces the transform samples *Y0*, *Y1*, *Y2* in accordance with the following formulas:

$$Y_0 = \left\lfloor \frac{I_0 + 2I_1 + I_2}{4} \right\rfloor$$

$$Y_1 = I_2 - I_1$$

$$Y_2 = I_0 - I_1$$

where the notation

$$\lfloor x \rfloor$$

designates the floor function, that is, the largest integer not exceeding  $x$ .

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when any of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <code>roiSize</code> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if any of the specified buffer step values is zero or negative.

### Example

The code example below shows how to use the function `ippiRCTFwd_JPEG2K_32s_C3P3R`.

```
void func_RCTFwd_JPEG2K()
{
    Ipp32s pSrc[(3*2)*4] = {1, 5, 2, 5, 1, 2,
                             3, 1, 2, 6, 2, 1,
```

```

        1, 3, 5, 3, 3, 1,
        2, 3, 1, 4, 2, 3};

int srcStep = (3*2)*sizeof(Ipp32s);
IppiSize roiSize = {2, 4};

Ipp32s* pDst[3];
int dstStep;

pDst[0] = ippiMalloc_32s_C1(roiSize.width, roiSize.height, &dstStep);
pDst[1] = ippiMalloc_32s_C1(roiSize.width, roiSize.height, &dstStep);
pDst[2] = ippiMalloc_32s_C1(roiSize.width, roiSize.height, &dstStep);

ippiRCTFwd_JPEG2K_32s_C3P3R(pSrc, srcStep, pDst, dstStep, roiSize);
}

```

Result:

pDst[0]	pDst[1]	pDst[2]
3 2	-3 1	-4 4
1 2	1 -1	2 4
3 2	2 -2	-2 0
2 2	-2 1	-1 2

## RCTInv\_JPEG2K

Performs inverse reversible component transformation.

### Syntax

#### Case 1: Operation on planar data

```

IppStatus ippiRCTInv_JPEG2K_32s_P3C3R(const Ipp32s* pSrc[3], int srcStep, Ipp32s* pDst,
int dstStep, IppiSize roiSize);

```

#### Case 2: In-place operation on planar data

```

IppStatus ippiRCTInv_JPEG2K_32s_P3IR(Ipp32s* pSrcDst[3], int srcDstStep, IppiSize roiSize);

```

## Parameters

<i>pSrc</i>	Pointer to the array of pointers to the ROIs in the planes of the source image.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>pSrcDst</i>	Pointer to the array of pointers to ROIs in the planes of the source and destination image for the in-place operation.
<i>dstSrcStep</i>	Distance in bytes between starts of consecutive lines in the source and destination image buffer for the in-place operation.
<i>roiSize</i>	Size of the source and destination ROI in pixels.

## Description

The function `ippiRCTFwd_JPEG2K` is declared in the `ippj.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function performs inverse RCT of the image buffer *pSrc* after inverse wavelet transformation. The inverse RCT is applied to transformed image component samples *Y0*, *Y1*, *Y2* and produces the corresponding samples *I0*, *I1*, *I2* in accordance with the following formulas:

$$I_1 = Y_0 - \left\lfloor \frac{Y_2 + Y_1}{4} \right\rfloor$$

$$I_0 = Y_2 + I_1$$

$$I_2 = Y_1 + I_1$$

where the notation

$$\lfloor x \rfloor$$

designates the floor function, that is, the largest integer not exceeding *x*.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when any of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.

`ippStsStepErr`

Indicates an error condition if any of the specified buffer step values is zero or negative.

## ICTFwd\_JPEG2K

*Performs forward irreversible component transformation.*

### Syntax

#### Case 1: Operation on pixel-order data

```
ippStatus ippiICTFwd_JPEG2K_32f_C3P3R(const Ipp32f* pSrc, int srcStep, Ipp32f* pDst[3],
int dstStep, IppiSize roiSize);
```

#### Case 2: In-place operation on planar data

```
ippStatus ippiICTFwd_JPEG2K_<mod>(Ipp<datatype>* pSrcDst[3], int srcDstStep, IppiSize
roiSize);
```

Supported values for `mod`:

`16s_P3IR`      `32s_P3IR`      `32f_P3IR`

### Parameters

<code>pSrc</code>	Pointer to the source image ROI.
<code>srcStep</code>	Distance in bytes between starts of consecutive lines in the source image.
<code>pDst</code>	Pointer to the array of pointers to ROIs in the planes of the destination image.
<code>dstStep</code>	Distance in bytes between starts of consecutive lines in the destination image.
<code>pSrcDst</code>	Pointer to the array of pointers to the source and destination image ROIs for the in-place operation.
<code>dstSrcStep</code>	Distance in bytes between starts of consecutive lines in the source and destination image buffer for the in-place operation.
<code>roiSize</code>	Size of the source and destination ROI in pixels.

### Description

The function `ippiICTFwd_JPEG2K` is declared in the `ippj.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function performs forward irreversible component transformation (ICT) of the image buffer `pSrc`. The ICT is applied to all image component samples  $I_0$ ,  $I_1$ ,  $I_2$ , corresponding to the first, second, and third components, and produces the transform samples  $Y_0$ ,  $Y_1$ ,  $Y_2$  in accordance with the following formulas:

$$Y_0 = 0.299 * I_0 + 0.587 * I_1 + 0.114 * I_2$$

$$Y_1 = -0.16875 * I_0 - 0.33126 * I_1 + 0.5 * I_2$$

$$Y_2 = 0.5 * I_0 - 0.41869 * I_1 - 0.08131 * I_2$$



**NOTE.** If the first three components are Red, Green and Blue components, then the forward ICT is of a YCbCr transformation.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when any of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <code>roiSize</code> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if any of the specified buffer step values is zero or negative.

## ICTInv\_JPEG2K

*Performs inverse irreversible component transformation.*

## Syntax

### Case 1: Operation on planar data

```
IppStatus ippICTInv_JPEG2K_32f_P3C3R(const Ipp32f* pSrc[3], int srcStep, Ipp32f* pDst,
int dstStep, IppiSize roiSize);
```

### Case 2: In-place operation on planar data

```
IppStatus ippICTInv_JPEG2K_<mod>(Ipp<datatype>* pSrcDst[3], int srcDstStep, IppiSize
roiSize);
```

Supported values for `mod`:

16s\_P3IR      32s\_P3IR      32f\_P3IR

## Parameters

<code>pSrc</code>	Pointer to the array of pointers to the ROIs in the planes of the source image.
<code>srcStep</code>	Distance in bytes between starts of consecutive lines in the source image.
<code>pDst</code>	Pointer to the destination image ROI.
<code>dstStep</code>	Distance in bytes between starts of consecutive lines in the destination image.
<code>pSrcDst</code>	Pointer to the array of pointers to ROIs in the planes of the source and destination image for the in-place operation.
<code>dstSrcStep</code>	Distance in bytes between starts of consecutive lines in the source and destination image buffer for the in-place operation.
<code>roiSize</code>	Size of the source and destination ROI in pixels.



## Description

The function `ippiICTFwd_JPEG2K` is declared in the `ippj.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function performs inverse irreversible component transformation (ICT) of the image buffer `pSrc` after inverse wavelet transformation. The inverse ICT is applied to transformed image component samples  $Y_0$ ,  $Y_1$ ,  $Y_2$  and produces the corresponding samples  $I_0$ ,  $I_1$ ,  $I_2$  in accordance with the following formulas:

$$I_0 = Y_0 + 1.402 * Y_2$$

$$I_1 = Y_0 - 0.34413 * Y_1 - 0.71414 * Y_2$$

$$I_2 = Y_0 + 1.772 * Y_1$$

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when any of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <code>roiSize</code> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if any of the specified buffer step values is zero or negative.

## Functions for RLE Coding

This section describes the functions that perform RLE compression of images in accordance with the DICOM "Digital imaging and communication in medicine" specification [[DICOM](#)].

The table below lists these functions described in more detail later in this section.

### RLE Compression Functions

Function Base Name	Description
<code>PackBitsRow_TIFF</code>	Encodes one row of the image.
<code>UnpackBitsRow_TIFF</code>	Decodes one RLE segment.
<code>SplitRow_TIFF</code>	Splits 16-bit monochrome Composite Pixel Code.
<code>JoinRow_TIFF</code>	Restores 16-bit monochrome Composite Pixel Code.

## RLE Coding Model

According to the DICOM specification, RLE compression consists of the following steps:

- the image is converted to a sequence of *Composite Pixel Codes*;
- the Composite Pixel Codes are used to generate a set of *Byte Segments*;
- each Byte Segment is RLE compressed to produce a *RLE Segment*;
- the *RLE Header* is appended in front of the concatenated RLE Segments.

## PackBitsRow\_TIFF

*Encodes one row of the image.*

---

### Syntax

```
IppStatus ippPackBitsRow_TIFF_8u_C1(const Ipp8u* pSrc, int srcLenBytes, Ipp8u* pDst, int*
pDstCurrPos, int dstLenBytes);
```

### Parameters

<i>pSrc</i>	Pointer to the input buffer to keep <i>Byte Segment</i> or part of <i>Byte Segment</i> .
<i>srcLenBytes</i>	Length of the input buffer, bytes.
<i>pDst</i>	Pointer to the output buffer.
<i>pDstCurrPos</i>	Pointer to the current byte position in the output buffer.
<i>dstLenBytes</i>	Length of the output buffer, bytes.

### Description

The function `ippPackBitsRow_TIFF` is declared in the `ippj.h` file. This function encodes *srcLenBytes* data from the input buffer *pSrc*. Encoded data are stored in the output buffer *pDst*. The function uses the encoding scheme called *PackBits* (see pseudo-code below). The input buffer keeps the part of the *Byte Segment* that represents one row of an image. The output buffer is used to keep RLE compressed input data. The size of output buffer must be sufficient to keep at least one row of output *RLE Segment*. Its size can be estimated in accordance with the following formula:

$dstLenBytes = srcLenBytes + (srcLenBytes + 127)/128$  for the worst case.

The algorithm can be expressed in the pseudo-code as follows:

a sequence of identical bytes (called as Replicate Run) is encoded as a two-byte code:

`<-count + 1><byte value>`, where `count` is number of bytes in the run, and  $2 \leq count \leq 128$

and a non repetitive sequence of bytes (called as Literal Run) is encoded as:

`<count - 1><literal sequence of bytes>`, where `count` is number of bytes in the sequence, and  $1 \leq count \leq 128$

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error if <i>pSrc</i> or <i>pDst</i> pointer is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>pDstCurrPos</i> has an illegal value, or if <i>srcLenBytes</i> has a negative value.
<code>ippStsUnderRunErr</code>	Indicates an error condition if size of the buffer <i>pDst</i> is insufficient to keep encoded row of the <i>RLE Segment</i> .

## UnpackBitsRow\_TIFF

*Decodes one RLE Segment.*

---

### Syntax

```
IppStatus ippiUnpackBitsRow_TIFF_8u_C1(const Ipp8u* pSrc, int* pSrcCurrPos, int srcLenBytes,
Ipp8u* pDst, int dstLenBytes);
```

### Parameters

<i>pSrc</i>	Pointer to input buffer with RLE segment
<i>srcLenBytes</i>	Size of the input RLE segment, bytes.
<i>pSrcCurrPos</i>	Pointer to the current byte position in the input buffer.
<i>pDst</i>	Pointer to the output buffer.
<i>dstLenBytes</i>	Size of single uncompressed row, bytes.

### Description

The function `ippiUnpackBitsRow_TIFF` is declared in the `ippj.h` file. This function decodes *srcLenBytes* data from the input buffer *pSrc* containing *RLE Segment*. Decoded *Byte segment* is stored in the output buffer *pDst*. The size of the input buffer must be sufficient to keep data for at least one row of *Byte Segment*. The function uses the decompression algorithm called *UnPackBits* (TIFF 6.0 specification) that can be expressed in pseudo-code as follows:

```
LOOP until the number of desired output bytes
Read the next source byte into N
IF N >= 0 and N <= 127 THEN
output the next N+1 bytes literally
ELSE IF N <= -1 and N >= -127 THEN
output the next byte -N+1 times
ELSE IF N = -128
output nothing
ENDIF
ENDLOOP
```

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error if <i>pSrc</i> or <i>pDst</i> pointer is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>pSrcCurrPos</i> has an illegal value, or if <i>dstLenBytes</i> has a negative value.
<code>ippStsUnderRunErr</code>	Indicates an error condition if size of the buffer <i>pSrc</i> is insufficient to decompress row of the Byte segment.

## SplitRow\_TIFF

*Splits 16-bit monochrome Composite Pixel Code.*

---

### Syntax

```
IppStatus ippiSplitRow_TIFF_16u8u_C1(const Ipp16u* pSrc, Ipp8u pDst[2], int srcLen);
```

### Parameters

<i>pSrc</i>	Pointer to the input buffer containing 16-bit monochrome Composite Pixel Code.
<i>srcLen</i>	Number of elements in the source buffer.
<i>pDst</i>	Pointer to the output buffers.

### Description

The function `ippiSplitRow_TIFF` is declared in the `ippj.h` file. This function takes the most significant and the least significant bytes of each of *srcLen* elements of the input buffer *pSrc*, and stores them sequentially to the output buffers pointed by *pDst*. The *pDst*[0] contains only the most significant bytes, and the *pDst*[1] contains only the least significant bytes.

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error if <i>pSrc</i> or <i>pDst</i> pointer is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>srcLen</i> has a negative or illegal value.

## JoinRow\_TIFF

*Restores 16-bit monochrome Composite Pixel Code.*

---

### Syntax

```
IppStatus ippiJoinRow_TIFF_8u16u_C1(const Ipp8u pSrc[2], Ipp16u* pDst, int dstLen);
```

### Parameters

<i>pSrc</i>	Pointers to the input buffer containing decoded RLE segments.
<i>pDst</i>	Pointer to the output buffer.
<i>dstLen</i>	Number of elements in the destination buffer.

### Description

The function `ippiJoinRow_TIFF` is declared in the `ippj.h` file.

This function restores 16-bit elements in the output buffer *pDst* by joining for each element the most significant and the least significant bytes stored in the source buffers *pSrc*[0] and *pSrc*[1] respectively. The number of elements is specified by the parameter *dstLen*.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error if <code>pSrc</code> or <code>pDst</code> pointer is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <code>dstLen</code> has a negative or illegal value.

## Texture Compression Functions

This section describes the functions that perform compression of the textures. Textures are images that are applied to the surface of shapes to add visual detail or color. In the modern computer graphics textures are used extremely wide and can consume large amounts of system and video memory. The compression of textures reduces the size of the required memory.

Intel IPP implements the Microsoft DirectX\* texture compression algorithms (DXTC or DXTn) also known as S3 Texture Compression [S3TC]. It is a lossy compression with a fixed compression ratio of 4:1 or 8:1. This compression is a form of *block truncating coding* (BTC), where an image is divided into non-overlapping blocks and pixels in each block are quantized to a limited number of values.

The color values of pixels in a 4x4 pixel block are approximated with equidistant points on a line through color space. This line is defined by two end points and for each pixel in the 4x4 block an index is stored to one of the equidistant points on the line. The end points of the line through color space are quantized to 16-bit 5:6:5 RGB format and either one or two intermediate points are generated through interpolation. The DXT1 format allows an 1-bit alpha channel to be encoded by using only one intermediate point and specifying one color which is black and fully transparent. The DXT2 and DXT3 formats encode a separate explicit 4-bit alpha value for each pixel in a 4x4 block. The DXT4 and DXT5 formats store a separate alpha channel which is compressed similarly to the color channels where the alpha values in a 4x4 block are approximated with equidistant points on a line through alpha space. In the DXT2 and DXT4 formats it is assumed that the color values have been premultiplied by the alpha values. In the DXT3 and DXT5 formats it is assumed that the color values are not premultiplied by the alpha values (see also more details at [http://msdn.microsoft.com/en-us/library/bb204843\(VS.85\).aspx](http://msdn.microsoft.com/en-us/library/bb204843(VS.85).aspx)).

Intel IPP supports three variations of the S3TC algorithm: DXT1, DXT3, and DXT5.

The table below lists functions for texture compression described in more detail later in this section.

### Texture Compression Functions

Function Base Name	Description
<code>TextureEncodeBlockFromRGBA</code>	Encodes RGBA texture images.
<code>TextureEncodeBlockFromRG</code>	Encodes two-channel texture images.
<code>TextureEncodeBlock</code>	Encodes one-channel texture images.
<code>TextureEncodeBlock4x4FromRGBA</code>	
<code>TextureDecodeBlockToRGBA</code>	Decodes compressed RGBA texture images.
<code>TextureDecodeBlockToRGB</code>	Decodes BC6 blocks to the texture image.
<code>TextureDecodeBlockToRG</code>	Decodes BC5 blocks to the texture image.
<code>TextureDecodeBlock</code>	Decodes BC4 blocks to the texture image.
<code>TextureEncodeBlockFromYCoCg</code>	Encodes YCoCg texture images.

## TextureEncodeBlockFromRGBA

*Encodes RGBA texture images.*

---

### Syntax

```
IppStatus ippiTextureEncodeBlockFromRGBA_DXT1_8u_C4C1R(const Ipp8u* pSrc, Ipp32u srcStep,
IppiSize roiSize, Ipp8u* pDst);
```

```
IppStatus ippiTextureEncodeBlockFromRGBA_DXT3_8u_C4C1R(const Ipp8u* pSrc, Ipp32u srcStep,
IppiSize roiSize, Ipp8u* pDst);
```

```
IppStatus ippiTextureEncodeBlockFromRGBA_DXT5_8u_C4C1R(const Ipp8u* pSrc, Ipp32u srcStep,
IppiSize roiSize, Ipp8u* pDst);
```

### Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>roiSize</i>	Size of the source image ROI in pixels.
<i>pDst</i>	Pointer to the destination array of the encoded blocks.

### Description

The function `ippiTextureEncodeBlockFromRGBA` is declared in the `ippj.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function encodes the source image ROI *pSrc* from RGBA to DXT1, DXT3 or DXT5 format in accordance with the descriptor DXT1, DXT3 and DXT5 respectively. You must ensure that the *pDst* buffer contains enough space for compressed data. For the DXT1 compression (compression ratio 8:1) the size of one compressed block is 8 bytes, for the DXT3 and DXT5 compressions (compression ratio 4:1) the size of one compressed block is 16 bytes.



---

**NOTE.** Negative step values are not supported.

---

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error if <i>pSrc</i> or <i>pDst</i> pointer is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.

## TextureEncodeBlockFromRG

*Encodes a texture image to the BC5 format.*

---

### Syntax

```
IppStatus ippTextureEncodeBlockFromRG_BC5_8u_C2C1R(const Ipp8u* pSrc, Ipp32u srcStep,
IppiSize roiSize, Ipp8u* pDst);
```

### Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>roiSize</i>	Size of the source and destination ROI in pixels.
<i>pDst</i>	Pointer to the destination image.

### Description

The function `ippTextureEncodeBlockFromRG` is declared in the `ippj.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function encodes a two-channel texture image ROI *pSrc* to BC5 compression format and stores result in *pDst*.

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error if <i>pSrc</i> or <i>pDst</i> pointer is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> has a zero or negative value.

## TextureEncodeBlock

*Encodes a texture image to the BC4 format.*

---

### Syntax

```
IppStatus ippTextureEncodeBlock_BC4_8u_C1R(const Ipp8u* pSrc, Ipp32u srcStep, IppiSize
roiSize, Ipp8u* pDst);
```

### Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>roiSize</i>	Size of the source and destination ROI in pixels.

*pDst* Pointer to the destination image.

### Description

The function `ippiTextureEncodeBlock` is declared in the `ippj.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function encodes a one-channel texture image ROI *pSrc* to BC4 compression format and stores result in *pDst*.

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error if <i>pSrc</i> or <i>pDst</i> pointer is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> has a zero or negative value.

## TextureEncodeBlock4x4FromRGBA

*Encodes RGBA texture images.*

---

### Syntax

```
ippStatus ippiTextureEncodeBlock4x4FromRGBA_BC7_8u_C4C1R(const Ipp8u* pSrc, Ipp32u srcStep,
Ipp8u* pDst, Ipp32u* pMSE, Ipp32u mode);
```

### Parameters

<i>pSrc</i>	Pointer to the source data.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image.
<i>pMSE</i>	Pointer to the calculated mean square error.
<i>mode</i>	Block coding mode (0,.., 7)

### Description

The function `ippiTextureEncodeBlock4x4FromRGBA` is declared in the `ippj.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function encodes the source image ROI *pSrc* from RGBA to BC7 format. You must ensure that the *pDst* buffer contains enough space for compressed data. For BC7 (compression ratio 4:1) the size of one compressed block is 16 bytes.




---

**NOTE.** Negative step values are not supported.

---



## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error if <i>pSrc</i> or <i>pDst</i> pointer is NULL.

## TextureDecodeBlockToRGBA

*Decodes compressed RGBA texture images.*

### Syntax

```
IppStatus ippiTextureDecodeBlockToRGBA_DXT1_8u_C1C4R(const Ipp8u* pSrc, Ipp8u* pDst, Ipp32u dstStep, IppiSize roiSize);
```

```
IppStatus ippiTextureDecodeBlockToRGBA_DXT3_8u_C1C4R(const Ipp8u* pSrc, Ipp8u* pDst, Ipp32u dstStep, IppiSize roiSize);
```

```
IppStatus ippiTextureDecodeBlockToRGBA_DXT5_8u_C1C4R(const Ipp8u* pSrc, Ipp8u* pDst, Ipp32u dstStep, IppiSize roiSize);
```

```
IppStatus ippiTextureDecodeBlockToRGBA_BC7_8u_C1C4R(const Ipp8u* pSrc, Ipp8u* pDst, Ipp32u dstStep, IppiSize roiSize);
```

### Parameters

<i>pSrc</i>	Pointer to the source array of the encoded blocks.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>roiSize</i>	Size of the destination image ROI in pixels.

### Description

The function `ippiTextureDecodeBlockToRGBA` is declared in the `ippj.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function decodes the source data *pSrc* and restores the RGBA texture image *pDst*. Source data are the encoded blocks in DXT1, DXT3, DXT5, or BC7 formats in accordance with the descriptor DXT1, DXT3, DXT5 and BC7 respectively. You must ensure that the *pSrc* buffer contains enough compressed data. For the DXT1 compression (compression ratio 8:1) the size of one compressed block is 8 bytes, for the DXT3 and DXT5 compressions (compression ratio 4:1) the size of one compressed block is 16 bytes, for BC7 (compression ratio 4:1) the size of one compressed block is 16 bytes.



**NOTE.** Negative step values are not supported.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error if <i>pSrc</i> or <i>pDst</i> pointer is NULL.

`ippStsSizeErr`Indicates an error condition if *roiSize* has a field with zero or negative value.

## TextureDecodeBlockToRGB

*Decodes compressed RGB texture images.*

---

### Syntax

```
IppStatus ippiTextureDecodeBlockToRGB_BC6_16u_C1C3R(const Ipp8u* pSrc, Ipp16u* pDst, Ipp32u
dstStep, IppiSize roiSize);
```

```
IppStatus ippiTextureDecodeBlockToRGB_BC6_16s_C1C3R(const Ipp8u* pSrc, Ipp16s* pDst, Ipp32u
dstStep, IppiSize roiSize);
```

### Parameters

<i>pSrc</i>	Pointer to the source array of the encoded blocks.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>roiSize</i>	Size of the destination image ROI in pixels.

### Description

The function `ippiTextureDecodeBlockToRGB` is declared in the `ippj.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function decodes the source data *pSrc* and restores the RGB texture image *pDst*. Source data are the encoded blocks in BC6 format. You must ensure that the *pSrc* buffer contains enough compressed data. For the BC6 format (compression ratio 6:1) the size of one compressed block is 16 bytes.



---

**NOTE.** Negative step values are not supported.

---

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error if <i>pSrc</i> or <i>pDst</i> pointer is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.

## TextureDecodeBlockToRG

*Decodes texture BC5 blocks to two-channel image.*

---

### Syntax

```
IppStatus ippiTextureDecodeBlockToRG_BC5_8u_C1C2R(const Ipp8u* pSrc, Ipp8u* pDst, Ipp32u
dstStep, IppiSize roiSize);
```

## Parameters

<i>pSrc</i>	Pointer to the BC5 encoded source data.
<i>pDst</i>	Pointer to the destination image.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>roiSize</i>	Size of the destination ROI in pixels.

## Description

The function `ippiTextureDecodeBlockToRG` is declared in the `ippj.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function decodes BC5 texture blocks *pSrc* to the two-channel image and stores result in *pDst*.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error if <i>pSrc</i> or <i>pDst</i> pointer is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> has a zero or negative value.

## TextureDecodeBlock

Decodes texture BC4 blocks to the one-channel image.

## Syntax

```
ippStatus ippiTextureDecodeBlock_BC4_8u_C1R(const Ipp8u* pSrc, Ipp8u* pDst, Ipp32u dstStep,
IppiSize roiSize);
```

## Parameters

<i>pSrc</i>	Pointer to the BC4 encoded source data.
<i>pDst</i>	Pointer to the destination image.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>roiSize</i>	Size of the destination ROI in pixels.

## Description

The function `ippiTextureDecodeBlock` is declared in the `ippj.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function decodes BC4 texture blocks *pSrc* to the one-channel image and stores result in *pDst*.

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error if <code>pSrc</code> or <code>pDst</code> pointer is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <code>roiSize</code> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <code>srcStep</code> has a zero or negative value.

## TextureEncodeBlockFromYCoCg

*Encodes YCoCg texture images.*

---

### Syntax

```
IppStatus ippiTextureEncodeBlockFromYCoCg_DXT5_8u_C3C1R(const Ipp8u* pSrc, Ipp32u srcStep,
IppiSize roiSize, Ipp8u* pDst);
```

### Parameters

<code>pSrc</code>	Pointer to the source image ROI.
<code>srcStep</code>	Distance in bytes between starts of consecutive lines in the source image.
<code>roiSize</code>	Size of the source image ROI in pixels.
<code>pDst</code>	Pointer to the destination array of the encoded blocks.

### Description

The function `ippiTextureEncodeBlockFromYCoCg_DXT5` is declared in the `ippj.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function encodes the source image ROI `pSrc` from YCoCg to DXT5 format. The user must ensure that the `pDst` buffer contains enough space for compressed data. For the DXT5 compression (compression ratio 4:1) the size of one compressed block is 16 bytes.



---

**NOTE.** Negative step values are not supported.

---

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error if <code>pSrc</code> or <code>pDst</code> pointer is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <code>roiSize</code> has a field with zero or negative value.

## High Definition Photo Coding

This section describes Intel IPP functions for high definition photo coding.

### Photo Core Transform Functions

**THESE FUNCTIONS ARE DEPRECATED. PLEASE USE CORRESPONDING JPEG XR CODING FUNCTIONS**

The photo core transform (PCT) is a lossless nonlinear 4x4 transform operation inspired by a 4x4 DCT. The PCT is defined on integers. In this space it is involuntary lossless operation, that is original image can be restored with no losses by the inverse operation from its transformed copy. The entire 4x4 PCT transform process consists of cascade of 2x2 Hadamard 2D transforms combined with 1D and 2D rotate operations.

Table below lists functions for PCT described in more detail later in this section.

#### Photo Core Transform Functions

Function Base Name	Description
<a href="#">PCTFwd_HDP</a> , <a href="#">PCTFwd8x8_HDP</a> , <a href="#">PCTFwd8x16_HDP</a> , <a href="#">PCTFwd16x16_HDP</a>	<b>THESE FUNCTIONS ARE DEPRECATED.</b> Perform forward photo core transform.
<a href="#">PCTInv_HDP</a> , <a href="#">PCTInv8x8_HDP</a> , <a href="#">PCTInv8x16_HDP</a> , <a href="#">PCTInv16x16_HDP</a> ,	<b>THESE FUNCTIONS ARE DEPRECATED.</b> Perform inverse photo core transform.

### PCTFwd\_HDP

**THIS FUNCTION IS DEPRECATED.** *Performs forward photo core transform.*

#### Syntax

```
IppStatus ippiPCTFwd_HDP_32s_C1IR(Ipp32s* pSrcDst, Ipp32u srcDstStep, IppiSize roiSize);
IppStatus ippiPCTFwd16x16_HDP_32s_C1IR(Ipp32s* pSrcDst, Ipp32u srcDstStep);
IppStatus ippiPCTFwd8x16_HDP_32s_C1IR(Ipp32s* pSrcDst, Ipp32u srcDstStep);
IppStatus ippiPCTFwd8x8_HDP_32s_C1IR(Ipp32s* pSrcDst, Ipp32u srcDstStep);
```

#### Parameters

<i>pSrcDst</i>	Pointer to the source and destination image ROI.
<i>srcDstStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>roiSize</i>	Size of the source and destination image ROI in pixels.

#### Description



**CAUTION.** THESE FUNCTIONS ARE DEPRECATED. Please use the functions [ippiPCTFwd\\_JPEGXR](#) instead.

The functions `ippiPCTFwd_HDP` are declared in the `ippj.h` file. They operate with ROI (see [Regions of Interest in Intel IPP](#)).

The function `ippiPCTFwd_HDP` performs the forward photo core transform of integer data over all 4x4 blocks of pixels within the given ROI in the source image.

The function flavors `ippiPCTFwd16x16_HDP`, `ippiPCTFwd8x16_HDP` and `ippiPCTFwd8x8_HDP` operate on the ROI with the predefined size; their result is equivalent to the results of the function `ippiPCTFwd_HDP` with `roiSize` equal to `{16,16}`, `{8,16}` and `{8,8}` respectively.

The value of `srcDstStep` must be at least `sizeof(Ipp32s) * roiSize.width`, that is `4*roiSize.width`.

For function flavors with the predefined ROI size `srcDstStep` must be at least 64 for `ippiPCTFwd16x16_HDP`, and 32 for `ippiPCTFwd8x16_HDP` and `ippiPCTFwd8x8_HDP`.

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error if <code>pSrcDst</code> pointer is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <code>roiSize</code> has a field with zero or negative value, or if it is not a multiple of 4.
<code>ippStsStepErr</code>	Indicates an error if <code>srcDstStep</code> is less than <code>4*roiSize.width</code> .

## PCTInv

**THIS FUNCTION IS DEPRECATED.** Performs inverse photo core transform.

---

### Syntax

```
IppStatus ippiPCTInv_HDP_32s_C1IR(Ipp32s* pSrcDst, Ipp32u srcDstStep, IppiSize roiSize);
IppStatus ippiPCTInv16x16_HDP_32s_C1IR(Ipp32s* pSrcDst, Ipp32u srcDstStep);
IppStatus ippiPCTInv8x16_HDP_32s_C1IR(Ipp32s* pSrcDst, Ipp32u srcDstStep);
IppStatus ippiPCTInv8x8_HDP_32s_C1IR(Ipp32s* pSrcDst, Ipp32u srcDstStep);
```

### Parameters

<code>pSrcDst</code>	Pointer to the source and destination image ROI.
<code>srcDstStep</code>	Distance in bytes between starts of consecutive lines in the source image.
<code>roiSize</code>	Size of the source and destination image ROI in pixels.

### Description



**CAUTION.** THESE FUNCTIONS ARE DEPRECATED. Please use the functions `ippiPCTInv_JPEGXR` instead.

---

The functions `ippiPCTInv_HDP` are declared in the `ippj.h` file. They operate with ROI (see [Regions of Interest in Intel IPP](#)).

The function `ippiPCTInv_HDP` performs the inverse photo core transform of integer data over all 4x4 blocks of pixels within the given ROI in the source image.

The function flavors `ippiPCTInv16x16_HDP`, `ippiPCTInv8x16_HDP` and `ippiPCTInv8x8_HDP` operate on the ROI with the predefined size; their result is equivalent to the results of the function `ippiPCTInv` with `roiSize` equal to `{16,16}`, `{8,16}` and `{8,8}` respectively.

The value of `srcDstStep` must be at least `sizeof(Ipp32s) * roiSize.width`, that is `4*roiSize.width`.

For function flavors with the predefined ROI size `srcDstStep` must be at least 64 for `ippiPCTInv16x16_HDP`, and 32 for `ippiPCTInv8x16_HDP` and `ippiPCTInv8x8_HDP`.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error if <code>pSrcDst</code> pointer is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <code>roiSize</code> has a field with zero or negative value, or if it is not a multiple of 4.
<code>ippStsStepErr</code>	Indicates an error if <code>srcDstStep</code> is less than <code>4*roiSize.width</code> .

## JPEG XR Coding

This section describes Intel IPP functions for the JPEG XR (eXtended Range) image coding system. Table below lists these functions described in more detail later in this section.

### Functions for JPEG XR Coding

Function Base Name	Description
<code>PCTFwd_JPEGXR</code> , <code>PCTFwd8x8_JPEGXR</code> , <code>PCTFwd8x16_JPEGXR</code> , <code>PCTFwd16x16_JPEGXR</code>	Performs forward core transform.
<code>PCTInv_JPEGXR</code> , <code>PCTInv8x8_JPEGXR</code> , <code>PCTInv8x16_JPEGXR</code> , <code>PCTInv16x16_JPEGXR</code>	Performs inverse core transform.
<code>VLCGetStateSize_JPEGXR</code>	Returns the size of the VLC state structure.
<code>VLCInit_JPEGXR</code>	Initializes the VLC state structure.
<code>VLCAdapt_JPEGXR</code>	Adapts the VLC model.
<code>VLCScanSet_JPEGXR</code> , <code>VLCScanReset_JPEGXR</code>	Sets or resets the VLC scan type.
<code>VLCScan4x4DC_JPEGXR</code>	Performs adaptive scanning of the DC coefficients.
<code>VLCEncode4x4Flex_JPEGXR</code> , <code>VLCEncode4x4_JPEGXR</code> , <code>VLCEncodeDC444_JPEGXR</code> , <code>VLCEncodeDC422_JPEGXR</code> , <code>VLCEncodeDC420_JPEGXR</code>	Performs VLC encoding.
<code>VLCDecode4x4_JPEGXR</code> , <code>VLCDecodeDC444_JPEGXR</code> , <code>VLCDecodeDC422_JPEGXR</code> , <code>VLCDecodeDC420_JPEGXR</code>	Performs VLC decoding.
<code>FLCEncode4x4_JPEGXR</code>	Performs FLC encoding.
<code>FLCDecode4x4_JPEGXR</code>	Performs FLC decoding.

## Core Transform Functions

The core transform is a nonlinear 4x4 transform operation inspired by a 4x4 DCT. In integer space it is involutory lossless operation, that is original image can be restored with no losses by the inverse operation from its transformed copy. The entire 4x4 core transform process consists of cascade of 2x2 Hadamard 2D transforms combined with 1D and 2D rotate operations.

## PCTFwd\_JPEGXR

*Performs forward JPEG-XR core transform.*

---

### Syntax

```

IppStatus ippiPCTFwd_JPEGXR_16s_C1IR(Ipp16s* pSrcDst, Ipp32u srcDstStep, IppiSize roiSize);
IppStatus ippiPCTFwd16x16_JPEGXR_16s_C1IR(Ipp16s* pSrcDst, Ipp32u srcDstStep);
IppStatus ippiPCTFwd8x16_JPEGXR_16s_C1IR(Ipp16s* pSrcDst, Ipp32u srcDstStep);
IppStatus ippiPCTFwd8x8_JPEGXR_16s_C1IR(Ipp16s* pSrcDst, Ipp32u srcDstStep);
IppStatus ippiPCTFwd_JPEGXR_32s_C1IR(Ipp32s* pSrcDst, Ipp32u srcDstStep, IppiSize roiSize);
IppStatus ippiPCTFwd16x16_JPEGXR_32s_C1IR(Ipp32s* pSrcDst, Ipp32u srcDstStep);
IppStatus ippiPCTFwd8x16_JPEGXR_32s_C1IR(Ipp32s* pSrcDst, Ipp32u srcDstStep);
IppStatus ippiPCTFwd8x8_JPEGXR_32s_C1IR(Ipp32s* pSrcDst, Ipp32u srcDstStep);
IppStatus ippiPCTFwd_JPEGXR_32f_C1IR(Ipp32f* pSrcDst, Ipp32u srcDstStep, IppiSize roiSize);
IppStatus ippiPCTFwd16x16_JPEGXR_32f_C1IR(Ipp32f* pSrcDst, Ipp32u srcDstStep);
IppStatus ippiPCTFwd8x16_JPEGXR_32f_C1IR(Ipp32f* pSrcDst, Ipp32u srcDstStep);
IppStatus ippiPCTFwd8x8_JPEGXR_32f_C1IR(Ipp32f* pSrcDst, Ipp32u srcDstStep);

```

### Parameters

<i>pSrcDst</i>	Pointer to the source and destination image ROI.
<i>srcDstStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>roiSize</i>	Size of the source and destination image ROI in pixels.

### Description

The functions `ippiPCTFwd` are declared in the `ippj.h` file. They operate with ROI (see [Regions of Interest in Intel IPP](#)).

The function `ippiPCTFwd` performs the forward core transform of integer data over all 4x4 blocks of pixels within the given ROI in the source image.

The function flavors `ippiPCTFwd16x16_JPEGXR`, `ippiPCTFwd8x16_JPEGXR` and `ippiPCTFwd8x8_JPEGXR` operate on the 16x16, 8x16, and 8x8 macroblocks respectively. Their result is equivalent to the results of the function `ippiPCTFwd_JPEGXR` with *roiSize* equal to `{16,16}`, `{8,16}` and `{8,8}` respectively.



For the function flavors with the arbitrary ROI the value of *srcDstStep* must be at least *sizeof(dataType) \* roiSize.width*.

JPEG-XR forward core transform consists of cascade of three elementary 2x2 transform operations: Hadamard transform, 1D rotation, and 2D rotation.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error if <i>pSrcDst</i> pointer is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value, or if it is not a multiple of 4.
<code>ippStsStepErr</code>	Indicates an error if <i>srcDstStep</i> is less than <i>sizeof(dataType) * roiSize.width</i> for the function flavors with the arbitrary ROI.

## PCTInv\_JPEGXR

*Performs inverse JPEG-XR core transform.*

### Syntax

```

IppStatus ippipCTInv_JPEGXR_16s_C1IR(Ipp16s* pSrcDst, Ipp32u srcDstStep, IppiSize roiSize);
IppStatus ippipCTInv16x16_JPEGXR_16s_C1IR(Ipp16s* pSrcDst, Ipp32u srcDstStep);
IppStatus ippipCTInv8x16_JPEGXR_16s_C1IR(Ipp16s* pSrcDst, Ipp32u srcDstStep);
IppStatus ippipCTInv8x8_JPEGXR_16s_C1IR(Ipp16s* pSrcDst, Ipp32u srcDstStep);
IppStatus ippipCTInv_JPEGXR_32s_C1IR(Ipp32s* pSrcDst, Ipp32u srcDstStep, IppiSize roiSize);
IppStatus ippipCTInv16x16_JPEGXR_32s_C1IR(Ipp32s* pSrcDst, Ipp32u srcDstStep);
IppStatus ippipCTInv8x16_JPEGXR_32s_C1IR(Ipp32s* pSrcDst, Ipp32u srcDstStep);
IppStatus ippipCTInv8x8_JPEGXR_32s_C1IR(Ipp32s* pSrcDst, Ipp32u srcDstStep);
IppStatus ippipCTInv_JPEGXR_32f_C1IR(Ipp32f* pSrcDst, Ipp32u srcDstStep, IppiSize roiSize);
IppStatus ippipCTInv16x16_JPEGXR_32f_C1IR(Ipp32f* pSrcDst, Ipp32u srcDstStep);
IppStatus ippipCTInv8x16_JPEGXR_32f_C1IR(Ipp32f* pSrcDst, Ipp32u srcDstStep);
IppStatus ippipCTInv8x8_JPEGXR_32f_C1IR(Ipp32f* pSrcDst, Ipp32u srcDstStep);

```

### Parameters

<i>pSrcDst</i>	Pointer to the source and destination image ROI.
<i>srcDstStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>roiSize</i>	Size of the source and destination image ROI in pixels.

### Description

The functions `ippipCTInv` are declared in the `ippj.h` file. They operate with ROI (see [Regions of Interest in Intel IPP](#)).

The function `ippiPCTInv` performs the inverse core transform of integer data over all 4x4 blocks of pixels within the given ROI in the source image.

The function flavors `ippiPCTInv16x16`, `ippiPCTInv8x16` and `ippiPCTInv8x8` operate on the 16x16, 8x16, and 8x8 macroblocks respectively. Their result is equivalent to the results of the function `ippiPCTInv` with `roiSize` equal to `{16,16}`, `{8,16}` and `{8,8}` respectively.

For the function flavors with the arbitrary ROI the value of `srcDstStep` must be at least `sizeof(dataType) * roiSize.width`.

JPEG-XR inverse core transform consists of cascade of three elementary 2x2 transform operations: 2D inverse rotation, 1D inverse rotation, and Hadamard transform.

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error if <code>pSrcDst</code> pointer is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <code>roiSize</code> has a field with zero or negative value, or if it is not a multiple of 4.
<code>ippStsStepErr</code>	Indicates an error if <code>srcDstStep</code> is less than <code>sizeof(dataType) * roiSize.width</code> for the function flavors with the arbitrary ROI.

### Adaptive Variable and Fixed Length Coding Functions

Transform coefficients coding proceeds in the following stages:

- Adaptive scanning of each 4x4 block of transform coefficients to identify the number of non-zero coefficients as a measure of the statistical variance of the block.
- Transform coefficients are dynamically partitioned into more significant bits and less significant bits (flexbits). The more significant bits are coded using entropy, or *variable length coding* (VLC), and the flexbits are coded using adoptive *fixed length coding* (FLC).

### VLCGetStateSize\_JPEGXR

*Calculates the size of the VLC state structure.*

---

#### Syntax

```
IppStatus ippiVLCGetStateSize_JPEGXR(int* pSize);
```

#### Parameters

`pSize`                      Pointer to the size of the memory for the VLC state structure.

#### Description

The function `ippiVLCGetStateSize_JPEGXR` is declared in the `ippj.h` file. This function returns the size of the VLC state structure.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pSize</i> pointer is <code>NULL</code> .

## VLCInit\_JPEGXR

*Initializes the VLC state structure.*

---

### Syntax

```
IppStatus ippivLCInit_JPEGXR(IppivLCState_JPEGXR* pVLCState);
```

### Parameters

*pVLCState*                      Pointer to the memory block for the VLC state structure.

### Description

The function `ippivLCInit_JPEGXR` is declared in the `ippj.h` file. This function initializes the VLC state structure.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pVLCState</i> pointer is <code>NULL</code> .

## VLCAdapt\_JPEGXR

*Adapts the VLC model.*

---

### Syntax

```
IppStatus ippivLCAdapt_JPEGXR(IppivLCState_JPEGXR* pVLCState, IppivLCAdaptType adaptType);
```

### Parameters

*pVLCState*                      Pointer to the VLC state structure.  
*adaptType*                      Adaptive VLC model type, possible values: `ippVLCAdaptLowpass` or `ippVLCAdaptHighpass`.

### Description

The function `ippivLCAdapt_JPEGXR` is declared in the `ippj.h` file.

This function performs adaptation of the VLC model of given type *adaptType*, that is it selects the VLC table with maximal relative advantage for entropy coding using statistic accumulated to the moment for each table.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pVLCState</i> pointer is <code>NULL</code> .

`ippStsRangeErr` Indicates an error condition if *adaptType* is not equal to `ippVLCAdaptLowpass` or `ippVLCAdaptHighpass`.

## VLCScanSet\_JPEGXR, VLCScanReset

*Sets or resets the VLC scan type.*

---

### Syntax

```
IppStatus ippivLCScanSet_JPEGXR(IppiVLCState_JPEGXR* pVLCState, IppiVLCScanType scanType);
IppStatus ippivLCScanReset_JPEGXR(IppiVLCState_JPEGXR* pVLCState, IppiVLCScanType scanType);
```

### Parameters

*pVLCState* Pointer to the VLC state structure.

*scanType* VLC scan type, possible values: `ippVLCScanVert`, `ippVLCScanHoriz`, or `ippVLCScanRaster`.

### Description

The functions `ippivLCScanSet_JPEGXR` and `ippivLCScanReset_JPEGXR` are declared in the `ippj.h` file.

The function `ippivLCScanSet_JPEGXR` sets type of the current scan table. The function `ippivLCScanReset_JPEGXR` initializes the scan table of the given scan type.

### Return Values

`ippStsNoErr` Indicates no error.

`ippStsNullPtrErr` Indicates an error condition if *pVLCState* pointer is `NULL`.

`ippStsRangeErr` Indicates an error condition if *adaptType* is not equal to `ippVLCScanVert`, `ippVLCScanHoriz`, or `ippVLCScanRaster`.

## VLCScan4x4DC\_JPEGXR

*Performs adaptive scanning of the DC coefficients.*

---

### Syntax

```
IppStatus ippivLCScan4x4DC_JPEGXR_32s(const Ipp32s pSrcCoeffs[16], Ipp32u numModelBits,
Ipp32s pDstResidual[16], Ipp32s* pDstCoeffsScan, Ipp32u* pNumNonZero, IppiVLCState_JPEGXR*
pVLCState);
```

### Parameters

*pSrcCoeffs* Pointer to the source vector of DC coefficient in scan order.

*numModelBits* Maximal number of bits to encode.

*pDstResidual* Pointer to the destination prediction residuals.

*pDstCoeffsScan* Pointer to the destination scan coefficients with statistic.

*pNumNonZero* Pointer to the number of coefficients with non-zero model bits.

*pVLCState* Pointer to the VLC state structure.

## Description

The function `ippiVLCScan4x4DC_JPEGXR` is declared in the `ippj.h` file.

This function performs adaptive scanning of the DC coefficients and calculates adaptive scan residuals.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointers is <code>NULL</code> .
<code>ippStsRangeErr</code>	Indicates an error condition if <code>numModelBits</code> is greater than 15.

## VLCEncode\_JPEGXR

*Performs VLC encoding.*

---

## Syntax

```
ippStatus ippiVLCEncode4x4Flex_JPEGXR_32slu_C1R(const Ipp32s* pSrc, Ipp32u srcStep, IppBool
chroma, Ipp32u numModelBits, Ipp32u numTrimBits, Ipp32s pDstResidual[16], Ipp8u**
ppBitStream, Ipp32u* pBitsBuf, Ipp32u* pNumBits, Ipp32u* pNumNonZero, IppiVLCState_JPEGXR*
pVLCState);
```

```
ippStatus ippiVLCEncode4x4Flex_JPEGXR_16slu_C1R(const Ipp16s* pSrc, Ipp32u srcStep, IppBool
chroma, Ipp32u numModelBits, Ipp32u numTrimBits, Ipp32s pDstResidual[16], Ipp8u**
ppBitStream, Ipp32u* pBitsBuf, Ipp32u* pNumBits, Ipp32u* pNumNonZero, IppiVLCState_JPEGXR*
pVLCState);
```

```
ippStatus ippiVLCEncode4x4_JPEGXR_32slu_C1R(const Ipp32s* pSrc, Ipp32u srcStep, IppBool
chroma, Ipp32u numModelBits, Ipp8u** ppBitStream, Ipp32u* pBitsBuf, Ipp32u* pNumBits,
Ipp32u* pNumNonZero, IppiVLCState_JPEGXR* pVLCState);
```

```
ippStatus ippiVLCEncode4x4_JPEGXR_16slu_C1R(const Ipp16s* pSrc, Ipp32u srcStep, IppBool
chroma, Ipp32u numModelBits, Ipp8u** ppBitStream, Ipp32u* pBitsBuf, Ipp32u* pNumBits,
Ipp32u* pNumNonZero, IppiVLCState_JPEGXR* pVLCState);
```

```
ippStatus ippiVLCEncodeDC444_JPEGXR_32slu_C1R(const Ipp32s* pSrcCoeffsScan, Ipp32s
numCoeffs, IppBool chroma, Ipp8u** ppBitStream, Ipp32u* pBitsBuf, Ipp32u* pNumBits,
IppiVLCState_JPEGXR* pVLCState);
```

```
ippStatus ippiVLCEncodeDC422_JPEGXR_32slu_C1R(const Ipp32s* pSrcCoeffsScan[14], Ipp32s
numCoeffs, Ipp8u** ppBitStream, Ipp32u* pBitsBuf, Ipp32u* pNumBits, IppiVLCState_JPEGXR*
pVLCState);
```

```
ippStatus ippiVLCEncodeDC420_JPEGXR_32slu_C1R(const Ipp32s* pSrcCoeffsScan[6], Ipp32s
numCoeffs, Ipp8u** ppBitStream, Ipp32u* pBitsBuf, Ipp32u* pNumBits, IppiVLCState_JPEGXR*
pVLCState);
```

## Parameters

*pSrc* Pointer to the source 4x4 block of samples.

<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pSrcCoeffsScan</i>	Pointer to the source vector of DC coefficients scanned.
<i>chroma</i>	Channel type flag: <code>ippFalse</code> - luma channel, <code>ippTrue</code> - chroma channel.
<i>numModelBits</i>	Maximal number of bits to encode.
<i>numTrimBits</i>	Number of bits to trim.
<i>pDstResidual</i>	Pointer to the destination prediction residuals.
<i>ppBitStream</i>	Pointer to the moving pointer to the memory using for destination bitstream.
<i>pBitsBuf</i>	Pointer to the prefetch buffer with encoded bits that are not written into the bitstream.
<i>pNumBits</i>	Number of valid bits in the prefetch buffer.
<i>pNumNonZero</i>	Pointer to the number of coefficients with non-zero model bits.
<i>pVLCState</i>	Pointer to the VLC state structure.

## Description

The functions `ippiVLCDecode` are declared in the `ippj.h` file.

These functions perform entropy encoding of 4x4 block or DC samples with or without prediction residual calculation.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointers is <code>NULL</code> .
<code>ippStsRangeErr</code>	Indicates an error condition if <i>pNumBits</i> is greater than 32 or <i>numTrimBits</i> is greater than 15.

## VLCDecode\_JPEGXR

*Performs VLC decoding.*

---

## Syntax

```
ippStatus ippiVLCDecode4x4_JPEGXR_1u32s_C1R(const Ipp8u** ppBitStream, Ipp32u* pBitsBuf,
Ipp32u* pNumBits, Ipp32s dequantMult, Ipp32s* pDst, Ipp32u dstStep, IppBool chroma, Ipp32u*
pNumNonZero, IppiVLCState_JPEGXR* pVLCState);
```

```
ippStatus ippiVLCDecode4x4_JPEGXR_1u16s_C1R(const Ipp8u** ppBitStream, Ipp32u* pBitsBuf,
Ipp32u* pNumBits, Ipp16s dequantMult, Ipp16s* pDst, Ipp32u dstStep, IppBool chroma, Ipp32u*
pNumNonZero, IppiVLCState_JPEGXR* pVLCState);
```

```
ippStatus ippiVLCDecodeDC444_JPEGXR_1u32s_C1R(const Ipp8u** ppBitStream, Ipp32u* pBitsBuf,
Ipp32u* pNumBits, Ipp32s pDstCoeffsScan[16], IppBool chroma, Ipp32u* pNumNonZero,
IppiVLCState_JPEGXR* pVLCState);
```

```
IppStatus ippiVLCDecodeDC422_JPEGXR_1u32s_C1R(const Ipp8u** ppBitStream, Ipp32u* pBitsBuf,
Ipp32u* pNumBits, Ipp32s pDstCoeffsScan[14], Ipp32u* pNumNonZero, IppiVLCState_JPEGXR*
pVLCState);
```

```
IppStatus ippiVLCDecodeDC420_JPEGXR_1u32s_C1R(const Ipp8u** ppBitStream, Ipp32u* pBitsBuf,
Ipp32u* pNumBits, Ipp32s pDstCoeffsScan[6], Ipp32u* pNumNonZero, IppiVLCState_JPEGXR*
pVLCState);
```

## Parameters

<i>ppBitStream</i>	Pointer to the moving pointer to the source bitstream data.
<i>pBitsBuf</i>	Pointer to the prefetch buffer with bits that are fetched from the bitstream but not decoded yet.
<i>pNumBits</i>	Number of valid bits in the prefetch buffer.
<i>pDst</i>	Pointer to to the destination 4x4 block of samples.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>pDstCoeffsScan</i>	Pointer to to the destination vector of DC coefficients decoded in scan order.
<i>chroma</i>	Channel type flag: <code>ippFalse</code> - luma channel, <code>ippTrue</code> - chroma channel.
<i>pNumNonZero</i>	Pointer to the number of coefficients actually decoded.
<i>pVLCState</i>	Pointer to the VLC state structure.

## Description

The functions `ippiVLCDecode` are declared in the `ippj.h` file.

These functions perform entropy decoding of 4x4 block or DC samples.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointers is <code>NULL</code> .
<code>ippStsRangeErr</code>	Indicates an error condition if <i>pNumBits</i> is greater than 32 or <i>numTrimBits</i> is greater than 15.

## FLCEncode\_JPEGXR

*Performs FLC encoding.*

## Syntax

```
IppStatus ippiFLCEncode4x4_JPEGXR_16s1u_C1R(const Ipp16s* pSrc, Ipp32u srcStep, const
Ipp32s* pSrcResidual, Ipp8u** ppBitStream, Ipp32u* pBitsBuf, Ipp32u* pNumBits, Ipp32u
numFlexBits, Ipp32u numTrimBits, IppBool acFlag);
```

```
IppStatus ippiFLCEncode4x4_JPEGXR_32s1u_C1R(const Ipp32s* pSrc, Ipp32u srcStep, const
Ipp32s* pSrcResidual, Ipp8u** ppBitStream, Ipp32u* pBitsBuf, Ipp32u* pNumBits, Ipp32u
numFlexBits, Ipp32u numTrimBits, IppBool acFlag);
```

## Parameters

<i>pSrc</i>	Pointer to to the source 4x4 block of samples.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pSrcResidual</i>	Pointer to to the source prediction residuals.
<i>ppBitStream</i>	Pointer to the moving pointer to the memory using for destination bitstream.
<i>pBitsBuf</i>	Pointer to the prefetch buffer with encoded bits that are not written into the bitstream.
<i>pNumBits</i>	Pointer to number of valid bits in the prefetch buffer.
<i>numFlexBits</i>	Number of flexbits to encode.
<i>numTrimBits</i>	Number of bits to trim.
<i>acFlag</i>	Adaptive coding flag: if <code>ippTrue</code> - use residual data, if <code>ippFalse</code> - process only flexbits.

## Description

The function `ippiFLCEncode4x4_JPEGXR` is declared in the `ippj.h` file.

These functions perform fixed length encoding of the 4x4 block of sample flexbits.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointers is <code>NULL</code> .
<code>ippStsRangeErr</code>	Indicates an error condition if <i>pNumBits</i> is greater than 32, or if <i>numFlexBits</i> or <i>numTrimBits</i> is greater than 15.

## FLCDecode\_JPEGXR

*Performs FLC decoding.*

---

## Syntax

```
IppStatus ippiFLCDecode4x4_JPEGXR_1u16s_C1R(const Ipp8u** ppBitStream, Ipp32u* pBitsBuf,
Ipp32u* pNumBits, Ipp32s* pSrcDst, Ipp32u srcDstStep, Ipp32u numFlexBits, Ipp16s trimMult,
IppBool acFlag);
```

```
IppStatus ippiFLCDecode4x4_JPEGXR_1u32s_C1R(const Ipp8u** ppBitStream, Ipp32u* pBitsBuf,
Ipp32u* pNumBits, Ipp32s* pSrcDst, Ipp32u srcDstStep, Ipp32u numFlexBits, Ipp32s trimMult,
IppBool acFlag);
```



## Parameters

<i>ppBitStream</i>	Pointer to the moving pointer to the source bitstream data.
<i>pBitsBuf</i>	Pointer to the prefetch buffer with bits that are fetched from the bitstream but not decoded.
<i>pNumBits</i>	Number of valid bits in the prefetch buffer.
<i>pSrcDst</i>	Pointer to to the 4x4 block of samples of the source and destination image.
<i>srcDstStep</i>	Distance in bytes between starts of consecutive lines in the source and destination image.
<i>numFlexBits</i>	Number of flexbits to decode.
<i>trimMult</i>	Multiplier, which depends on the number of trimmed bits.
<i>acFlag</i>	Adaptive coding flag: if <code>ippTrue</code> - use residual data from the bitstream, if <code>ippFalse</code> - processes only flexbits.

## Description

The functions `ippiVLCDdecode` are declared in the `ippj.h` file.

These functions perform entropy decoding of 4x4 block or DC samples.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointers is <code>ISNULL</code> .
<code>ippStsRangeErr</code>	Indicates an error condition if <i>pNumBits</i> is greater than 32, or if <i>numFlexBits</i> is greater than 15.



This chapter describes the subset of the Intel® IPP functions designed for video coding applications. This subset is a library of functions for encoding and decoding of video data according to MPEG-1 ([ISO11172]), MPEG-2 ([ISO13818]), MPEG-4 ([ISO14496A]), DV ([IEC61834] and [SMPTE314M]), H.261 ([ITUH261]), H.263 ([ITUH263]), H.264 ([JVTG050] and [ITUH264]), AVS ([AVS]), and VC-1 ([SMPTE421M]) standards. These functions have a convenient interface and present an appropriate solution for both encoding and decoding pipeline and, as all other parts of Intel IPP, are intended for the development of high-performance cross-platform code.

In addition to common data types used in Intel IPP for image processing (see [Data Types](#) in Chapter 2), video coding functions subset defines its specific data type:

## Conventional vs Intel IPP Data Types

Bit Depth	Conventional Type	Intel IPP Type
32	signed int	Ipp32s
16	signed short	Ipp16s
8	signed char	Ipp8s
32	unsigned int	Ipp32u
16	unsigned short	Ipp16u
8	unsigned char	Ipp8u

Descriptors in the names of video coding functions (see [Function Naming](#) in Chapter 2) may indicate the standard according to which the function operates (MPEG1, MPEG2, MPEG4, H263, H264, DV, etc.) or the size of a block of elements processed together (16x16, 16x8, 8x16, 8x8, 8x4). The first number is the width of the block and the second is its height. For functions that process one channel, the descriptor C1 is used. In-place function names contain the descriptor I, while the functions are not in-place by default.

The use of video coding functions is demonstrated in Intel® IPP Samples downloadable from <http://www.intel.com/cd/software/products/asmo-na/eng/220046.htm>.

Descriptions of the Intel IPP video coding functions are grouped into the following sections:

- [General Functions](#)
- [MPEG-1 and MPEG-2](#)
- [DV](#)
- [MPEG-4](#)
- [H.261](#)
- [H.263](#)
- [H.264](#)
- [AVS](#)
- [VC-1](#)

## General Functions

These functions are universal and can be used in different codecs. This category includes functions of the following types:

- [Variable Length Decoding](#) functions that use Variable Length Code tables to decode video data.
- [Motion Compensation](#) functions that calculate a sum of the residual block and the predicted block for reconstruction of the source block.
- [Motion Estimation](#) functions that calculate:
  - residual block - difference between the source block and the predicted block
  - some characteristics of the residual block
  - some characteristics of the source block. These characteristics can be used for comparison of the blocks.
- [Scanning functions](#) that convert two-dimensional arrays into one-dimensional data and vice versa using the predefined scan pattern.
- [Color Conversion](#) functions that convert images with rotation of the destination image.
- [Video Processing](#) functions that are used to process decompressed video data.

See the table below for the complete list of Video Coding general functions.

### General Functions

Function Short Name	Description
<b>Variable Length Decoding</b>	
<a href="#">HuffmanTableInitAlloc</a>	Allocates memory and initializes structure for table with one-to-one code/value correspondence.
<a href="#">HuffmanRunLevelTableInitAlloc</a>	Allocates memory and initializes structure for table with one-to-two code/value correspondence.
<a href="#">DecodeHuffmanOne</a>	Decodes one code using specified table and gets one decoded value.
<a href="#">DecodeHuffmanPair</a>	Decodes one code using specified table and gets two decoded values.
<a href="#">HuffmanTableFree</a>	Frees memory allocated for VLC table.
<b>Motion Compensation</b>	
<a href="#">MC16x16</a>	Performs motion compensation for a predicted 16X16 block.
<a href="#">MC16x8</a>	Performs motion compensation for a predicted 16X8 block.
<a href="#">MC8x16</a>	Performs motion compensation for a predicted 8X16 block.
<a href="#">MC8x8</a>	Performs motion compensation for a predicted 8X8 block.
<a href="#">MC8x4</a>	Performs motion compensation for a predicted 8X4 block.
<a href="#">MC4x8</a>	Performs motion compensation for a predicted 4X8 block.

Function Short Name	Description
MC4x4	Performs motion compensation for a predicted 4X4 block.
MC2x4	Performs motion compensation for a predicted 2X4 block.
MC4x2	Performs motion compensation for a predicted 4X2 block.
MC2x2	Performs motion compensation for a predicted 2X2 block.
MC16x4	Performs motion compensation for a predicted UV 16X4 block.
MC16x8UV	Performs motion compensation for a predicted UV 16X8 block.
MC16x16B	Performs motion compensation for a bi-predicted 16X16 block.
MC16x8B	Performs motion compensation for a bi-predicted 16X8 block.
MC8x16B	Performs motion compensation for a bi-predicted 8X16 block.
MC8x8B	Performs motion compensation for a bi-predicted 8X8 block.
MC8x4B	Performs motion compensation for a bi-predicted 8X4 block.
MC4x8B	Performs motion compensation for a bi-predicted 4X8 block.
MC4x4B	Performs motion compensation for a bi-predicted 4X4 block.
MC2x4B	Performs motion compensation for a bi-predicted 2X4 block.
MC4x2B	Performs motion compensation for a bi-predicted 4X2 block.
MC2x2B	Performs motion compensation for a bi-predicted 2X2 block.
MC16x4B	Performs motion compensation for a bi-predicted UV 16X4 block.
MC16x8UVB	Performs motion compensation for a bi-predicted UV 16X8 block.
Copy8x8, Copy16x16	Copy the fixed size block to the destination block.
Copy8x4HP, Copy8x8HP, Copy16x8HP, Copy16x16HP	Copy fixed size blocks with half-pixel accuracy.
InterpolateAverage8x4, InterpolateAverage8x8, InterpolateAverage16x8, InterpolateAverage16x16	Interpolate the source block according to half-pixel offset and average the result with the destination block.

Function Short Name	Description
<a href="#">Add8x8</a>	Adds two blocks with saturation.
<a href="#">Add8x8HP</a>	Adds blocks interpolated with half-pixel accuracy prediction to difference with saturation.
<a href="#">AddC8x8</a>	Adds a constant to an 8x8 block with saturation.
<a href="#">Average8x8</a> , <a href="#">Average16x16</a>	Average two blocks.

### Motion Estimation

#### Difference Evaluation

<a href="#">GetDiff16x16</a>	Evaluates difference between current predicted and reference blocks of 16x16 elements.
<a href="#">GetDiff16x8</a>	Evaluates difference between current predicted and reference blocks of 16x8 elements.
<a href="#">GetDiff8x8</a>	Evaluates difference between current predicted and reference blocks of 8x8 elements.
<a href="#">GetDiff8x16</a>	Evaluates difference between current predicted and reference blocks of 8x16 elements.
<a href="#">GetDiff8x4</a>	Evaluates difference between current predicted and reference blocks of 8x4 elements.
<a href="#">GetDiff4x4</a>	Evaluates difference between current predicted and reference 4x4 blocks.
<a href="#">GetDiff16x16B</a>	Evaluates difference between current bi-predicted and mean of two reference blocks of 16x16 elements.
<a href="#">GetDiff16x8B</a>	Evaluates difference between current bi-predicted and mean of two reference blocks of 16x8 elements.
<a href="#">GetDiff8x8B</a>	Evaluates difference between current bi-predicted and mean of two reference blocks of 8x8 elements.
<a href="#">GetDiff8x16B</a>	Evaluates difference between current bi-predicted and mean of two reference blocks of 8x16 elements.
<a href="#">GetDiff8x4B</a>	Evaluates difference between current bi-predicted and mean of two reference blocks of 8x4 elements.
<a href="#">Sub4x4</a> , <a href="#">Sub8x8</a> , <a href="#">Sub16x16</a>	Subtract two blocks and store the result in the third block.
<a href="#">SubSAD8x8</a>	Subtracts two block, stores the result in the third block and computes a sum of absolute differences.

#### Sum of Squares of Differences Evaluation

Function Short Name	Description
<a href="#">SqrDiff16x16</a>	Evaluates sum of squares of differences between current and reference blocks.
<a href="#">SqrDiff16x16B</a>	Evaluates sum of squares of differences between the current bi-predicted block and the mean of two reference blocks.
<a href="#">SSD8x8</a>	Evaluates sum of squares of differences between current and reference 8X8 blocks.
<a href="#">SSD4x4</a>	Evaluates sum of squares of differences between current and reference 4X4 blocks.
<b>Block Variance and Mean Evaluation</b>	
<a href="#">VarMean8x8</a>	Evaluates variance and mean of a 8X8 block of unsigned char or short integer values.
<b>Evaluation of Variances and Means of Blocks of Difference Between Two Blocks</b>	
<a href="#">VarMeanDiff16x16</a>	Evaluates variances and means of four 8x8 blocks of difference between two 16x16 blocks.
<a href="#">VarMeanDiff16x8</a>	Evaluates variances and means of four 8x8 blocks of difference between two 16x8 blocks.
<b>Block Variance Evaluation</b>	
<a href="#">Variance16x16</a>	Evaluates variance of the current block.
<b>Evaluation of Block Deviation</b>	
<a href="#">MeanAbsDev8x8</a>	Evaluates mean absolute deviation for an 8x8 block.
<a href="#">MeanAbsDev16x16</a>	Evaluates mean absolute deviation for a 16x16 block.
<b>Edges Detection</b>	
<a href="#">EdgesDetect16x16</a>	Detects edges inside a 16X16 block.
<b>SAD Functions</b>	
<a href="#">SAD16x16</a>	Evaluates sum of absolute difference between current and reference 16X16 blocks.
<a href="#">SAD16x8</a>	Evaluates sum of absolute difference between current and reference 16X8 blocks.
<a href="#">SAD8x16</a>	Evaluates sum of absolute difference between current and reference 8X16 blocks.

Function Short Name	Description
<a href="#">SAD8x8</a>	Evaluates sum of absolute difference between current and reference 8X8 blocks.
<a href="#">SAD8x4</a>	Evaluates sum of absolute difference between current and reference 8X4 blocks.
<a href="#">SAD4x8</a>	Evaluates sum of absolute difference between current and reference 4X8 blocks.
<a href="#">SAD4x4</a>	Evaluates sum of absolute difference between current and reference 4X4 blocks.
<a href="#">SAD16x16Blocks8x8</a>	Evaluates four partial sums of absolute differences between current and reference 16X16 blocks.
<a href="#">SAD16x16Blocks4x4</a>	Evaluates 16 partial sums of absolute differences between current and reference 16X16 blocks.
<a href="#">SATD16x16</a>	Evaluates sum of absolute transformed differences between current and reference 16X16 blocks using 4x4 transform.
<a href="#">SATD16x8</a>	Evaluates sum of absolute transformed differences between current and reference 16X8 blocks using 4x4 transform.
<a href="#">SATD8x16</a>	Evaluates sum of absolute transformed differences between current and reference 8X16 blocks using 4x4 transform.
<a href="#">SATD8x8</a>	Evaluates sum of absolute transformed differences between current and reference 8X8 blocks using 4x4 transform.
<a href="#">SATD8x4</a>	Evaluates sum of absolute transformed differences between current and reference 8X4 blocks using 4x4 transform.
<a href="#">SATD4x8</a>	Evaluates sum of absolute transformed differences between current and reference 4X8 blocks using 4x4 transform.
<a href="#">SATD4x4</a>	Evaluates sum of absolute transformed differences between current and reference 4X4 blocks using 4x4 transform.
<a href="#">SAT8x8D</a>	Evaluates sum of absolute transformed differences between current and reference 8X8 blocks using 8x8 transform.
<a href="#">FrameFieldSAD16x16</a>	Calculates SAD between field lines and SAD between frame lines of a 16x16 block.
<b>Sum of Differences Evaluation</b>	
<a href="#">SumsDiff16x16Blocks4x4</a>	Evaluates difference between current and reference 4X4 blocks and calculates sums of 4X4 residual blocks elements for 16X16 blocks.



Function Short Name	Description
<code>SumsDiff8x8Blocks4x4</code>	Evaluates difference between current and reference 4X4 blocks and calculates sums of 4X4 residual blocks elements for 8X8 blocks.
<b>Scanning</b>	
<code>ScanInv</code>	Performs classical zigzag, alternate-horizontal, or alternate-vertical inverse scan on a block stored in a compact buffer.
<code>ScanFwd</code>	Performs classical zigzag, alternate-horizontal, or alternate-vertical forward scan on a block.
<b>Color Conversion</b>	
<code>CbYCr422ToYCbCr420_Rotate</code>	Converts 4:2:2 CbYCr image to 4:2:0 YCbCr image with rotation.
<code>ResizeCCRotate</code>	Creates a low-resolution preview image for high-resolution video or still capture applications.
<b>Video Processing</b>	
<b>Deinterlacing</b>	
<code>DeinterlaceFilterTriangle</code>	Deinterlaces video plane.
<code>DeinterlaceFilterCAVT</code>	Performs deinterlacing of two-field image.
<code>Median</code>	Creates an image consisting of median values of three source images.
<code>DeinterlaceMedianThreshold</code>	Performs median deinterlacing with threshold.
<code>DeinterlaceEdgeDetect</code>	Generates image field using the <code>EdgeDetect</code> filter.
<code>DeinterlaceMotionAdaptive</code>	Performs deinterlacing using temporal and spatial interpolations.
<code>DeinterlaceBlendGetSize</code>	Calculates the size of internal structure for <code>DeinterlaceBlend</code> function.
<code>DeinterlaceBlendInit</code>	Initializes an internal structure for the <code>DeinterlaceBlend</code> function.
<code>DeinterlaceBlendInitAlloc</code>	Allocates and initializes an internal structure for the <code>DeinterlaceBlend</code> function.
<code>DeinterlaceBlend</code>	Calculates output pixel as alpha blends of the results of two filters applied to input pixels.

Function Short Name	Description
<code>DeinterlaceBlendFree</code>	Releases memory allocated by the <code>DeinterlaceBlendInitAlloc</code> function.
<b>Denoising</b>	
<code>FilterDenoiseCASTInit</code>	Initializes a denoise specification structure for content adaptive spatio-temporal noise reduction filtering.
<code>FilterDenoiseCAST</code>	Performs content adaptive spatio-temporal (CAST) noise reduction filtering.
<code>FilterDenoiseSmooth</code>	Performs spatial noise reduction (SNR) filtering.
<code>FilterDenoiseAdaptiveInitAlloc</code>	Creates and initializes a denoise specification structure for spatio-temporal adaptive noise reduction filtering.
<code>FilterDenoiseAdaptiveFree</code>	Closes a denoise specification structure for spatio-temporal adaptive noise reduction filtering.
<code>FilterDenoiseAdaptive</code>	Performs spatio-temporal adaptive noise reduction filtering.
<code>FilterDenoiseMosquitoInitAlloc</code>	Creates and initializes a denoise specification structure for spatio-temporal motion adaptive mosquito noise reduction filtering.
<code>FilterDenoiseMosquitoFree</code>	Closes a denoise specification structure for spatio-temporal motion adaptive mosquito noise reduction filtering.
<code>FilterDenoiseMosquito</code>	Performs spatio-temporal motion adaptive mosquito noise reduction filtering.

## Structures and Enumerators

The structure `IppMotionVector` can be used in both H.263 and MPEG-4 video processing functions and is defined as follows:

```
typedef struct {
    Ipp16s dx;
    Ipp16s dy;
} IppMotionVector;
```

The following enumerator indicates the type of a decoded macroblock or a macroblock to be encoded. `IPPVC_MB_STUFFING` indicates that a bit stuffing codeword was decoded from the bitstream or should be encoded into the bitstream.

```
enum
{
    IPPVC_MBTYPETYPE_INTER = 0,      /* valid in P-picture or P-VOP*/
    IPPVC_MBTYPETYPE_INTER_Q = 1,    /* P-picture or P-VOP*/
    IPPVC_MBTYPETYPE_INTER4V = 2,    /* P-picture or P-VOP*/
    IPPVC_MBTYPETYPE_INTRA = 3,      /* I- and P-picture, or I- and P-VOP*/
    IPPVC_MBTYPETYPE_INTRA_Q = 4,    /* I- and P-picture, or I- and P-VOP*/
    IPPVC_MBTYPETYPE_INTER4V_Q = 5,  /* P-picture or P-VOP*/
    IPPVC_MBTYPETYPE_DIRECT = 6,     /* B-picture or B-VOP*/
}
```

```

IPPVC_MBTTYPE_INTERPOLATE = 7, /* B-picture or B-VOP*/
IPPVC_MBTTYPE_BACKWARD = 8,  /* B-picture or B-VOP*/
IPPVC_MBTTYPE_FORWARD = 9,   /* B-picture or B-VOP*/
IPPVC_MB_STUFFING = 255      /* indicates bit stuffing codeword*/
};

```

The following enumerator indicates the type of scan performed on DCT coefficients:

```

enum
{
    IPPVC_SCAN_NONE = -1,      no scan to be performed
    IPPVC_SCAN_ZIGZAG = 0,     classical zigzag scan
    IPPVC_SCAN_VERTICAL = 1,   alternate vertical scan
    IPPVC_SCAN_HORIZONTAL = 2  alternate horizontal scan
};

```

See typical scan patterns in [Figure "Standard Decoding Pipeline"](#) and the scanning process according to the classical scanning pattern in [Figure "Standard Decoding Pipeline"](#).

The following enumerator indicates the type of interpolation performed on a picture. This enumerator is used, for example, in spatial scalability mode.

```

enum
{
    IPPVC_INTERP_NONE = 0,      no interpolation
    IPPVC_INTERP_HORIZONTAL = 1, 1-D horizontal interpolation
    IPPVC_INTERP_VERTICAL = 2,   1-D vertical interpolation
    IPPVC_INTERP_2D = 3         2-D interpolation
};

```

The following enumerator indicates the type of rotate operation for general color conversion functions:

```

enum {
    IPPVC_ROTATE_DISABLE = 0,
    IPPVC_ROTATE_90CCW = 1,
    IPPVC_ROTATE_90CW = 2,
    IPPVC_ROTATE_180 = 3
};

```

The following enumerator indicates the type of color space conversion for general color conversion functions:

```

enum
{
    IPPVC_CbYCr422ToBGR565 = 0,
    IPPVC_CbYCr422ToBGR555 = 1
};

```

The enumerator `IPPVC_MC_APX` indicates the type of prediction for motion compensation or motion estimation.

```

typedef enum _IPPVC_MC_APX{

    IPPVC_MC_APX_FF = 0x0,

    IPPVC_MC_APX_FH = 0x4,

    IPPVC_MC_APX_HF = 0x8,

    IPPVC_MC_APX_HH = 0x0c

}IPPVC_MC_APX;

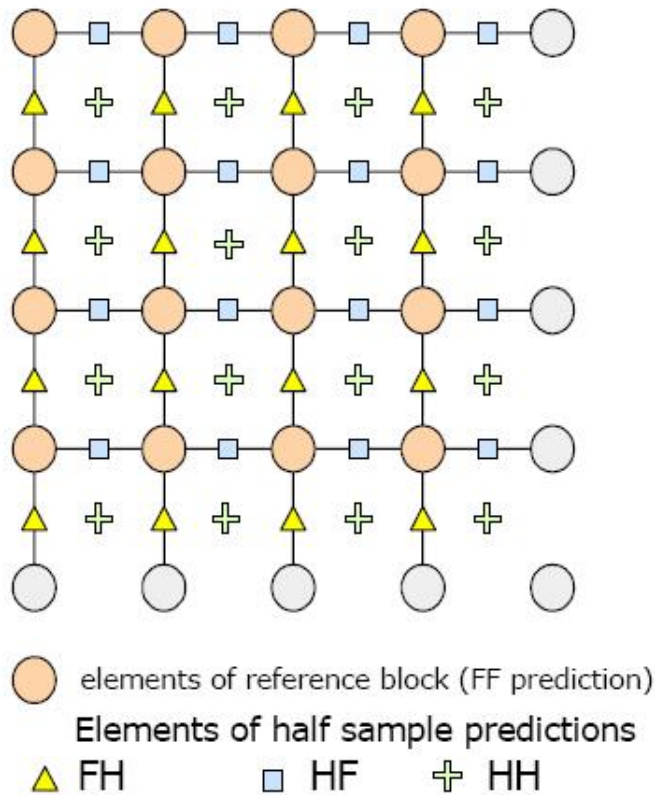
```

The first descriptors `FF`, `FH`, `HF`, `HH` show whether prediction is accurate to full pel (`F`) or to half a pel (`H`). The first letter indicates the accuracy in horizontal direction and the second — in vertical direction.

Most of general functions use parameter `mcType` (see enumeration `IPPVC_MC_APX`). It is used for calculating prediction on the basis of block in the reference frame. See [Figure "Predictions for 4x4 Reference Block"](#).

- If  $mcType = IPPVC\_MC\_APX\_FF$ , the reference block is used as the prediction block.
- If  $mcType = IPPVC\_MC\_APX\_FH$  or  $IPPVC\_MC\_APX\_HF$ , each element of the prediction block is calculated as average of two elements of the reference block.
- If  $mcType = IPPVC\_MC\_APX\_HH$ , each element of the prediction block is calculated as average of four elements of the reference block.

### Predictions for 4x4 Reference Block



### UV Block

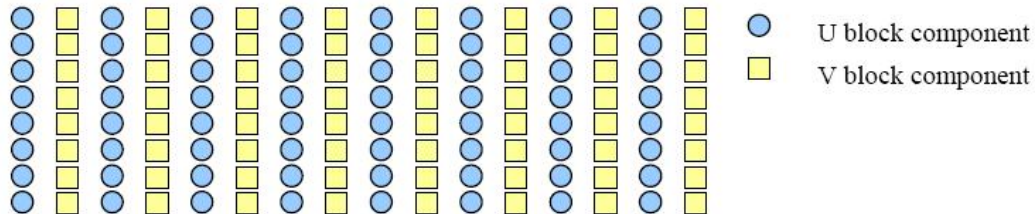
$UV$  block of size  $(2 \times H) \times W$  combines  $U$  block of size  $H \times W$  and  $V$  block of size  $H \times W$ . The formula for  $UV$  is as follows:

$$UV[2 \times i + 1, j] = V[i, j]$$

$$UV[2 \times i, j] = U[i, j],$$

where  $i=[0, H-1]$  and  $j=[0, W-1]$ . See the figure below for an example of a  $UV$  block.

### 16x8 UV Block (8x8 U Block and 8x8 V Block)



## Variable Length Decoding

Video data in the bit stream is encoded with Variable Length Code (VLC) tables so that the shortest codes correspond to the most frequent values and the longer codes correspond to the less frequent values. Tables for each video standard contain specific possible codes and their values.

VLC decoding functions can work with two types of tables:

- Tables, in which one code corresponds to one value (see [Table "One-to-One Code/Value Correspondence"](#) for an example)
- Tables, in which one code corresponds to two values (see [Table "One-to-Two Code/Value Correspondence"](#) for an example).

### One-to-One Code/Value Correspondence

Code	Value
0001 1	-3
0011	-2
011	-1
1	0
010	1
0010	2
0001 0	3

### One-to-Two Code/Value Correspondence

Code	Value 1	Value 2
0001 1	1	0
0011	1	1
011	1	2
1	2	0
010	2	2
0010	3	2
0001 0	4	1

Video data VLC-decoding starts with memory allocation and table initialization. Then the decoding proper should be made and the allocated memory must be released.

**VLC Decoding Functions**

One-to-One Correspondence	One-to-Two Correspondence	Short Description
HuffmanTableInitAlloc	HuffmanRunLevelTableInitAlloc	Allocates memory and initializes structure that is used for decoding.
DecodeHuffmanOne	DecodeHuffmanPair	Decodes one code using specified table.
HuffmanTableFree		Frees memory allocated for VLC table.

For using `ippiVCHuffmanInitAlloc_32s` and `ippiVCHuffmanInitAllocRL_32s` functions, the source table should have the following structure:

**Example**

```
static Ipp32s Table[] =
{
    max_bits,           // The maximum length of code
    total_subt,         // The total number of all subtables
    sub_sz1,            // The sizes of subtables. Their sum must
    sub_sz2,            // be equal to the maximum length of codes
    ...,               // max_bits and their number should be
                        // equal to total_subt.
    N1,                // The number of 1-bit codes
    Code1, Value1_1, [Value2_1] //
    Code2, Value1_2, [Value2_2] // The 1-bit codes
    ...,              //
    N2,                // The number of 2-bit codes
    Code1, Value1_1, [Value2_1] //
    Code2, Value1_2, [Value2_2] // The 2-bit codes
    ...,              //
    Nm,                // The number of max_bits, -bit codes
    Code1, Value1_1, [Value2_1] //
    Code2, Value1_2, [Value2_2] // max_bits codes
    ...,              //
    -1                 // The significant value to indicate the
                        // end of table
};
```

**Notes:**

- The values  $Value2_i$  are used for the second type table (one-to-two code/value correspondence) and must be in the range  $[-32768, 32767]$  or  $[0, 65535]$ .
- The values  $Value1_i$  must be in the range:
  - $[-8388608, 8388607]$  or  $[0, 16777215]$  in the case of first type table (one-to-one correspondence)
  - $[-128, 127]$  or  $[0, 255]$  in the case of second type table (one-to-two correspondence).

**Using Subtables**

Subtables are used for large source tables. Internally, VLC decoding is done by table lookups. A straightforward approach would be to have one lookup of a lookup table of size  $2^L$ , where  $L$  is the maximum number of code bits. This approach however is proven to be memory inefficient. Therefore, subtables are used to balance the memory and the lookup speed. Using subtables reduces the memory size but increases the number of lookups. The greater is the number of subtables, the smaller is the structure size but the number of decoding operations increases. A table structure of this kind provides the optimal ratio between the table redundancy and the number

of decoding operations. Each subtable works on some numbers of bits out of  $L$ . In the `ippiVCHuffmanInitAlloc_32s` and `ippiVCHuffmanInitAllocRL_32s` functions subtables sizes are specified manually. Any combinations of subtables are possible as long as their sum adds up to  $L$ .

Consider an example of VLC decoding. The table below gives the codes and their values.

#### Example of VLC Decoding

Code	Value
0	A
1100	B
101	C
1110	D
100	E
11110	F
11111	G
1101	H

First, let us consider decoding when subtables are not used. In this case, the source table should look as follows:

```
static Ipp32s Table[]=
{
    5,                // The maximum length of code
    1,                // The total number of all subtables
    5,                // The size of 1 subtable
    1,                // The number of 1-bit codes
    0/*0*/ ,A,        // code1, value1
    0,                // The number of 2-bit codes
    2                 // The number of 3-bit codes
    5/*101*/ ,C,       // code1, value1
    4/*100*/ ,E,       // code2, value2
    3                 // The number of 4-bit codes
    12/*1100*/ ,B,     // code1, value1
    14/*1110*/ ,D,     // code2, value2
    13/*1101*/ ,H,     // code3, value3
    2                 // The number of 4-bit codes
    30/*11110*/ ,F,    // code1, value1
    31/*11111*/ ,G,    // code2, value2
    -1
};
```

The function `ippiVCHuffmanInitAlloc_32s` is used to form a structure that contains the following data:

## Structure Data For a Case With One Subtable

Code (length=5)	Value	Length
00000	A	1
00000	A	1
...	...	
01111	A	1
10100	C	3
10101	C	3
10110	C	3
10111	C	3
10000	E	3
10000	E	3
10000	E	3
10000	E	3
11000	B	4
11001	B	4
11100	D	4
11101	D	4
11010	H	4
11011	H	4
11110	F	5
11111	G	5

In process of decoding, 5 bits are extracted from the bitstream. They are used to find a corresponding code in the above table and matching value and length. (5-Length) bits are returned to the bitstream.



Now, consider an example when two subtables are used for decoding. The source table structure will look as follows:

```
static Ipp32s Table[]=  
{  
5,           // The maximum length of code  
2,           // The total number of all subtables  
3,           // The size of subtable 1  
2,           // The size of subtable 2  
1,           // The number of 1-bit codes  
0/*0*/,A,    // code1, value1  
0,           // The number of 2-bit codes  
2           // The number of 3-bit codes  
5/*101*/,C,   // code1, value1  
4/*100*/,E,   // code2, value2  
3           // The number of 4-bit codes  
12/*1100*/,B, // code1, value1  
14/*1110*/,D, // code2, value2  
13/*1101*/,H, // code3, value3  
2           // The number of 4-bit codes  
30/*11110*/,F, // code1, value1  
31/*11111*/,G, // code2, value2  
-1  
};
```

The function `ippiVCHuffmanInitAlloc_32s` is used to form a structure that contains two subtables:

### VLC Subtables

Length	1	1	1	1	3	3	3	3
Code (length = 3)	000	001	010	011	100	101	110	111
Value	A	A	A	A	E	C		

SubTable 1

Length	1	1	1	1
Code (length = 2)	00	01	10	11
Value	B	B	H	H

SubTable 2

Length	1	1	2	2
Code (length = 2)	00	01	10	11
Value	D	D	F	G

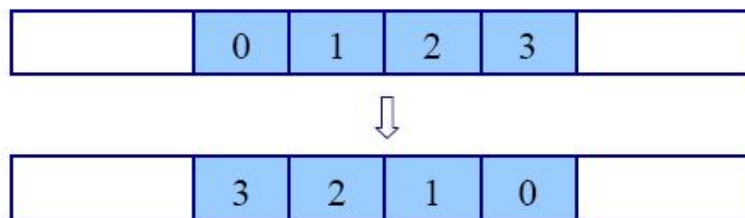
Subtable 1 contains the values with the code length that does not exceed 3. Subtable 2 contains the values with the code length that exceeds 3, with the corresponding code consisting of 3-bit code from Subtable 1 and 2-bit code from Subtable 2.

In process of decoding 3 bits are extracted from the bitstream. They are used to find a corresponding code in Subtable 1. The matching value and length are found if the code is other than 100 or 111. (3-Length) bits are returned to the bitstream. Otherwise, two more bits are extracted from the bitstream to find respective value and length in Subtable 2 and (2-Length) bits are returned to the bitstream afterwards.

### *ppBitStream* and *pOffset* Parameters

Before decoding and applying the Intel IPP functions, all bytes in each 32-bit double word in the bit stream must be flipped (see the figure below) to improve performance of decoding functions.

### Flipping of Bytes



The parameters *ppBitStream* and *pOffset* define start position for a subsequent code: current element of the array and position in this element. These parameters are updated by function.

*ppBitStream*

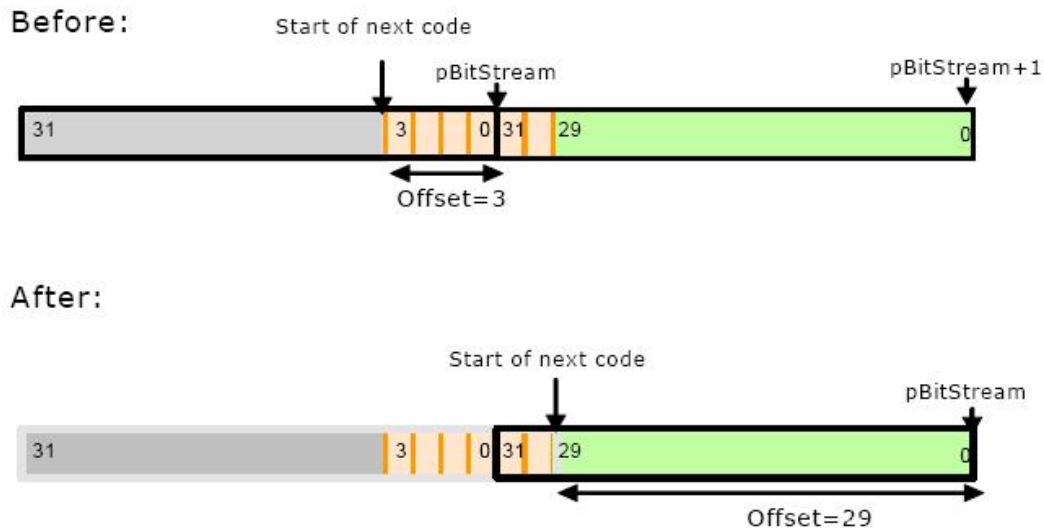
Double pointer to the current position in the bitstream.

*pOffset*

Pointer to the offset between the bit that *\*\*ppBitStream* points to and the start of the code.

See the figure below for an example of getting one code from the bitstream.

### Getting One Code From Bitstream



#### Optimization Notice

Intel's compilers may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include SSE2, SSE3, and SSSE3 instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on microprocessors not manufactured by Intel. Microprocessor-dependent optimizations in this product are intended for use with Intel microprocessors. Certain optimizations not specific to Intel microarchitecture are reserved for Intel microprocessors. Please refer to the applicable product User and Reference Guides for more information regarding the specific instruction sets covered by this notice.

Notice revision #20110804

The VLC functions are used in the decoders included into Intel® IPP Samples. See [the introduction](#) to the chapter.

## HuffmanTableInitAlloc

*Allocates memory and initializes structure for table with one-to-one code/value correspondence.*

### Syntax

```
IppStatus ippiHuffmanTableInitAlloc_32s(const Ipp32s* pSrcTable, IppVCHuffmanSpec_32s** ppDstSpec);
```

## Parameters

<i>pSrcTable</i>	Pointer to the source table (see <a href="#">Source Table Structure</a> ).
<i>ppDstSpec</i>	Double pointer to the destination decoding table.

## Description

This function is declared in the `ippvc.h` header file. The function `ippiHuffmanTableInitAlloc_32s` allocates memory and initializes a structure for a table in which one code corresponds to one value (See [Table "One-to-One Code/Value Correspondence"](#)). This structure is used for decoding.

See [Table "MPEG-1 and MPEG-2 Video Decoding Functions"](#) for details of specific use of the function in the MPEG1 and MPEG2 standards.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when at least one input pointer is <code>NULL</code> .
<code>ippStsMemAllocErr</code>	Indicates an error when no memory is allocated.

## HuffmanRunLevelTableInitAlloc

*Allocates memory and initializes structure for table with one-to-two code/value correspondence.*

---

## Syntax

```
ippStatus ippiHuffmanRunLevelTableInitAlloc_32s(const Ipp32s* pSrcTable,
IppVCHuffmanSpec_32s** ppDstSpec);
```

## Parameters

<i>pSrcTable</i>	Pointer to the source table (see <a href="#">Source Table Structure</a> ).
<i>ppDstSpec</i>	Double pointer to the destination decoding table.

## Description

This function is declared in the `ippvc.h` header file. The function `ippiHuffmanRunLevelTableInitAlloc_32s` allocates memory and initializes structure for a table in which one code corresponds to two value. This structure is used for decoding.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when at least one input pointer is <code>NULL</code> .
<code>ippStsMemAllocErr</code>	Indicates an error when no memory is allocated.

## HuffmanTableFree

*Frees memory allocated for VLC table.*

---

### Syntax

```
IppStatus ippiHuffmanTableFree_32s(IppVCHuffmanSpec_32s** ppDecodeTable);
```

### Parameters

*ppDecodeTable* Double pointer to the decoding table.

### Description

This function is declared in the `ippvc.h` header file. The function `ippiHuffmanTableFree_32s` frees memory at *ppDecodeTable* allocated for the VLC table.

### Return Values

`ippStsNoErr` Indicates no error.  
`ippStsNullPtrErr` Indicates an error when at least one input pointer is NULL.

## DecodeHuffmanOne

*Decodes one code using specified table and gets one decoded value.*

---

### Syntax

```
IppStatus ippiDecodeHuffmanOne_lu32s(Ipp32u** ppBitStream, Ipp32s* pOffset, Ipp32s* pDst,
const IppVCHuffmanSpec_32s* pDecodeTable);
```

### Parameters

*ppBitStream* Double pointer to the current position in the bitstream.  
*pOffset* Pointer to offset between the bit that *ppBitStream* points to and the start of the code. *pOffset* may vary from zero to 31.  
*pDst* Pointer to the destination result, that is, one value.  
*pDecodeTable* Pointer to the decoding table.

### Description

This function is declared in the `ippvc.h` header file. The function `ippiDecodeHuffmanOne_lu32s` decodes one code from the bitstream using a specified table, sends the result (one value) to the destination data, and resets the pointers to new positions (See example in [Figure "Getting One Code From Bitstream"](#)). The function uses the table derived by the [HuffmanTableInitAlloc](#) function.

### Return Values

`ippStsNoErr` Indicates no error.  
`ippStsNullPtrErr` Indicates an error when at least one input pointer is NULL.

`ippStsH263VLCCodeErr` Indicates an error in process of decoding.

## DecodeHuffmanPair

*Decodes one code using specified table and gets two decoded values.*

---

### Syntax

```
IppStatus ippDecodeHuffmanPair_lu16s(Ipp32u** ppBitStream, Ipp32s* pOffset, const
IppVCHuffmanSpec_32s* pDecodeTable, Ipp8s* pFirst, Ipp16s* pSecond);
```

### Parameters

<code>ppBitStream</code>	Double pointer to the current position in the bitstream.
<code>pOffset</code>	Pointer to offset between the bit that <code>ppBitStream</code> points to and the start of the code. <code>pOffset</code> may vary from zero to 31.
<code>pDecodeTable</code>	Pointer to the decoding table.
<code>pFirst</code>	Pointer to the first destination value.
<code>pSecond</code>	Pointer to the second destination value.

### Description

This function is declared in the `ippvc.h` header file. The function `ippDecodeHuffmanPair_lu16s` decodes one code from the bitstream using a specified table, sends the result (two values) to destination data, and resets the pointers to new positions (See example in [Figure "Getting One Code From Bitstream"](#)). The function uses the table derived by the [HuffmanRunLevelTableInitAlloc](#) function.

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when at least one input pointer is <code>NULL</code> .
<code>ippStsH263VLCCodeErr</code>	Indicates an error in process of decoding.

## Motion Compensation

These functions calculate sum of the residual block and the predicted block for reconstruction of the source block. After processing, the data is converted with saturation from `Ipp16s` to `Ipp8u` type.

Motion compensation functions are divided into two groups:

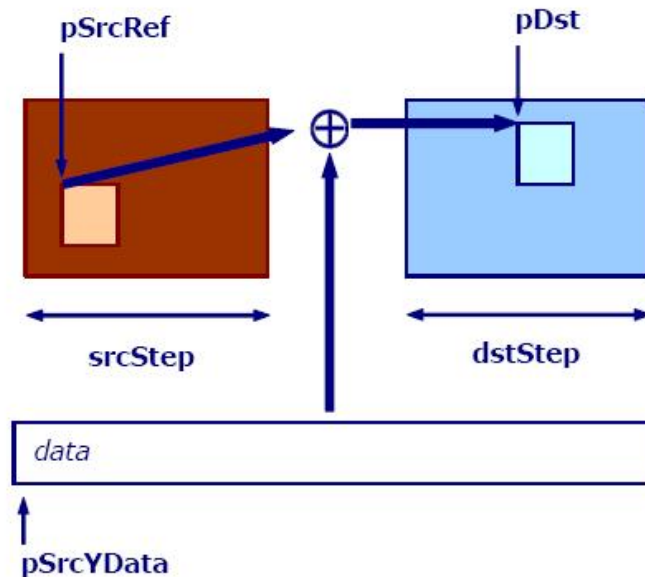
- Functions for [predicted blocks](#), using one reference block for prediction
- Functions for [bi-predicted blocks](#), using two reference blocks for prediction; in this case, predictions are first calculated for each reference block, and then prediction is calculated as average of two predictions.

The use of some Intel IPP motion compensation functions is demonstrated in Intel® IPP Samples downloadable from <http://www.intel.com/cd/software/products/asmo-na/eng/220046.htm> .

## Predicted Blocks

These functions use one reference block for prediction.

### Motion Compensation Scheme for Predicted Block



## MC16x16

*Performs motion compensation for predicted 16X16 block.*

### Syntax

```
IppStatus ippiMC16x16_8u_C1(const Ipp8u* pSrcRef, Ipp32s srcStep, const Ipp16s* pSrcYData,
Ipp32s srcYDataStep, Ipp8u* pDst, Ipp32s dstStep, Ipp32s mcType, Ipp32s roundControl);
```

### Parameters

<i>pSrcRef</i>	Pointer to the reference intra block.
<i>srcStep</i>	Size of the row in bytes, specifying the aligned reference frame width.
<i>pSrcYData</i>	Pointer to the data obtained after inverse DCT.
<i>srcYDataStep</i>	Number of bytes, specifying the width of the aligned data obtained after inverse DCT.
<i>pDst</i>	Pointer to the destination predicted block.
<i>dstStep</i>	Size of the row in bytes, specifying the aligned destination frame width.
<i>mcType</i>	MC type <a href="#">IPPVC_MC_APX</a> .
<i>roundControl</i>	Parameter that determines type of rounding for half a pel approximation; may be 0 or 1.

## Description

This function is declared in the `ippvc.h` header file. The function `ippiMC16x16_8u_C1` calculates a sum of a 16x16 residual block and a 16x16 predicted block for reconstruction of the source block (See [Figure Motion Compensation Scheme for Predicted Block](#)). Prediction is calculated on the basis of the reference block and `mcType` (see [IPPVC\\_MC\\_APX](#)).

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when at least one input pointer is <code>NULL</code> .

## MC16x8

Performs motion compensation for predicted 16X8 block.

## Syntax

```
ippStatus ippiMC16x8_8u_C1(const Ipp8u* pSrcRef, Ipp32s srcStep, const Ipp16s* pSrcYData,
Ipp32s srcYDataStep, Ipp8u* pDst, Ipp32s dstStep, Ipp32s mcType, Ipp32s roundControl);
```

## Parameters

<code>pSrcRef</code>	Pointer to the reference intra block.
<code>srcStep</code>	Size of the row in bytes, specifying the aligned reference frame width.
<code>pSrcYData</code>	Pointer to the data obtained after inverse DCT.
<code>srcYDataStep</code>	Number of bytes, specifying the width of the aligned data obtained after inverse DCT.
<code>pDst</code>	Pointer to the destination predicted block.
<code>dstStep</code>	Size of the row in bytes, specifying the aligned destination frame width.
<code>mcType</code>	MC type <a href="#">IPPVC_MC_APX</a> .
<code>roundControl</code>	Parameter that determines type of rounding for half a pel approximation; may be 0 or 1.

## Description

This function is declared in the `ippvc.h` header file. The function `ippiMC16x8_8u_C1` calculates a sum of a 16x8 residual block and a 16x8 predicted block for reconstruction of the source block (See [Figure Motion Compensation Scheme for Predicted Block](#)). Prediction is calculated on the basis of the reference block and `mcType` (see [IPPVC\\_MC\\_APX](#)).

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when at least one input pointer is <code>NULL</code> .



## MC8x16

*Performs motion compensation for predicted 8X16 block.*

### Syntax

```
IppStatus ippiMC8x16_8u_C1(const Ipp8u* pSrcRef, Ipp32s srcStep, const Ipp16s* pSrcYData,
Ipp32s srcYDataStep, Ipp8u* pDst, Ipp32s dstStep, Ipp32s mcType, Ipp32s roundControl);
```

### Parameters

<i>pSrcRef</i>	Pointer to the reference intra block.
<i>srcStep</i>	Size of the row in bytes, specifying the aligned reference frame width.
<i>pSrcYData</i>	Pointer to the data obtained after inverse DCT.
<i>srcYDataStep</i>	Number of bytes, specifying the width of the aligned data obtained after inverse DCT.
<i>pDst</i>	Pointer to the destination predicted block.
<i>dstStep</i>	Size of the row in bytes, specifying the aligned destination frame width.
<i>mcType</i>	MC type <a href="#">IPPVC_MC_APX</a> .
<i>roundControl</i>	Parameter that determines type of rounding for half a pel approximation; may be 0 or 1.

### Description

This function is declared in the `ippvc.h` header file. The function `ippiMC8x16_8u_C1` calculates a sum of an 8x16 residual block and an 8x16 predicted block for reconstruction of the source block (See [Figure Motion Compensation Scheme for Predicted Block](#)). Prediction is calculated on the basis of the reference block and *mcType* (see [IPPVC\\_MC\\_APX](#)).

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when at least one input pointer is NULL.

## MC8x8

*Performs motion compensation for predicted 8X8 block.*

### Syntax

```
IppStatus ippiMC8x8_8u_C1(const Ipp8u* pSrcRef, Ipp32s srcStep, const Ipp16s* pSrcYData,
Ipp32s srcYDataStep, Ipp8u* pDst, Ipp32s dstStep, Ipp32s mcType, Ipp32s roundControl);

IppStatus ippiMC8x8_16s8u_P2C2R(const Ipp8u* pSrcRef, Ipp32s srcRefStep, const Ipp16s*
pSrcU, const Ipp16s* pSrcV, Ipp32s srcUVStep, Ipp8u* pDst, Ipp32s dstStep, Ipp32s mcType,
Ipp32s roundControl);
```

### Parameters

<i>pSrcRef</i>	Pointer to the reference intra block.
----------------	---------------------------------------

For `ippiMC8x8_16s8u_P2C2R`, pointer to the chrominance part of the NV12 plane:

```
0  UV UV UV UV UV UV UV UV
1  UV UV UV UV UV UV UV UV
...
7  UV UV UV UV UV UV UV UV
```

<code>srcStep, srcRefStep</code>	Size of the row in bytes, specifying the aligned reference frame width. A negative value is acceptable.
<code>pSrcYData</code>	Pointer to the data obtained after inverse DCT
<code>srcYDataStep</code>	Number of bytes, specifying the width of the aligned data obtained after inverse DCT.
<code>pSrcU</code>	Pointer to the buffer that contains <code>U</code> coefficients obtained after inverse DCT
<code>pSrcV</code>	Pointer to the buffer that contains <code>V</code> coefficients obtained after inverse DCT
<code>srcUVStep</code>	Number of bytes, specifying the width of the source block. A negative value is acceptable.
<code>pDst</code>	Pointer to the destination predicted block. For <code>ip- piMC8x8_16s8u_P2C2R</code> , pointer to the chrominance part of the NV12 plane.
<code>dstStep</code>	Size of the row in bytes, specifying the aligned destination frame width. A negative value is acceptable.
<code>mcType</code>	MC type <a href="#">IPPVC_MC_APX</a>
<code>roundControl</code>	Parameter that determines type of rounding for half-pixel approximation; may be 0 or 1

## Description

This function is declared in the `ippvc.h` header file. The function `ippiMC8x8_8u_C1` calculates a sum of an 8x8 residual block and an 8x8 predicted block for reconstruction of the source block (See [Figure Motion Compensation Scheme for Predicted Block](#)). Prediction is calculated on the basis of the reference block and `mcType` (see [IPPVC\\_MC\\_APX](#)).

`ippiMC8x8_16s8u_P2C2R` performs motion compensation for an NV12 chrominance plane:

NV12 Plane:

```
YY YY YY YY
YY YY YY YY
UV UV UV UV
```

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when at least one input pointer is <code>NULL</code> .

## MC8x4

*Performs motion compensation for predicted 8X4 block.*

### Syntax

```

IppStatus ippiMC8x4_8u_C1(const Ipp8u* pSrcRef, Ipp32s srcStep, const Ipp16s* pSrcYData,
Ipp32s srcYDataStep, Ipp8u* pDst, Ipp32s dstStep, Ipp32s mcType, Ipp32s roundControl);

IppStatus ippiMC8x4_16s8u_P2C2R(const Ipp8u* pSrcRef, Ipp32s srcRefStep, const Ipp16s*
pSrcU, const Ipp16s* pSrcV, Ipp32s srcUVStep, Ipp8u* pDst, Ipp32s dstStep, Ipp32s mcType,
Ipp32s roundControl);

```

### Parameters

<i>pSrcRef</i>	<p>Pointer to the reference intra block.</p> <p>For <code>ippiMC8x4_16s8u_P2C2R</code>, pointer to the chrominance part of the NV12 plane:</p> <pre> 0  UV UV UV UV UV UV UV UV 1  UV UV UV UV UV UV UV UV ... 7  UV UV UV UV UV UV UV UV </pre>
<i>srcStep, srcRefStep</i>	<p>Size of the row in bytes, specifying the aligned reference frame width. A negative value is acceptable.</p>
<i>pSrcYData</i>	<p>Pointer to the data obtained after inverse DCT</p>
<i>srcYDataStep</i>	<p>Number of bytes, specifying the width of the aligned data obtained after inverse DCT.</p>
<i>pSrcU</i>	<p>Pointer to the buffer that contains <i>U</i> coefficients obtained after inverse DCT</p>
<i>pSrcV</i>	<p>Pointer to the buffer that contains <i>V</i> coefficients obtained after inverse DCT</p>
<i>srcUVStep</i>	<p>Number of bytes, specifying the width of the source block. A negative value is acceptable.</p>
<i>pDst</i>	<p>Pointer to the destination predicted block. For <code>ippiMC8x4_16s8u_P2C2R</code>, pointer to the chrominance part of the NV12 plane.</p>
<i>dstStep</i>	<p>Size of the row in bytes, specifying the aligned destination frame width. A negative value is acceptable.</p>
<i>mcType</i>	<p>MC type <a href="#">IPPVC_MC_APX</a></p>
<i>roundControl</i>	<p>Parameter that determines type of rounding for half-pixel approximation; may be 0 or 1</p>

### Description

This function is declared in the `ippvc.h` header file. The function `ippiMC8x4_8u_C1` calculates a sum of an 8x4 residual block and an 8x4 predicted block for reconstruction of the source block (See [Figure Motion Compensation Scheme for Predicted Block](#)). Prediction is calculated on the basis of the reference block and *mcType* (see [IPPVC\\_MC\\_APX](#)).

`ippiMC8x4_16s8u_P2C2R` performs motion compensation for an NV12 chrominance plane:

NV12 Plane:

```
YY YY YY YY
YY YY YY YY
UV UV UV UV
```

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when at least one input pointer is <code>NULL</code> .

## MC4x8

*Performs motion compensation for predicted 4X8 block.*

## Syntax

```
IppStatus ippiMC4x8_8u_C1(const Ipp8u* pSrcRef, Ipp32s srcStep, const Ipp16s* pSrcYData,
Ipp32s srcYDataStep, Ipp8u* pDst, Ipp32s dstStep, Ipp32s mcType, Ipp32s roundControl);
```

## Parameters

<code>pSrcRef</code>	Pointer to the reference intra block.
<code>srcStep</code>	Size of the row in bytes, specifying the aligned reference frame width.
<code>pSrcYData</code>	Pointer to the data obtained after inverse DCT.
<code>srcYDataStep</code>	Number of bytes, specifying the width of the aligned data obtained after inverse DCT.
<code>pDst</code>	Pointer to the destination predicted block.
<code>dstStep</code>	Size of the row in bytes, specifying the aligned destination frame width.
<code>mcType</code>	MC type <a href="#">IPPVC_MC_APX</a> .
<code>roundControl</code>	Parameter that determines type of rounding for half a pel approximation; may be 0 or 1.

## Description

This function is declared in the `ippvc.h` header file. The function `ippiMC4x8_8u_C1` calculates a sum of a 4x8 residual block and a 4x8 predicted block for reconstruction of the source block (See [Figure Motion Compensation Scheme for Predicted Block](#)). Prediction is calculated on the basis of the reference block and `mcType` (see [IPPVC\\_MC\\_APX](#)).

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when at least one input pointer is <code>NULL</code> .

## MC4x4

*Performs motion compensation for predicted 4X4 block.*

### Syntax

```
IppStatus ippiMC4x4_8u_C1(const Ipp8u* pSrcRef, Ipp32s srcStep, const Ipp16s* pSrcYData,
Ipp32s srcYDataStep, Ipp8u* pDst, Ipp32s dstStep, Ipp32s mcType, Ipp32s roundControl);
```

### Parameters

<i>pSrcRef</i>	Pointer to the reference intra block.
<i>srcStep</i>	Size of the row in bytes, specifying the aligned reference frame width.
<i>pSrcYData</i>	Pointer to the data obtained after inverse DCT.
<i>srcYDataStep</i>	Number of bytes, specifying the width of the aligned data obtained after inverse DCT.
<i>pDst</i>	Pointer to the destination predicted block.
<i>dstStep</i>	Size of the row in bytes, specifying the aligned destination frame width.
<i>mcType</i>	MC type <a href="#">IPPVC_MC_APX</a> .
<i>roundControl</i>	Parameter that determines type of rounding for half a pel approximation; may be 0 or 1.

### Description

This function is declared in the `ippvc.h` header file. The function `ippiMC4x4_8u_C1` calculates a sum of a 4x4 residual block and a 4x4 predicted block for reconstruction of the source block (See [Figure Motion Compensation Scheme for Predicted Block](#)). Prediction is calculated on the basis of the reference block and *mcType* (see [IPPVC\\_MC\\_APX](#)).

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when at least one input pointer is NULL.

## MC2x4

*Performs motion compensation for predicted 2X4 block.*

### Syntax

```
IppStatus ippiMC2x4_8u_C1(const Ipp8u* pSrcRef, Ipp32s srcStep, const Ipp16s* pSrcYData,
Ipp32s srcYDataStep, Ipp8u* pDst, Ipp32s dstStep, Ipp32s mcType, Ipp32s roundControl);
```

### Parameters

<i>pSrcRef</i>	Pointer to the reference intra block.
<i>srcStep</i>	Size of the row in bytes, specifying the aligned reference frame width.
<i>pSrcYData</i>	Pointer to the data obtained after inverse DCT.

<i>srcYDataStep</i>	Number of bytes, specifying the width of the aligned data obtained after inverse DCT.
<i>pDst</i>	Pointer to the destination predicted block.
<i>dstStep</i>	Size of the row in bytes, specifying the aligned destination frame width.
<i>mcType</i>	MC type <a href="#">IPPVC_MC_APX</a> .
<i>roundControl</i>	Parameter that determines type of rounding for half a pel approximation; may be 0 or 1.

## Description

This function is declared in the `ippvc.h` header file. The function `ippiMC2x4_8u_C1` calculates a sum of a 2x4 residual block and a 2x4 predicted block for reconstruction of the source block (See [Figure Motion Compensation Scheme for Predicted Block](#)). Prediction is calculated on the basis of the reference block and *mcType* (see [IPPVC\\_MC\\_APX](#)).

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when at least one input pointer is NULL.

## MC4x2

Performs motion compensation for predicted 4X2 block.

## Syntax

```
ippStatus ippiMC4x2_8u_C1(const Ipp8u* pSrcRef, Ipp32s srcStep, const Ipp16s* pSrcYData,
Ipp32s srcYDataStep, Ipp8u* pDst, Ipp32s dstStep, Ipp32s mcType, Ipp32s roundControl);
```

## Parameters

<i>pSrcRef</i>	Pointer to the reference intra block.
<i>srcStep</i>	Size of the row in bytes, specifying the aligned reference frame width.
<i>pSrcYData</i>	Pointer to the data obtained after inverse DCT.
<i>srcYDataStep</i>	Number of bytes, specifying the width of the aligned data obtained after inverse DCT.
<i>pDst</i>	Pointer to the destination predicted block.
<i>dstStep</i>	Size of the row in bytes, specifying the aligned destination frame width.
<i>mcType</i>	MC type <a href="#">IPPVC_MC_APX</a> .
<i>roundControl</i>	Parameter that determines type of rounding for half a pel approximation; may be 0 or 1.

## Description

This function is declared in the `ippvc.h` header file. The function `ippiMC4x2_8u_C1` calculates a sum of a 4x2 residual block and a 4x2 predicted block for reconstruction of the source block (See [Figure Motion Compensation Scheme for Predicted Block](#)). Prediction is calculated on the basis of the reference block and *mcType* (see [IPPVC\\_MC\\_APX](#)).

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when at least one input pointer is NULL.

## MC2x2

*Performs motion compensation for predicted 2X2 block.*

### Syntax

```
ippStatus ippIMC2x2_8u_C1(const Ipp8u* pSrcRef, Ipp32s srcStep, const Ipp16s* pSrcYData,
Ipp32s srcYDataStep, Ipp8u* pDst, Ipp32s dstStep, Ipp32s mcType, Ipp32s roundControl);
```

### Parameters

<code>pSrcRef</code>	Pointer to the reference intra block.
<code>srcStep</code>	Size of the row in bytes, specifying the aligned reference frame width.
<code>pSrcYData</code>	Pointer to the data obtained after inverse DCT.
<code>srcYDataStep</code>	Number of bytes, specifying the width of the aligned data obtained after inverse DCT.
<code>pDst</code>	Pointer to the destination predicted block.
<code>dstStep</code>	Size of the row in bytes, specifying the aligned destination frame width.
<code>mcType</code>	MC type <a href="#">IPPVC_MC_APX</a> .
<code>roundControl</code>	Parameter that determines type of rounding for half a pel approximation; may be 0 or 1.

### Description

This function is declared in the `ippvc.h` header file. The function `ippIMC2x2_8u_C1` calculates a sum of a 2x2 residual block and a 2x2 predicted block for reconstruction of the source block (See [Figure Motion Compensation Scheme for Predicted Block](#)). Prediction is calculated on the basis of the reference block and `mcType` (see [IPPVC\\_MC\\_APX](#)).

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when at least one input pointer is NULL.

## MC16x4

*Performs motion compensation for predicted UV 16X4 block.*

### Syntax

```
ippStatus ippIMC16x4_8u_C1(const Ipp8u* pSrcRef, Ipp32s srcStep, const Ipp16s* pSrcYData,
Ipp32s srcYDataStep, Ipp8u* pDst, Ipp32s dstStep, Ipp32s mcType, Ipp32s roundControl);
```

## Parameters

<i>pSrcRef</i>	Pointer to the reference intra block.
<i>srcStep</i>	Size of the row in bytes, specifying the aligned reference frame width.
<i>pSrcYData</i>	Pointer to the data obtained after inverse DCT.
<i>srcYDataStep</i>	Number of bytes, specifying the width of the aligned data obtained after inverse DCT.
<i>pDst</i>	Pointer to the destination predicted block.
<i>dstStep</i>	Size of the row in bytes, specifying the aligned destination frame width.
<i>mcType</i>	MC type <a href="#">IPPVC_MC_APX</a> .
<i>roundControl</i>	Parameter that determines type of rounding for half a pel approximation; may be 0 or 1.

## Description

This function is declared in the `ippvc.h` header file. The function `ippiMC16x4_8u_C1` calculates a sum of a 16x4 residual UV block and a 16x4 predicted UV block for reconstruction of the source UV block (See [Figure Motion Compensation Scheme for Predicted Block](#)). Prediction is calculated on the basis of the reference UV block and *mcType* (see [IPPVC\\_MC\\_APX](#)).

Half sample prediction is calculated taking into account structure of [UV block](#) . So (horizontally) neighboring elements of U-block (or V-block) have indexes *i* and *i*+2 in UV block.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when at least one input pointer is NULL.

## MC16x8UV

*Performs motion compensation for predicted UV 16X8 block.*

---

## Syntax

```
ippStatus ippiMC16x8UV_8u_C1(const Ipp8u* pSrcRef, Ipp32s srcStep, const Ipp16s* pSrcYData,
Ipp32s srcYDataStep, Ipp8u* pDst, Ipp32s dstStep, Ipp32s mcType, Ipp32s roundControl);
```

## Parameters

<i>pSrcRef</i>	Pointer to the reference intra block.
<i>srcStep</i>	Size of the row in bytes, specifying the aligned reference frame width.
<i>pSrcYData</i>	Pointer to the data obtained after inverse DCT.
<i>srcYDataStep</i>	Number of bytes, specifying the width of the aligned data obtained after inverse DCT.
<i>pDst</i>	Pointer to the destination predicted block.
<i>dstStep</i>	Size of the row in bytes, specifying the aligned destination frame width.
<i>mcType</i>	MC type <a href="#">IPPVC_MC_APX</a> .



*roundControl* Parameter that determines type of rounding for half a pel approximation; may be 0 or 1.

## Description

This function is declared in the `ippvc.h` header file. The function `ippiMC16x8UV_8u_C1` calculates sum of 16x8 residual UV block and 16x8 predicted UV block for reconstruction of the source UV block (See [Figure Motion Compensation Scheme for Predicted Block](#)). Prediction is calculated on the basis of the reference UV block and *mcType* (see `IPPVC_MC_APX`).

Half sample prediction is calculated taking into account structure of [UV block](#). So (horizontally) neighboring elements of U-block (or V-block) have indexes  $i$  and  $i+2$  in UV block.

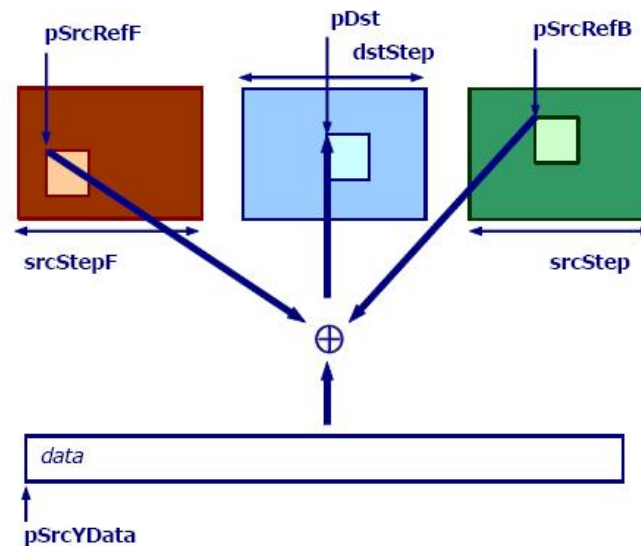
## Return Values

`ippStsNoErr` Indicates no error.  
`ippStsNullPtrErr` Indicates an error when at least one input pointer is NULL.

## Bi-Predicted Blocks

These functions use two reference blocks for prediction. Predictions are calculated for each reference block, and then prediction is calculated as average of two predictions. The prediction calculated as a rounded average of two blocks is the first to be added to the data.

### Motion Compensation Scheme for Bi-Predicted Block



## MC16x16B

*Performs motion compensation for bi-predicted block.*

---

### Syntax

```
IppStatus ippiMC16x16B_8u_C1(const Ipp8u* pSrcRefF, Ipp32s srcStepF, Ipp32s mcTypeF, const
Ipp8u* pSrcRefB, Ipp32s srcStepB, Ipp32s mcTypeB, const Ipp16s* pSrcYData, Ipp32s
srcYDataStep, Ipp8u* pDst, Ipp32s dstStep, Ipp32s roundControl);
```

### Parameters

<i>pSrcRefF</i>	Pointer to the forward reference block.
<i>srcStepF</i>	Size of the row in bytes, specifying the aligned forward reference frame width.
<i>mcTypeF</i>	Forward MC type <a href="#">IPPVC_MC_APX</a> .
<i>pSrcRefB</i>	Pointer to the backward reference block.
<i>srcStepB</i>	Size of the row in bytes, specifying the aligned backward reference frame width.
<i>mcTypeB</i>	Backward MC type <a href="#">IPPVC_MC_APX</a> .
<i>pSrcYData</i>	Pointer to the data obtained after inverse DCT.
<i>srcYDataStep</i>	Number of bytes, specifying the width of the aligned data obtained after inverse DCT.
<i>pDst</i>	Pointer to the destination predicted block.
<i>dstStep</i>	Size of the row in bytes, specifying the aligned destination frame width.
<i>roundControl</i>	Parameter that determines type of rounding for half a pel approximation; may be 0 or 1.

### Description

This function is declared in the `ippvc.h` header file. The function `ippiMC16x16B_8u_C1` calculates sum of 16x16 residual block and 16x16 predicted block, which is calculated as average of two 16x16 predictions (F-prediction and B-prediction) (See [Figure Motion Compensation Scheme for Bi-Predicted Block](#)). Each prediction is calculated on the basis of the corresponding reference block and corresponding *mcType* (see [IPPVC\\_MC\\_APX](#)).

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when at least one input pointer is <code>NULL</code> .

## MC16x8B

*Performs motion compensation for bi-predicted 16X8 block.*

---

### Syntax

```
IppStatus ippIMC16x8B_8u_C1(const Ipp8u* pSrcRefF, Ipp32s srcStepF, Ipp32s mcTypeF, const
Ipp8u* pSrcRefB, Ipp32s srcStepB, Ipp32s mcTypeB, const Ipp16s* pSrcYData, Ipp32s
srcYDataStep, Ipp8u* pDst, Ipp32s dstStep, Ipp32s roundControl);
```

### Parameters

<i>pSrcRefF</i>	Pointer to the forward reference block.
<i>srcStepF</i>	Size of the row in bytes, specifying the aligned forward reference frame width.
<i>mcTypeF</i>	Forward MC type <a href="#">IPPVC_MC_APX</a> .
<i>pSrcRefB</i>	Pointer to the backward reference block.
<i>srcStepB</i>	Size of the row in bytes, specifying the aligned backward reference frame width.
<i>mcTypeB</i>	Backward MC type <a href="#">IPPVC_MC_APX</a> .
<i>pSrcYData</i>	Pointer to the data obtained after inverse DCT.
<i>srcYDataStep</i>	Number of bytes, specifying the width of the aligned data obtained after inverse DCT.
<i>pDst</i>	Pointer to the destination predicted block.
<i>dstStep</i>	Size of the row in bytes, specifying the aligned destination frame width.
<i>roundControl</i>	Parameter that determines type of rounding for half a pel approximation; may be 0 or 1.

### Description

This function is declared in the `ippvc.h` header file. The function `ippIMC16x8B_8u_C1` calculates sum of 16x8 residual block and 16x8 predicted block, which is calculated as average of two 16x8 predictions (F-prediction and B-prediction) (See [Figure Motion Compensation Scheme for Bi-Predicted Block](#)). Each prediction is calculated on the basis of the corresponding reference block and corresponding *mcType* (see [IPPVC\\_MC\\_APX](#)).

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when at least one input pointer is <code>NULL</code> .

## MC8x16B

*Performs motion compensation for bi-predicted 8X16 block.*

---

### Syntax

```
IppStatus ippMC8x16B_8u_C1(const Ipp8u* pSrcRefF, Ipp32s srcStepF, Ipp32s mcTypeF, const
Ipp8u* pSrcRefB, Ipp32s srcStepB, Ipp32s mcTypeB, const Ipp16s* pSrcYData, Ipp32s
srcYDataStep, Ipp8u* pDst, Ipp32s dstStep, Ipp32s roundControl);
```

### Parameters

<i>pSrcRefF</i>	Pointer to the forward reference block.
<i>srcStepF</i>	Size of the row in bytes, specifying the aligned forward reference frame width.
<i>mcTypeF</i>	Forward MC type <a href="#">IPPVC_MC_APX</a> .
<i>pSrcRefB</i>	Pointer to the backward reference block.
<i>srcStepB</i>	Size of the row in bytes, specifying the aligned backward reference frame width.
<i>mcTypeB</i>	Backward MC type <a href="#">IPPVC_MC_APX</a> .
<i>pSrcYData</i>	Pointer to the data obtained after inverse DCT.
<i>srcYDataStep</i>	Number of bytes, specifying the width of the aligned data obtained after inverse DCT.
<i>pDst</i>	Pointer to the destination predicted block.
<i>dstStep</i>	Size of the row in bytes, specifying the aligned destination frame width.
<i>roundControl</i>	Parameter that determines type of rounding for half a pel approximation; may be 0 or 1.

### Description

This function is declared in the `ippvc.h` header file. The function `ippMC8x16B_8u_C1` calculates sum of 8x16 residual block and 8x16 predicted block, which is calculated as average of two 8x16 predictions (F-prediction and B-prediction) (See [Figure Motion Compensation Scheme for Bi-Predicted Block](#)). Each prediction is calculated on the basis of the corresponding reference block and corresponding *mcType* (see [IPPVC\\_MC\\_APX](#)).

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when at least one input pointer is <code>NULL</code> .

## MC8x8B

*Performs motion compensation for bi-predicted 8X8 block.*

### Syntax

```
IppStatus ippiMC8x8B_8u_C1(const Ipp8u* pSrcRefF, Ipp32s srcStepF, Ipp32s mcTypeF, const
Ipp8u* pSrcRefB, Ipp32s srcStepB, Ipp32s mcTypeB, const Ipp16s* pSrcYData, Ipp32s
srcYDataStep, Ipp8u* pDst, Ipp32s dstStep, Ipp32s roundControl);
```

```
IppStatus ippiMC8x8B_16s8u_P2C2R(const Ipp8u* pSrcRefF, Ipp32s srcRefFStep, Ipp32s mcTypeF,
const Ipp8u* pSrcRefB, Ipp32s srcRefBStep, Ipp32s mcTypeB, const Ipp16s* pSrcU, const
Ipp16s* pSrcV, Ipp32s srcUVStep, Ipp8u* pDst, Ipp32s dstStep, Ipp32s roundControl);
```

### Parameters

<i>pSrcRefF</i>	Pointer to the forward reference block. For <code>ippiMC8x8B_16s8u_P2C2R</code> , pointer to the chrominance part of the NV12 plane: <pre> 0  UV UV UV UV UV UV UV UV 1  UV UV UV UV UV UV UV UV ... 7  UV UV UV UV UV UV UV UV </pre>
<i>srcStepF, srcRefFStep</i>	Size of the row in bytes, specifying the aligned forward reference frame width. A negative value is acceptable.
<i>mcTypeF</i>	Forward MC type <a href="#">IPPVC_MC_APX</a>
<i>pSrcRefB</i>	Pointer to the backward reference block. For <code>ippiMC8x8B_16s8u_P2C2R</code> , pointer to the chrominance part of the NV12 plane.
<i>srcStepB, srcRefBStep</i>	Size of the row in bytes, specifying the aligned backward reference frame width
<i>mcTypeB</i>	Backward MC type <a href="#">IPPVC_MC_APX</a>
<i>pSrcYData</i>	Pointer to the data obtained after inverse DCT.
<i>pSrcU</i>	Pointer to the buffer that contains U coefficients obtained after inverse DCT
<i>pSrcV</i>	Pointer to the buffer that contains V coefficients obtained after inverse DCT
<i>srcYDataStep</i>	Number of bytes, specifying the width of the aligned data obtained after inverse DCT
<i>srcUVStep</i>	Number of bytes, specifying the width of the source block. A negative value is acceptable.
<i>pDst</i>	Pointer to the destination predicted block. For <code>ippiMC8x8B_16s8u_P2C2R</code> , pointer to the chrominance part of the NV12 plane.
<i>dstStep</i>	Size of the row in bytes, specifying the aligned destination frame width. A negative value is acceptable.

*roundControl* Parameter that determines type of rounding for half-pixel approximation; may be 0 or 1

## Description

This function is declared in the `ippvc.h` header file. The function `ippiMC8x8B_8u_C1` calculates sum of 8x8 residual block and 8x8 predicted block, which is calculated as average of two 8x8 predictions (F-prediction and B-prediction) (See [Figure Motion Compensation Scheme for Bi-Predicted Block](#)). Each prediction is calculated on the basis of the corresponding reference block and corresponding *mcType* (see [IPPVC\\_MC\\_APX](#)).

`ippiMC8x8B_16s8u_P2C2R` performs motion compensation for an NV12 chrominance plane:

NV12 Plane:

```
YY YY YY YY
YY YY YY YY
UV UV UV UV
```

## Return Values

`ippStsNoErr` Indicates no error.

`ippStsNullPtrErr` Indicates an error when at least one input pointer is NULL.

## MC8x4B

Performs motion compensation for bi-predicted 8x4 block.

## Syntax

```
IppStatus ippiMC8x4B_8u_C1(const Ipp8u* pSrcRefF, Ipp32s srcStepF, Ipp32s mcTypeF, const
Ipp8u* pSrcRefB, Ipp32s srcStepB, Ipp32s mcTypeB, const Ipp16s* pSrcYData, Ipp32s
srcYDataStep, Ipp8u* pDst, Ipp32s dstStep, Ipp32s roundControl);
```

```
IppStatus ippiMC8x4B_16s8u_P2C2R(const Ipp8u* pSrcRefF, Ipp32s srcRefFStep, Ipp32s mcTypeF,
const Ipp8u* pSrcRefB, Ipp32s srcRefBStep, Ipp32s mcTypeB, const Ipp16s* pSrcU, const
Ipp16s* pSrcV, Ipp32s srcUVStep, Ipp8u* pDst, Ipp32s dstStep, Ipp32s roundControl);
```

## Parameters

*pSrcRefF* Pointer to the forward reference block.  
For `ippiMC8x8B_16s8u_P2C2R`, pointer to the chrominance part of the NV12 plane:

```
0  UV UV UV UV UV UV UV UV
1  UV UV UV UV UV UV UV UV
...
7  UV UV UV UV UV UV UV UV
```

*srcStepF, srcRefFStep* Size of the row in bytes, specifying the aligned forward reference frame width. A negative value is acceptable.

*mcTypeF* Forward MC type [IPPVC\\_MC\\_APX](#)

*pSrcRefB* Pointer to the backward reference block. For `ippiMC8x4B_16s8u_P2C2R`, pointer to the chrominance part of the NV12 plane.

<i>srcStepB</i> , <i>srcRefBStep</i>	Size of the row in bytes, specifying the aligned backward reference frame width
<i>mcTypeB</i>	Backward MC type <a href="#">IPPVC_MC_APX</a>
<i>pSrcYData</i>	Pointer to the data obtained after inverse DCT.
<i>pSrcU</i>	Pointer to the buffer that contains U coefficients obtained after inverse DCT
<i>pSrcV</i>	Pointer to the buffer that contains V coefficients obtained after inverse DCT
<i>srcYDataStep</i>	Number of bytes, specifying the width of the aligned data obtained after inverse DCT
<i>srcUVStep</i>	Number of bytes, specifying the width of the source block. A negative value is acceptable.
<i>pDst</i>	Pointer to the destination predicted block. For <i>ip- piMC8x4B_16s8u_P2C2R</i> , pointer to the chrominance part of the NV12 plane.
<i>dstStep</i>	Size of the row in bytes, specifying the aligned destination frame width. A negative value is acceptable.
<i>roundControl</i>	Parameter that determines type of rounding for half-pixel approximation; may be 0 or 1

## Description

This function is declared in the `ippvc.h` header file. The function `ippiMC8x4B_8u_C1` calculates sum of 8x4 residual block and 8x4 predicted block, which is calculated as average of two 8x4 predictions (F-prediction and B-prediction) (See [Figure Motion Compensation Scheme for Bi-Predicted Block](#)). Each prediction is calculated on the basis of the corresponding reference block and corresponding *mcType* (see [IPPVC\\_MC\\_APX](#)).

`ippiMC8x4B_16s8u_P2C2R` performs motion compensation for an NV12 chrominance plane:

NV12 Plane:

```
YY YY YY YY
YY YY YY YY
UV UV UV UV
```

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when at least one input pointer is <code>NULL</code> .

## MC4x8B

*Performs motion compensation for bi-predicted 4X8 block.*

---

## Syntax

```
IppStatus ippiMC4x8B_8u_C1(const Ipp8u* pSrcRefF, Ipp32s srcStepF, Ipp32s mcTypeF, const
Ipp8u* pSrcRefB, Ipp32s srcStepB, Ipp32s mcTypeB, const Ipp16s* pSrcYData, Ipp32s
srcYDataStep, Ipp8u* pDst, Ipp32s dstStep, Ipp32s roundControl);
```

## Parameters

<i>pSrcRefF</i>	Pointer to the forward reference block.
<i>srcStepF</i>	Size of the row in bytes, specifying the aligned forward reference frame width.
<i>mcTypeF</i>	Forward MC type <a href="#">IPPVC_MC_APX</a> .
<i>pSrcRefB</i>	Pointer to the backward reference block.
<i>srcStepB</i>	Size of row in bytes, specifying the aligned backward reference frame width.
<i>mcTypeB</i>	Backward MC type <a href="#">IPPVC_MC_APX</a> .
<i>pSrcYData</i>	Pointer to the data obtained after inverse DCT.
<i>srcYDataStep</i>	Number of bytes, specifying the width of the aligned data obtained after inverse DCT.
<i>pDst</i>	Pointer to the destination predicted block.
<i>dstStep</i>	Size of the row in bytes, specifying the aligned destination frame width.
<i>roundControl</i>	Parameter that determines type of rounding for half a pel approximation; may be 0 or 1.

## Description

This function is declared in the `ippvc.h` header file. The function `ippiMC4x8B_8u_C1` calculates sum of 4x8 residual block and 4x8 predicted block, which is calculated as average of two 4x8 predictions (F-prediction and B-prediction) (See [Figure Motion Compensation Scheme for Bi-Predicted Block](#)). Each prediction is calculated on the basis of the corresponding reference block and corresponding *mcType* (see [IPPVC\\_MC\\_APX](#)).

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when at least one input pointer is NULL.

## MC4x4B

*Performs motion compensation for bi-predicted 4X4 block.*

---

## Syntax

```
ippStatus ippiMC4x4B_8u_C1(const Ipp8u* pSrcRefF, Ipp32s srcStepF, Ipp32s mcTypeF, const
Ipp8u* pSrcRefB, Ipp32s srcStepB, Ipp32s mcTypeB, const Ipp16s* pSrcYData, Ipp32s
srcYDataStep, Ipp8u* pDst, Ipp32s dstStep, Ipp32s roundControl);
```

## Parameters

<i>pSrcRefF</i>	Pointer to the forward reference block.
<i>srcStepF</i>	Size of the row in bytes, specifying the aligned forward reference frame width.
<i>mcTypeF</i>	Forward MC type <a href="#">IPPVC_MC_APX</a> .



<i>pSrcRefB</i>	Pointer to the backward reference block.
<i>srcStepB</i>	Size of row in bytes, specifying the aligned backward reference frame width.
<i>mcTypeB</i>	Backward MC type <a href="#">IPPVC_MC_APX</a> .
<i>pSrcYData</i>	Pointer to the data obtained after inverse DCT.
<i>srcYDataStep</i>	Number of bytes, specifying the width of the aligned data obtained after inverse DCT.
<i>pDst</i>	Pointer to the destination predicted block.
<i>dstStep</i>	Size of the row in bytes, specifying the aligned destination frame width.
<i>roundControl</i>	Parameter that determines type of rounding for half a pel approximation; may be 0 or 1.

## Description

This function is declared in the `ippvc.h` header file. The function `ippiMC4x4B_8u_C1` calculates sum of a 4x4 residual block and a 4x4 predicted block, which is calculated as average of two 4x4 predictions (F-prediction and B-prediction) (See [Figure Motion Compensation Scheme for Bi-Predicted Block](#)). Each prediction is calculated on the basis of the corresponding reference block and corresponding *mcType* (see [IPPVC\\_MC\\_APX](#)).

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when at least one input pointer is NULL.

## MC2x4B

*Performs motion compensation for bi-predicted 2X4 block.*

## Syntax

```
ippStatus ippiMC2x4B_8u_C1(const Ipp8u* pSrcRefF, Ipp32s srcStepF, Ipp32s mcTypeF, const Ipp8u* pSrcRefB, Ipp32s srcStepB, Ipp32s mcTypeB, const Ipp16s* pSrcYData, Ipp32s srcYDataStep, Ipp8u* pDst, Ipp32s dstStep, Ipp32s roundControl);
```

## Parameters

<i>pSrcRefF</i>	Pointer to the forward reference block.
<i>srcStepF</i>	Size of the row in bytes, specifying the aligned forward reference frame width.
<i>mcTypeF</i>	Forward MC type <a href="#">IPPVC_MC_APX</a> .
<i>pSrcRefB</i>	Pointer to the backward reference block.
<i>srcStepB</i>	Size of row in bytes, specifying the aligned backward reference frame width.
<i>mcTypeB</i>	Backward MC type <a href="#">IPPVC_MC_APX</a> .
<i>pSrcYData</i>	Pointer to the data obtained after inverse DCT.

<i>srcYDataStep</i>	Number of bytes, specifying the width of the aligned data obtained after inverse DCT.
<i>pDst</i>	Pointer to the destination predicted block.
<i>dstStep</i>	Size of the row in bytes, specifying the aligned destination frame width.
<i>roundControl</i>	Parameter that determines type of rounding for half a pel approximation; may be 0 or 1.

## Description

This function is declared in the `ippvc.h` header file. The function `ippiMC2x4B_8u_C1` calculates a sum of a 2x4 residual block and a 2x4 predicted block, which is calculated as average of two 2x4 predictions (F-prediction and B-prediction) (See [Figure Motion Compensation Scheme for Bi-Predicted Block](#)). Each prediction is calculated on the basis of the corresponding reference block and corresponding *mcType* (see [IPPVC\\_MC\\_APX](#)).

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when at least one input pointer is <code>NULL</code> .

## MC4x2B

*Performs motion compensation for bi-predicted 4X2 block.*

---

## Syntax

```
ippStatus ippiMC4x2B_8u_C1(const Ipp8u* pSrcRefF, Ipp32s srcStepF, Ipp32s mcTypeF, const
Ipp8u* pSrcRefB, Ipp32s srcStepB, Ipp32s mcTypeB, const Ipp16s* pSrcYData, Ipp32s
srcYDataStep, Ipp8u* pDst, Ipp32s dstStep, Ipp32s roundControl);
```

## Parameters

<i>pSrcRefF</i>	Pointer to the forward reference block.
<i>srcStepF</i>	Size of the row in bytes, specifying the aligned forward reference frame width.
<i>mcTypeF</i>	Forward MC type <a href="#">IPPVC_MC_APX</a> .
<i>pSrcRefB</i>	Pointer to the backward reference block.
<i>srcStepB</i>	Size of row in bytes, specifying the aligned backward reference frame width.
<i>mcTypeB</i>	Backward MC type <a href="#">IPPVC_MC_APX</a> .
<i>pSrcYData</i>	Pointer to the data obtained after inverse DCT.
<i>srcYDataStep</i>	Number of bytes, specifying the width of the aligned data obtained after inverse DCT.
<i>pDst</i>	Pointer to the destination predicted block.
<i>dstStep</i>	Size of the row in bytes, specifying the aligned destination frame width.
<i>roundControl</i>	Parameter that determines type of rounding for half a pel approximation; may be 0 or 1.

## Description

This function is declared in the `ippvc.h` header file. The function `ippiMC4x2B_8u_C1` calculates a sum of a 4x2 residual block and a 4x2 predicted block, which is calculated as average of two 4x2 predictions (F-prediction and B-prediction) (See [Figure Motion Compensation Scheme for Bi-Predicted Block](#)). Each prediction is calculated on the basis of the corresponding reference block and corresponding *mcType* (see [IPPVC\\_MC\\_APX](#)).

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when at least one input pointer is <code>NULL</code> .

## MC2x2B

*Performs motion compensation for bi-predicted 2X2 block.*

---

## Syntax

```
ippStatus ippiMC2x2B_8u_C1(const Ipp8u* pSrcRefF, Ipp32s srcStepF, Ipp32s mcTypeF, const
Ipp8u* pSrcRefB, Ipp32s srcStepB, Ipp32s mcTypeB, const Ipp16s* pSrcYData, Ipp32s
srcYDataStep, Ipp8u* pDst, Ipp32s dstStep, Ipp32s roundControl);
```

## Parameters

<i>pSrcRefF</i>	Pointer to the forward reference block.
<i>srcStepF</i>	Size of the row in bytes, specifying the aligned forward reference frame width.
<i>mcTypeF</i>	Forward MC type <a href="#">IPPVC_MC_APX</a> .
<i>pSrcRefB</i>	Pointer to the backward reference block.
<i>srcStepB</i>	Size of row in bytes, specifying the aligned backward reference frame width.
<i>mcTypeB</i>	Backward MC type <a href="#">IPPVC_MC_APX</a> .
<i>pSrcYData</i>	Pointer to the data obtained after inverse DCT.
<i>srcYDataStep</i>	Number of bytes, specifying the width of the aligned data obtained after inverse DCT.
<i>pDst</i>	Pointer to the destination predicted block.
<i>dstStep</i>	Size of the row in bytes, specifying the aligned destination frame width.
<i>roundControl</i>	Parameter that determines type of rounding for half a pel approximation; may be 0 or 1.

## Description

This function is declared in the `ippvc.h` header file. The function `ippiMC2x2B_8u_C1` calculates a sum of a 2x2 residual block and a 2x2 predicted block, which is calculated as average of two 2x2 predictions (F-prediction and B-prediction) (See [Figure Motion Compensation Scheme for Bi-Predicted Block](#)). Each prediction is calculated on the basis of the corresponding reference block and corresponding *mcType* (see [IPPVC\\_MC\\_APX](#)).

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when at least one input pointer is <code>NULL</code> .

## MC16x4B

*Performs motion compensation for bi-predicted UV 16X4 block.*

---

## Syntax

```
IppStatus ippIMC16x4B_8u_C1(const Ipp8u* pSrcRefF, Ipp32s srcStepF, Ipp32s mcTypeF, const
Ipp8u* pSrcRefB, Ipp32s srcStepB, Ipp32s mcTypeB, const Ipp16s* pSrcYData, Ipp32s
srcYDataStep, Ipp8u* pDst, Ipp32s dstStep, Ipp32s roundControl);
```

## Parameters

<code>pSrcRefF</code>	Pointer to the forward reference block.
<code>srcStepF</code>	Size of the row in bytes, specifying the aligned forward reference frame width.
<code>mcTypeF</code>	Forward MC type <a href="#">IPPVC_MC_APX</a> .
<code>pSrcRefB</code>	Pointer to the backward reference block.
<code>srcStepB</code>	Size of the row in bytes, specifying the aligned backward reference frame width.
<code>mcTypeB</code>	Backward MC type <a href="#">IPPVC_MC_APX</a> .
<code>pSrcYData</code>	Pointer to the data obtained after inverse DCT.
<code>srcYDataStep</code>	Number of bytes, specifying the width of the aligned data obtained after inverse DCT.
<code>pDst</code>	Pointer to the destination predicted block.
<code>dstStep</code>	Size of the row in bytes, specifying the aligned destination frame width.
<code>roundControl</code>	Parameter that determines type of rounding for half a pel approximation; may be 0 or 1.

## Description

This function is declared in the `ippvc.h` header file. The function `ippIMC16x4B_8u_C1` calculates a sum of a 16x4 residual UV block and a 16x4 predicted UV block, which is calculated as average of two 16x4 predictions (F-prediction and B-prediction) (See [Figure 16x8 UV Block \(8x8 U Block and 8x8 V Block\)](#)). Each prediction is calculated on the basis of the corresponding reference block and corresponding `mcType` (see [IPPVC\\_MC\\_APX](#)).

Half sample prediction is calculated taking into account structure of [UV block](#). So (horizontally) neighboring elements of U-block (or V-block) have indexes  $i$  and  $i+2$  in UV block.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when at least one input pointer is <code>NULL</code> .

## MC16x8UVB

*Performs motion compensation for bi-predicted 16X8 block.*

### Syntax

```
IppStatus ippIMC16x8BUV_8u_C1(const Ipp8u* pSrcRefF, Ipp32s srcStepF, Ipp32s mcTypeF,
const Ipp8u* pSrcRefB, Ipp32s srcStepB, Ipp32s mcTypeB, const Ipp16s* pSrcYData, Ipp32s
srcYDataStep, Ipp8u* pDst, Ipp32s dstStep, Ipp32s roundControl);
```

### Parameters

<i>pSrcRefF</i>	Pointer to the forward reference block.
<i>srcStepF</i>	Size of the row in bytes, specifying the aligned forward reference frame width.
<i>mcTypeF</i>	Forward MC type <a href="#">IPPVC_MC_APX</a> .
<i>pSrcRefB</i>	Pointer to the backward reference block.
<i>srcStepB</i>	Size of row in bytes, specifying the aligned backward reference frame width.
<i>mcTypeB</i>	Backward MC type <a href="#">IPPVC_MC_APX</a> .
<i>pSrcYData</i>	Pointer to the data obtained after inverse DCT.
<i>srcYDataStep</i>	Number of bytes, specifying the width of the aligned data obtained after inverse DCT.
<i>pDst</i>	Pointer to the destination predicted block.
<i>dstStep</i>	Size of the row in bytes, specifying the aligned destination frame width.
<i>roundControl</i>	Parameter that determines type of rounding for half a pel approximation; may be 0 or 1.

### Description

This function is declared in the `ippvc.h` header file. The function `ippIMC16x8BUV_8u_C1` calculates a sum of a 16x8 residual UV block and a 16x8 predicted UV block, which is calculated as average of two 16x8 predictions (F-prediction and B-prediction) (See [Figure 16x8 UV Block \(8x8 U Block and 8x8 V Block\)](#)). Each prediction is calculated on the basis of the corresponding reference block and corresponding *mcType* (see [IPPVC\\_MC\\_APX](#)).

Half sample prediction is calculated taking into account structure of [UV block](#). So (horizontally) neighboring elements of U-block (or V-block) have indexes *i* and *i*+2 in UV block.

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when at least one input pointer is NULL.

## Copy8x8, Copy16x16

*Copy fixed size block.*

---

### Syntax

```

IppStatus ippiCopy8x8_8u_C1R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst, int dstStep);
IppStatus ippiCopy16x16_8u_C1R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst, int dstStep);

```

### Parameters

<i>pSrc</i>	Pointer to the source block.
<i>srcStep</i>	Distance in bytes between starts of the consecutive lines in the source block.
<i>pDst</i>	Pointer to the destination block.
<i>dstStep</i>	Distance in bytes between starts of the consecutive lines in the destination block.

### Description

The functions `ippiCopy8x8_8u_C1R` and `ippiCopy16x16_8u_C1R` are declared in the `ippvc.h` file. These functions copy the source block to the destination block.

These functions are used in the H.261, H.263, and MPEG-4 encoders and decoders included into Intel IPP Samples. See introduction to this section.

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when at least one input pointer is <code>NULL</code> .

## Copy8x4HP, Copy8x8HP, Copy16x8HP, Copy16x16HP

*Copy fixed size blocks with half-pixel accuracy.*

---

### Syntax

```

IppStatus ippiCopy8x4HP_8u_C1R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst, int dstStep,
int acc, int rounding);
IppStatus ippiCopy8x8HP_8u_C1R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst, int dstStep,
int acc, int rounding);
IppStatus ippiCopy16x8HP_8u_C1R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst, int dstStep,
int acc, int rounding);
IppStatus ippiCopy16x16HP_8u_C1R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst, int dstStep,
int acc, int rounding);

```

### Parameters

<i>pSrc</i>	Pointer to the source block.
-------------	------------------------------

<i>srcStep</i>	Distance in bytes between starts of the consecutive lines in the source block.
<i>pDst</i>	Pointer to the destination block.
<i>dstStep</i>	Distance in bytes between starts of the consecutive lines in the destination block.
<i>acc</i>	Parameter that determines half-pixel accuracy.
<i>rounding</i>	Parameter that determines type of rounding for pixel interpolation; may be 0 or 1.

## Description

The functions `ippiCopy8x4HP_8u_C1R`, `ippiCopy8x8HP_8u_C1R`, `ippiCopy16x8HP_8u_C1R`, and `ippiCopy16x16HP_8u_C1R` are declared in the `ippvc.h` file. These functions copy the source block to the destination block with half-pixel accuracy. Parameter *rounding* has the same meaning as `RTYPE` in [ITUH263] and `vop_rounding_type` in [ISO14496]. Parameter *acc* is a two-bit value. Bit 0 defines half-pixel offset in horizontal direction and bit 1 defines half-pixel offset in vertical direction. The process of half-pixel interpolation is described in [ITUH263] subclause 6.1.2 and in [ISO14496] subclause 7.6.2.1.

The functions `ippiCopy8x4HP_8u_C1R` and `ippiCopy16x8HP_8u_C1R` are used in the MPEG-4 encoders and decoders included into Intel IPP Samples. The functions `ippiCopy8x8HP_8u_C1R` and `ippiCopy16x16HP_8u_C1R` are used in the H.263 and MPEG-4 encoders and decoders included into Intel IPP Samples. See [introduction](#) to this section.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when at least one input pointer is <code>NULL</code> .

## InterpolateAverage8x4, InterpolateAverage8x8, InterpolateAverage16x8, InterpolateAverage16x16

*Interpolate source block according to half-pixel offset and average the result with destination block.*

## Syntax

```
ippStatus ippiInterpolateAverage8x4_8u_C1IR(const Ipp8u* pSrc, int srcStep, Ipp8u* pSrcDst,
int srcDstStep, int acc, int rounding);
```

```
ippStatus ippiInterpolateAverage8x8_8u_C1IR(const Ipp8u* pSrc, int srcStep, Ipp8u* pSrcDst,
int srcDstStep, int acc, int rounding);
```

```
ippStatus ippiInterpolateAverage16x8_8u_C1IR(const Ipp8u* pSrc, int srcStep, Ipp8u* pSrcDst,
int srcDstStep, int acc, int rounding);
```

```
ippStatus ippiInterpolateAverage16x16_8u_C1IR(const Ipp8u* pSrc, int srcStep, Ipp8u*
pSrcDst, int srcDstStep, int acc, int rounding);
```

## Parameters

<i>pSrc</i>	Pointer to the source block.
-------------	------------------------------

<i>srcStep</i>	Distance in bytes between starts of the consecutive lines in the source block.
<i>pSrcDst</i>	Pointer to the destination block.
<i>srcDstStep</i>	Distance in bytes between starts of the consecutive lines in the destination block.
<i>acc</i>	Parameter that determines half-pixel offset.
<i>rounding</i>	Parameter that determines type of rounding for pixel interpolation; may be 0 or 1.

### Description

The functions `ippiInterpolateAverage8x4_8u_C1IR`, `ippiInterpolateAverage8x8_8u_C1IR`, `ippiInterpolateAverage16x8_8u_C1IR`, and `ippiInterpolateAverage16x16_8u_C1IR` are declared in the `ippvc.h` file. These functions interpolate the source block according to half-pixel offset and average the result with the destination block. Parameters *rounding* has the meaning as RTYPE in [ITUH263] and `vop_rounding_type` in [ISO14496]. Parameter *acc* is a two-bit value. Bit 0 defines half-pixel offset in horizontal direction and bit 1 defines half-pixel offset in vertical direction. These functions are used in MPEG-2 encoder and decoder included into Intel IPP Samples together with functions `Copy8x4HP`, `Copy8x8HP`, `Copy16x8HP`, `Copy16x16HP` for bidirectional motion compensation. See [introduction](#) to this section.

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when at least one input pointer is NULL.

## Add8x8

*Adds two blocks with saturation.*

---

### Syntax

```
ippStatus ippiAdd8x8_16s8u_C1IRS(const Ipp16s* pSrc, int srcStep, Ipp8u* pSrcDst, int srcDstStep);
```

### Parameters

<i>pSrc</i>	Pointer to the source block.
<i>srcStep</i>	Distance in bytes between starts of the consecutive lines in the source block.
<i>pSrcDst</i>	Pointer to the second source/destination block.
<i>srcDstStep</i>	Distance in bytes between starts of the consecutive lines in the destination block.

### Description

The function `ippiAdd8x8_16s8u_C1IRS` is declared in the `ippvc.h` file. This function adds 16s data from *pSrc* block to the 8u data from *pSrcDst* block with saturation and stores the result in *pSrcDst* block. It can be used in motion compensation process to add a reconstructed block of prediction errors to the predictor.



This function is used in the H.261, H.263, and MPEG-4 encoders and decoders included into Intel IPP Samples. See [introduction](#) to this section.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when at least one input pointer is <code>NULL</code> .

## Add8x8HP

*Adds blocks interpolated with half-pixel accuracy prediction to difference with saturation.*

## Syntax

```
ippStatus ippiAdd8x8HP_16s8u_C1RS(const Ipp16s* pSrc1, int src1Step, Ipp8u* pSrc2, int src2Step, Ipp8u* pDst, int dstStep, int acc, int rounding);
```

## Parameters

<code>pSrc1</code>	Pointer to the source block of differences.
<code>src1Step</code>	Distance in bytes between starts of the consecutive lines in the <code>pSrc1</code> plane.
<code>pSrc2</code>	Pointer to the source block of prediction.
<code>src2Step</code>	Distance in bytes between starts of the consecutive lines in the <code>pSrc2</code> plane.
<code>pDst</code>	Pointer to the second source/destination block.
<code>dstStep</code>	Distance in bytes between starts of the consecutive lines in the destination plane.
<code>acc</code>	Parameter that determines half-pixel accuracy.
<code>rounding</code>	Parameter that determines type of rounding for pixel interpolation; may be 0 or 1.

## Description

The function `ippiAdd8x8HP_16s8u_C1RS` is declared in the `ippvc.h` file. This function adds 16s data from `pSrc1` block to the data from `pSrc2` block interpolated with half-pixel accuracy with saturation. It can be used in motion compensation process to add a reconstructed block of prediction errors to the predictor. Parameter `rounding` has the same meaning as `RTYPE` in [\[ITUH263\]](#) and `vop_rounding_type` in [\[ISO14496\]](#). Parameter `acc` is a two-bit value. Bit 0 defines half-pixel offset in horizontal direction and bit 1 defines half-pixel offset in vertical direction. The process of half-pixel interpolation is described in [\[ITUH263\]](#) subclause 6.1.2 and in [\[ISO14496\]](#) subclause 7.6.2.1.

This function is used in the H.263 and MPEG-4 encoders and decoders included into Intel IPP Samples. See [introduction](#) to this section.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
--------------------------	---------------------

`ippStsNullPtrErr`Indicates an error when at least one input pointer is `NULL`.

## AddC8x8

*Adds a constant to 8x8 block with saturation.*

---

### Syntax

```
IppStatus ippAddC8x8_16s8u_C1IR(Ipp16s value, Ipp8u* pSrcDst, int srcDstStep);
```

### Parameters

<code>value</code>	Constant value to be added to the block.
<code>pSrcDst</code>	Pointer to the source/destination block.
<code>srcDstStep</code>	Distance in bytes between starts of the consecutive lines in the destination block.

### Description

The function `ippAddC8x8_16s8u_C1IR` is declared in the `ippvc.h` file. This function adds a 16s value to an 8u block of 8x8 size with saturation.

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when at least one input pointer is <code>NULL</code> .

## Average8x8, Average16x16

*Average two blocks.*

---

### Syntax

```
IppStatus ippAverage8x8_8u_C1IR(Ipp8u* pSrc, int srcStep, Ipp8u* pSrcDst, int srcDstStep);
```

```
IppStatus ippAverage8x8_8u_C1R(const Ipp8u* pSrc1, int src1Step, const Ipp8u* pSrc2, int src2Step, Ipp8u* pDst, int dstStep);
```

```
IppStatus ippAverage16x16_8u_C1IR(Ipp8u* pSrc, int srcStep, Ipp8u* pSrcDst, int srcDstStep);
```

```
IppStatus ippAverage16x16_8u_C1R(const Ipp8u* pSrc1, int src1Step, const Ipp8u* pSrc2, int src2Step, Ipp8u* pDst, int dstStep);
```

### Parameters

<code>pSrc, pSrc1, pSrc2</code>	Pointers to the source blocks.
<code>pDst</code>	Pointer to the destination block.
<code>pSrcDst</code>	Pointer to the source and destination blocks.
<code>srcStep, src1Step, src2Step</code>	Width in bytes through the source planes.

<i>dstStep</i>	Width in bytes through the destination planes.
<i>srcDstStep</i>	Width in bytes through the source and destination planes.

## Description

The functions `ippiAverage8x8_8u_C1R`, `ippiAverage8x8_8u_C1R`, `ippiAverage16x16_8u_C1R`, and `ippiAverage16x16_8u_C1R` are declared in the `ippvc.h` file. These functions average two blocks pixel-by-pixel, as specified by [ISO14496], subclause 7.6.9.4. The functions can be used in the B-frame reconstruction.

These functions are used in the MPEG-4 encoders and decoders included into Intel IPP Samples. See [introduction](#) to this section.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when at least one input pointer is <code>NULL</code> .

## Motion Estimation

These functions calculate:

- residual block - difference between source block and predicted block ([Table “Evaluation of Residual Blocks”](#))
- some characteristics of the residual block ([Table “Evaluation of Residual Block Characteristics”](#))
- some characteristics of the blocks. These characteristics can be used for comparison of the blocks. ([Table “Evaluation of Block Characteristics”](#)).

The use of some functions described in this section is demonstrated in Intel® IPP Samples downloadable from <http://www.intel.com/cd/software/products/asmo-na/eng/220046.htm>.

### Evaluation of Residual Block

For predicted blocks	For bi-predicted blocks
<a href="#">GetDiff16x16</a>	<a href="#">GetDiff16x16B</a>
<a href="#">GetDiff16x8</a>	<a href="#">GetDiff16x8B</a>
<a href="#">GetDiff8x8</a>	<a href="#">GetDiff8x8B</a>
<a href="#">GetDiff8x16</a>	<a href="#">GetDiff8x16B</a>
<a href="#">GetDiff8x4</a>	<a href="#">GetDiff8x4B</a>
<a href="#">GetDiff4x4</a>	
<a href="#">Sub4x4</a> , <a href="#">Sub8x8</a> , <a href="#">Sub16x16</a>	
<a href="#">SubSAD8x8</a>	

### Evaluation of Residual Block Characteristics

For predicted blocks	For bi-predicted blocks
<a href="#">SqrDiff16x16</a>	<a href="#">SqrDiff16x16B</a>
<a href="#">SSD8x8</a>	

[SSD4x4](#)

Evaluation of variances and means of blocks of difference between two blocks

[VarMeanDiff16x16](#)

[VarMeanDiff16x8](#)

SAD functions

[SAD16x16](#)

[SAD16x16xN](#), [SAD16x16xNI](#)

[SAD16x8](#)

[SAD8x16](#)

[SAD8x8](#)

[SAD8x8xN](#), [SAD8x8xNI](#)

[SAD8x4](#)

[SAD4x8](#)

[SAD4x4](#)

[SAD4x4xN](#), [SAD4x4xNI](#)

[SAD16x16Blocks8x8](#)

[SAD16x16Blocks4x4](#)

[SATD16x16](#)

[SATD16x8](#)

[SATD8x16](#)

[SATD8x8](#)

[SATD8x4](#)

[SATD4x8](#)

[SATD4x4](#)

[SAT8x8D](#)

[FrameFieldSAD16x16](#)

Sum of differences evaluation

[SumsDiff16x16Blocks4x4](#)

[SumsDiff8x8Blocks4x4](#)

## Evaluation of Blocks Characteristics

Block variance and mean evaluation

[VarMean8x8](#)

Block variance evaluation

[Variance16x16](#)

Evaluation of block deviation

[MeanAbsDev8x8](#)

[MeanAbsDev16x16](#)

Edges detection

[EdgesDetect16x16](#)

## Difference Evaluation

These functions calculate residual block - difference between the source block and the predicted block.

Difference evaluation functions are divided into two groups:

- Functions for predicted block, using one reference block for prediction
- Functions for bi-predicted block, using two reference blocks for prediction; in this case predictions are first calculated for each reference block, and then prediction is calculated as average of two predictions.

### Predicted Blocks

These functions use one reference block for prediction.

## GetDiff16x16

*Evaluates difference between current predicted and reference blocks of 16x16 elements.*

### Syntax

```
IppStatus ippiGetDiff16x16_8u16s_C1(const Ipp8u* pSrcCur, Ipp32s srcCurStep, const Ipp8u* pSrcRef, Ipp32s srcRefStep, Ipp16s* pDstDiff, Ipp32s dstDiffStep, Ipp16s* pDstPredictor, Ipp32s dstPredictorStep, Ipp32s mcType, Ipp32s roundControl);
```

### Parameters

<i>pSrcCur</i>	Pointer to the block of size 16x16 in the current plane.
<i>srcCurStep</i>	Step of the current block, specifying width of the plane in bytes.
<i>pSrcRef</i>	Pointer to the block of size 16x16 in the reference plane.
<i>srcRefStep</i>	Step of the reference block, specifying width of the plane in bytes.
<i>pDstDiff</i>	Pointer to the block of size 16x16 in the destination plane, which contains difference between the current and reference blocks.
<i>dstDiffStep</i>	Step of the destination block, specifying width of the block in bytes.

<i>pDstPredictor</i>	Pointer to the destination block of size 16x16 that contains a block of predictors for the current block. The predictor is calculated from the reference block taking into account <i>mcType</i> . If predictor is not used, it must be 0.
<i>dstPredictorStep</i>	Step of the <i>pDstPredictor</i> block in bytes.
<i>mcType</i>	Type of the following MC type <a href="#">IPPVC_MC_APX</a> .
<i>roundControl</i>	Parameter that determines type of rounding for half a pel approximation; may be 0 or 1.

## Description

This function is declared in the `ippvc.h` header file. The function `ippiGetDiff16x16_8u16s_C1` evaluates the difference between the current block of specified size and the reference one. The result is stored in blocks *pDstDiff* and *pDstPredictor*. The latter stores some additional information about the coding block. This information is used for encoding next blocks that refer to the current one. This method helps to decrease the number of encoding errors. Encoding is performed accurate to half a pel and rounding must be specified.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when at least one input pointer is <code>NULL</code> .

## GetDiff16x8

*Evaluates difference between current predicted and reference blocks of 16x8 elements.*

---

## Syntax

```
ippStatus ippiGetDiff16x8_8u16s_C1(const Ipp8u* pSrcCur, Ipp32s srcCurStep, const Ipp8u* pSrcRef, Ipp32s srcRefStep, Ipp16s* pDstDiff, Ipp32s dstDiffStep, Ipp16s* pDstPredictor, Ipp32s dstPredictorStep, Ipp32s mcType, Ipp32s roundControl);
```

## Parameters

<i>pSrcCur</i>	Pointer to the block of size 16x8 in the current plane.
<i>srcCurStep</i>	Step of the current block, specifying width of the plane in bytes.
<i>pSrcRef</i>	Pointer to the block of size 16x8 in the reference plane.
<i>srcRefStep</i>	Step of the reference block, specifying width of the plane in bytes.
<i>pDstDiff</i>	Pointer to the block of size 16x8 in the destination plane, which contains difference between the current and reference blocks.
<i>dstDiffStep</i>	Step of the destination block, specifying width of the block in bytes.
<i>pDstPredictor</i>	Pointer to the destination block of size 16x8 that contains a block of predictors for the current block. The predictor is calculated from the reference block taking into account <i>mcType</i> . If predictor is not used, it must be 0.
<i>dstPredictorStep</i>	Step of the <i>pDstPredictor</i> block in bytes.

<i>mcType</i>	Type of the following MC type <a href="#">IPPVC_MC_APX</a> .
<i>roundControl</i>	Parameter that determines type of rounding for half a pel approximation; may be 0 or 1.

## Description

This function is declared in the `ippvc.h` header file. The function `ippiGetDiff16x8_8u16s_C1` evaluates the difference between the current block of specified size and the reference one. The result is stored in blocks *pDstDiff* and *pDstPredictor*. The latter stores some additional information about the coding block. This information is used for encoding next blocks that refer to the current one. This method helps to decrease the number of encoding errors. Encoding is performed accurate to half a pel and rounding must be specified.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when at least one input pointer is NULL.

## GetDiff8x8

*Evaluates difference between current predicted and reference blocks of 8x8 elements.*

## Syntax

```

IppStatus ippiGetDiff8x8_8u16s_C1(const Ipp8u* pSrcCur, Ipp32s srcCurStep, const Ipp8u*
pSrcRef, Ipp32s srcRefStep, Ipp16s* pDstDiff, Ipp32s dstDiffStep, Ipp16s* pDstPredictor,
Ipp32s dstPredictorStep, Ipp32s mcType, Ipp32s roundControl);

IppStatus ippiGetDiff8x8_8u16s_C2P2(const Ipp8u* pSrcCur, Ipp32s srcCurStep, const Ipp8u*
pSrcRef, Ipp32s srcRefStep, Ipp16s* pDstDiffU, Ipp32s dstDiffStepU, Ipp16s* pDstDiffV,
Ipp32s dstDiffStepV, Ipp32s mcType, Ipp32s roundControl);

```

## Parameters

<i>pSrcCur</i>	<p>Pointer to the block of size 8x8 in the current plane. For <code>ippiGetDiff8x8_8u16s_C2P2</code>, pointer to the block of size 8x16 in the current plane in the NV12 format (or 8x8 for each U and V planes):</p> <pre> 0 UVUVUVUVUVUVUVUV 1 UVUVUVUVUVUVUVUV ... 7 UVUVUVUVUVUVUVUV </pre>
<i>srcCurStep</i>	Step of the current block, specifying width of the plane in bytes.
<i>pSrcRef</i>	Pointer to the block of size 8x8 in the reference plane (for <code>ippiGetDiff8x8_8u16s_C2P2</code> , pointer to the block of size 8x16 in the reference plane in the NV12 format).
<i>srcRefStep</i>	Step of the reference block, specifying width of the plane in bytes.
<i>pDstDiff</i>	Pointer to the block of size 8x8 in the destination plane, which contains difference between the current and reference blocks.

<i>pDstDiffU</i>	Pointer to the block of size 8x8 in the destination plane, which contains difference between the current and reference blocks (U plane): 0 UUUUUUUU 1 UUUUUUUU ... 7 UUUUUUUU
<i>pDstDiffV</i>	Pointer to the block of size 8x8 in the destination plane, which contains difference between the current and reference blocks (V plane): 0 VVVVVVVV 1 VVVVVVVV ... 7 VVVVVVVV
<i>dstDiffStep</i>	Step of the destination block, specifying width of the block in bytes.
<i>dstDiffStepU</i>	Step of the destination block, specifying width of the block in bytes (U plane).
<i>dstDiffStepV</i>	Step of the destination block, specifying width of the block in bytes (V plane).
<i>mcType</i>	Type of the following MC type <a href="#">IPPVC_MC_APX</a> .
<i>roundControl</i>	Parameter that determines type of rounding for half a pel approximation; may be 0 or 1.

## Description

This function is declared in the `ippvc.h` header file. The function `ippiGetDiff8x4_8u16s` evaluates the difference between the current block of specified size and the reference one.

For `ippiGetDiff8x8_8u16s_C1`, the result is stored in blocks *pDstDiff* and *pDstPredictor*. The latter stores some additional information about the coding block. This information is used for encoding next blocks that refer to the current one.

For `ippiGetDiff8x8_8u16s_C2P2`, the result is stored in blocks *pDstDiffU* and *pDstDiffV*.

This method helps to decrease the number of encoding errors. Encoding is performed accurate to half a pel and rounding must be specified.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when at least one input pointer is <code>NULL</code> .

## GetDiff8x16

*Evaluates difference between current predicted and reference blocks of 8x16 elements.*

---

### Syntax

```
ippStatus ippiGetDiff8x16_8u16s_C1(const Ipp8u* pSrcCur, Ipp32s srcCurStep, const Ipp8u*
pSrcRef, Ipp32s srcRefStep, Ipp16s* pDstDiff, Ipp32s dstDiffStep, Ipp16s* pDstPredictor,
Ipp32s dstPredictorStep, Ipp32s mcType, Ipp32s roundControl);
```



## Parameters

<i>pSrcCur</i>	Pointer to the block of size 8x16 in the current plane.
<i>srcCurStep</i>	Step of the current block, specifying width of the plane in bytes.
<i>pSrcRef</i>	Pointer to the block of size 8x16 in the reference plane.
<i>srcRefStep</i>	Step of the reference block, specifying width of the plane in bytes.
<i>pDstDiff</i>	Pointer to the block of size 8x16 in the destination plane, which contains difference between the current and reference blocks.
<i>dstDiffStep</i>	Step of the destination block, specifying width of the block in bytes.
<i>pDstPredictor</i>	Pointer to the destination block of size 8x16 that contains a block of predictors for the current block. The predictor is calculated from the reference block taking into account <i>mcType</i> . If predictor is not used, it must be 0.
<i>dstPredictorStep</i>	Step of the <i>pDstPredictor</i> block in bytes.
<i>mcType</i>	Type of the following MC type <a href="#">IPPVC_MC_APX</a> .
<i>roundControl</i>	Parameter that determines type of rounding for half a pel approximation; may be 0 or 1.

## Description

This function is declared in the `ippvc.h` header file. The function `ippiGetDiff8x16_8u16s_C1` evaluates the difference between the current block of specified size and the reference one. The result is stored in blocks *pDstDiff* and *pDstPredictor*. The latter stores some additional information about the coding block. This information is used for encoding next blocks that refer to the current one. This method helps to decrease the number of encoding errors. Encoding is performed accurate to half a pel and rounding must be specified.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when at least one input pointer is <code>NULL</code> .

## GetDiff8x4

*Evaluates difference between current predicted and reference blocks of 8x4 elements.*

---

## Syntax

```

IppStatus ippiGetDiff8x4_8u16s_C1(const Ipp8u* pSrcCur, Ipp32s srcCurStep, const Ipp8u*
pSrcRef, Ipp32s srcRefStep, Ipp16s* pDstDiff, Ipp32s dstDiffStep, Ipp16s* pDstPredictor,
Ipp32s dstPredictorStep, Ipp32s mcType, Ipp32s roundControl);

IppStatus ippiGetDiff8x4_8u16s_C2P2(const Ipp8u* pSrcCur, Ipp32s srcCurStep, const Ipp8u*
pSrcRef, Ipp32s srcRefStep, Ipp16s* pDstDiffU, Ipp32s dstDiffStepU, Ipp16s* pDstDiffV,
Ipp32s dstDiffStepV, Ipp32s mcType, Ipp32s roundControl);

```

## Parameters

<i>pSrcCur</i>	Pointer to the block of size 8x4 in the current plane.
----------------	--

	For <code>ippiGetDiff8x4_8u16s_C2P2</code> , pointer to the block of size 8x8 in the current plane in the NV12 format (or 8x4 for each U and V planes):
	<pre> 0 UVUVUVUV 1 UVUVUVUV ... 7 UVUVUVUV </pre>
<code>srcCurStep</code>	Step of the current block, specifying width of the plane in bytes.
<code>pSrcRef</code>	Pointer to the block of size 8x4 in the reference plane (for <code>ippiGetDiff8x4_8u16s_C2P2</code> , pointer to the block of size 8x8 in the reference plane in the NV12 format).
<code>srcRefStep</code>	Step of the reference block, specifying width of the plane in bytes.
<code>pDstDiff</code>	Pointer to the block of size 8x4 in the destination plane, which contains difference between the current and reference blocks.
<code>pDstDiffU</code>	Pointer to the block of size 8x4 in the destination plane, which contains difference between the current and reference blocks (U plane):
	<pre> 0 UUUU 1 UUUU ... 7 UUUU </pre>
<code>pDstDiffV</code>	Pointer to the block of size 8x4 in the destination plane, which contains difference between the current and reference blocks (V plane):
	<pre> 0 VVVV 1 VVVV ... 7 VVVV </pre>
<code>dstDiffStep</code>	Step of the destination block, specifying width of the block in bytes.
<code>dstDiffStepU</code>	Step of the destination block, specifying width of the block in bytes (U plane).
<code>dstDiffStepV</code>	Step of the destination block, specifying width of the block in bytes (V plane).
<code>mcType</code>	Type of the following MC type <a href="#">IPPVC_MC_APX</a> .
<code>roundControl</code>	Parameter that determines type of rounding for half a pel approximation; may be 0 or 1.

## Description

This function is declared in the `ippvc.h` header file. The function `ippiGetDiff8x4_8u16s` evaluates the difference between the current block of specified size and the reference one.

For `ippiGetDiff8x4_8u16s_C1`, the result is stored in blocks `pDstDiff` and `pDstPredictor`. The latter stores some additional information about the coding block. This information is used for encoding next blocks that refer to the current one.

For `ippiGetDiff8x4_8u16s_C2P2`, the result is stored in blocks `pDstDiffU` and `pDstDiffV`.

This method helps to decrease the number of encoding errors. Encoding is performed accurate to half a pel and rounding must be specified.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when at least one input pointer is <code>NULL</code> .

## GetDiff4x4

*Computes difference between current predicted and reference 4x4 blocks.*

---

### Syntax

```
ippStatus ippiGetDiff4x4_8u16s_C1(const Ipp8u* pSrcCur, Ipp32s srcCurStep, const Ipp8u*
pSrcRef, Ipp32s srcRefStep, Ipp16s* pDstDiff, Ipp32s dstDiffStep, Ipp16s* pDstPredictor,
Ipp32s dstPredictorStep, Ipp32s mcType, Ipp32s roundControl);
```

### Parameters

<code>pSrcCur</code>	Pointer to the block of size 4x4 in the current plane.
<code>srcCurStep</code>	Step of the current block, specifying width of the plane in bytes.
<code>pSrcRef</code>	Pointer to the block of size 4x4 in the reference plane.
<code>srcRefStep</code>	Step of the reference block, specifying width of the plane in bytes.
<code>pDstDiff</code>	Pointer to the block of size 4x4 in the destination plane, which contains difference between the current and reference blocks.
<code>dstDiffStep</code>	Step of the destination block, specifying width of the block in bytes.
<code>pDstPredictor</code>	Reserved parameter (must be 0).
<code>dstPredictorStep</code>	Reserved parameter (must be 0).
<code>mcType</code>	Reserved parameter (must be 0).
<code>roundControl</code>	Reserved parameter (must be 0).

### Description

This function is declared in the `ippvc.h` header file. The function `ippiGetDiff4x4_8u16s_C1` computes the difference between predictor 4x4 block and source 4x4 block.

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when at least one input pointer is <code>NULL</code> .

### Bi-Predicted Blocks

These functions use two reference blocks for prediction. At first, predictions are calculated for each reference block, and then prediction is calculated as average of two predictions.

## GetDiff16x16B

*Evaluates difference between current bi-predicted and mean of two reference blocks of 16x16 elements.*

---

### Syntax

```
IppStatus ippiGetDiff16x16B_8u16s_C1(const Ipp8u* pSrcCur, Ipp32s srcCurStep, const Ipp8u* pSrcRefF, Ipp32s srcRefStepF, Ipp32s mcTypeF, const Ipp8u* pSrcRefB, Ipp32s srcRefStepB, Ipp32s mcTypeB, Ipp16s* pDstDiff, Ipp32s dstDiffStep, Ipp32s roundControl);
```

### Parameters

<i>pSrcCur</i>	Pointer to the block of size 16x16 in the current plane.
<i>srcCurStep</i>	Step of the current block, specifying width of the plane in bytes.
<i>pSrcRefF</i>	Pointer to the forward block of size 16x16 in the reference plane.
<i>srcRefStepF</i>	Step of the forward reference block, specifying width of the plane in bytes.
<i>mcTypeF</i>	Type of the following forward MC type <a href="#">IPPVC_MC_APX</a> .
<i>pSrcRefB</i>	Pointer to the backward block of size 16x16 in the reference plane.
<i>srcRefStepB</i>	Step of the backward reference block, specifying width of the block in bytes.
<i>mcTypeB</i>	Type of the following backward MC type <a href="#">IPPVC_MC_APX</a> .
<i>pDstDiff</i>	Pointer to the destination block of size 16x16 that contains difference between current and reference blocks.
<i>dstDiffStep</i>	Step of the destination block, specifying width of the block in bytes.
<i>roundControl</i>	Parameter that determines type of rounding for half a pel approximation; may be 0 or 1.

### Description

This function is declared in the `ippvc.h` header file. The function `ippiGetDiff16x16B_8u16s_C1` evaluates the difference between the current block and the mean of two reference blocks of specified size. One of the reference blocks is called forward and belongs to the previous frame in accordance with the type of motion compensation. The other block is called backward and belongs to one of the following frames. The result is stored in block *pDstDiff*. Encoding is performed accurate to half a pel and rounding must be specified.

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when at least one input pointer is <code>NULL</code> .

## GetDiff16x8B

*Evaluates difference between current bi-predicted and mean of two reference blocks of 16x8 elements.*

---

### Syntax

```
IppStatus ippiGetDiff16x8B_8u16s_C1(const Ipp8u* pSrcCur, Ipp32s srcCurStep, const Ipp8u* pSrcRefF, Ipp32s srcRefStepF, Ipp32s mcTypeF, const Ipp8u* pSrcRefB, Ipp32s srcRefStepB, Ipp32s mcTypeB, Ipp16s* pDstDiff, Ipp32s dstDiffStep, Ipp32s roundControl);
```

### Parameters

<i>pSrcCur</i>	Pointer to the block of size 16x8 in the current plane.
<i>srcCurStep</i>	Step of the current block, specifying width of the plane in bytes.
<i>pSrcRefF</i>	Pointer to the forward block of size 16x8 in the reference plane.
<i>srcRefStepF</i>	Step of the forward reference block, specifying width of the plane in bytes.
<i>mcTypeF</i>	Type of the following forward MC type <a href="#">IPPVC_MC_APX</a> .
<i>pSrcRefB</i>	Pointer to the backward block of size 16x8 in the reference plane.
<i>srcRefStepB</i>	Step of the backward reference block, specifying width of the block in bytes.
<i>mcTypeB</i>	Type of the following backward MC type <a href="#">IPPVC_MC_APX</a> .
<i>pDstDiff</i>	Pointer to the destination block of specified size containing difference between the current and reference blocks.
<i>dstDiffStep</i>	Step of the destination block, specifying width of the block in bytes.
<i>roundControl</i>	Parameter that determines type of rounding for half a pel approximation; may be 0 or 1.

### Description

This function is declared in the `ippvc.h` header file. The function `ippiGetDiff16x8B_8u16s_C1` evaluates the difference between the current block and the mean of two reference blocks of specified size. One of the reference blocks is called forward and belongs to the previous frame in accordance with the type of motion compensation. The other block is called backward and belongs to one of the following frames. The result is stored in block *pDstDiff*. Encoding is performed accurate to half a pel and rounding must be specified.

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when at least one input pointer is <code>NULL</code> .

## GetDiff8x8B

*Evaluates difference between current bi-predicted and mean of two reference blocks of 8x8 elements.*

### Syntax

```
IppStatus ippiGetDiff8x8B_8u16s_C1(const Ipp8u* pSrcCur, Ipp32s srcCurStep, const Ipp8u*
pSrcRefF, Ipp32s srcRefStepF, Ipp32s mcTypeF, const Ipp8u* pSrcRefB, Ipp32s srcRefStepB,
Ipp32s mcTypeB, Ipp16s* pDstDiff, Ipp32s dstDiffStep, Ipp32s roundControl);
```

```
IppStatus ippiGetDiff8x8B_8u16s_C2P2(const Ipp8u* pSrcCur, Ipp32s srcCurStep, const Ipp8u*
pSrcRefF, Ipp32s srcRefStepF, Ipp32s mcTypeF, const Ipp8u* pSrcRefB, Ipp32s srcRefStepB,
Ipp32s mcTypeB, Ipp16s* pDstDiffU, Ipp32s dstDiffStepU, Ipp16s* pDstDiffV, Ipp32s
dstDiffStepV, Ipp32s roundControl);
```

### Parameters

<i>pSrcCur</i>	<p>Pointer to the block of size 8x8 in the current plane.</p> <p>For <code>ippiGetDiff8x8B_8u16s_C2P2</code>, pointer to the block of size 8x16 in the current plane in the NV12 format (or 8x8 for each U and V planes):</p> <pre>0 UVUVUVUVUVUVUVUV 1 UVUVUVUVUVUVUVUV ... 7 UVUVUVUVUVUVUVUV</pre>
<i>srcCurStep</i>	Step of the current block, specifying width of the plane in bytes.
<i>pSrcRefF</i>	<p>Pointer to the forward block of size 8x8 in the reference plane (for <code>ippiGetDiff8x8B_8u16s_C2P2</code>, pointer to the block of size 8x16 in the reference plane in the NV12 format).</p> <pre>0 UVUVUVUVUVUVUVUV 1 UVUVUVUVUVUVUVUV ... 7 UVUVUVUVUVUVUVUV</pre>
<i>srcRefStepF</i>	Step of the forward reference block, specifying width of the plane in bytes.
<i>mcTypeF</i>	Type of the following forward MC type <a href="#">IPPVC_MC_APX</a> .
<i>pSrcRefB</i>	<p>Pointer to the backward block of size 8x8 in the reference plane (for <code>ippiGetDiff8x8B_8u16s_C2P2</code>, pointer to the block of size 8x16 in the reference plane in the NV12 format).</p> <pre>0 UVUVUVUVUVUVUVUV 1 UVUVUVUVUVUVUVUV ... 7 UVUVUVUVUVUVUVUV</pre>
<i>srcRefStepB</i>	Step of the backward reference block, specifying width of the block in bytes.
<i>mcTypeB</i>	Type of the following backward MC type <a href="#">IPPVC_MC_APX</a> .

<i>pDstDiff</i>	Pointer to the destination block of specified size containing difference between the current and reference blocks.
<i>pDstDiffU</i>	Pointer to the block of size 8x8 in the destination plane, which contains difference between the current and reference blocks (U plane): <pre> 0  UUUUUUUU 1  UUUUUUUU ... 7  UUUUUUUU </pre>
<i>pDstDiffV</i>	Pointer to the block of size 8x8 in the destination plane, which contains difference between the current and reference blocks (V plane): <pre> 0  VVVVVVVV 1  VVVVVVVV ... 7  VVVVVVVV </pre>
<i>dstDiffStep</i>	Step of the destination block, specifying width of the block in bytes.
<i>dstDiffStepU</i>	Step of the destination block, specifying width of the block in bytes (U plane).
<i>dstDiffStepV</i>	Step of the destination block, specifying width of the block in bytes (V plane).
<i>roundControl</i>	Parameter that determines type of rounding for half a pel approximation; may be 0 or 1.

## Description

This function is declared in the `ippvc.h` header file. The function `ippiGetDiff8x8B_8u16s` evaluates the difference between the current block and the mean of two reference blocks of the specified size. One of the reference blocks is called forward and belongs to the previous frame in accordance with the type of motion compensation. The other block is called backward and belongs to one of the following frames. The result is stored in block *pDstDiff* (for `ippiGetDiff8x8B_8u16s_C1`) or blocks *pDstDiffU* and *pDstDiffV* (for `ippiGetDiff8x8B_8u16s_C2P2`). Encoding is performed accurate to half a pel and rounding must be specified.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when at least one input pointer is NULL.

## GetDiff8x16B

*Evaluates difference between current bi-predicted and mean of two reference blocks of 8x16 elements.*

## Syntax

```

IppStatus ippiGetDiff8x16B_8u16s_C1(const Ipp8u* pSrcCur, Ipp32s srcCurStep, const Ipp8u*
pSrcRefF, Ipp32s srcRefStepF, Ipp32s mcTypeF, const Ipp8u* pSrcRefB, Ipp32s srcRefStepB,
Ipp32s mcTypeB, Ipp16s* pDstDiff, Ipp32s dstDiffStep, Ipp32s roundControl);

```

## Parameters

<i>pSrcCur</i>	Pointer to the block of size 8x16 in the current plane.
<i>srcCurStep</i>	Step of the current block, specifying width of the plane in bytes.
<i>pSrcRefF</i>	Pointer to the forward block of size 8x16 in the reference plane.
<i>srcRefStepF</i>	Step of the forward reference block, specifying width of the plane in bytes.
<i>mcTypeF</i>	Type of the following forward MC type <a href="#">IPPVC_MC_APX</a> .
<i>pSrcRefB</i>	Pointer to the backward block of size 8x16 in the reference plane.
<i>srcRefStepB</i>	Step of the backward reference block, specifying width of the block in bytes.
<i>mcTypeB</i>	Type of the following backward MC type <a href="#">IPPVC_MC_APX</a> .
<i>pDstDiff</i>	Pointer to the destination block of specified size containing difference between the current and reference blocks.
<i>dstDiffStep</i>	Step of the destination block, specifying width of the block in bytes.
<i>roundControl</i>	Parameter that determines type of rounding for half a pel approximation; may be 0 or 1.

## Description

This function is declared in the `ippvc.h` header file. The function `ippiGetDiff8x16B_8u16s_C1` evaluates the difference between the current block and the mean of two reference blocks of specified size. One of the reference blocks is called forward and belongs to the previous frame in accordance with the type of motion compensation. The other block is called backward and belongs to one of the following frames. The result is stored in block *pDstDiff*. Encoding is performed accurate to half a pel and rounding must be specified.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when at least one input pointer is <code>NULL</code> .

## GetDiff8x4B

*Evaluates difference between current bi-predicted and mean of two reference blocks of 8x4 elements.*

---

## Syntax

```
IppStatus ippiGetDiff8x4B_8u16s_C1(const Ipp8u* pSrcCur, Ipp32s srcCurStep, const Ipp8u*
pSrcRefF, Ipp32s srcRefStepF, Ipp32s mcTypeF, const Ipp8u* pSrcRefB, Ipp32s srcRefStepB,
Ipp32s mcTypeB, Ipp16s* pDstDiff, Ipp32s dstDiffStep, Ipp32s roundControl);

IppStatus ippiGetDiff8x4B_8u16s_C2P2(const Ipp8u* pSrcCur, Ipp32s srcCurStep, const Ipp8u*
pSrcRefF, Ipp32s srcRefStepF, Ipp32s mcTypeF, const Ipp8u* pSrcRefB, Ipp32s srcRefStepB,
Ipp32s mcTypeB, Ipp16s* pDstDiffU, Ipp32s dstDiffStepU, Ipp16s* pDstDiffV, Ipp32s
dstDiffStepV, Ipp32s roundControl);
```



## Parameters

<i>pSrcCur</i>	<p>Pointer to the block of size 8x4 in the current plane.  For <code>ippiGetDiff8x4B_8u16s_C2P2</code>, pointer to the block of size 8x8 in the current plane in the NV12 format (or 8x4 for each U and V planes):</p> <pre> 0 UVUVUVUV 1 UVUVUVUV ... 7 UVUVUVUV </pre>
<i>srcCurStep</i>	Step of the current block, specifying width of the plane in bytes.
<i>pSrcRefF</i>	<p>Pointer to the forward block of size 8x4 in the reference plane (for <code>ippiGetDiff8x4B_8u16s_C2P2</code>, pointer to the block of size 8x8 in the reference plane in the NV12 format).</p> <pre> 0 UVUVUVUV 1 UVUVUVUV ... 7 UVUVUVUV </pre>
<i>srcRefStepF</i>	Step of the forward reference block, specifying width of the plane in bytes.
<i>mcTypeF</i>	Type of the following forward MC type <code>IPPVC_MC_APX</code> .
<i>pSrcRefB</i>	<p>Pointer to the backward block of size 8x4 in the reference plane (for <code>ippiGetDiff8x4B_8u16s_C2P2</code>, pointer to the block of size 8x8 in the reference plane in the NV12 format).</p> <pre> 0 UVUVUVUV 1 UVUVUVUV ... 7 UVUVUVUV </pre>
<i>srcRefStepB</i>	Step of the backward reference block, specifying width of the block in bytes.
<i>mcTypeB</i>	Type of the following backward MC type <code>IPPVC_MC_APX</code> .
<i>pDstDiff</i>	Pointer to the destination block of specified size containing difference between the current and reference blocks.
<i>pDstDiffU</i>	<p>Pointer to the block of size 8x4 in the destination plane, which contains difference between the current and reference blocks (U plane):</p> <pre> 0 UUUU 1 UUUU ... 7 UUUU </pre>
<i>pDstDiffV</i>	<p>Pointer to the block of size 8x4 in the destination plane, which contains difference between the current and reference blocks (V plane):</p> <pre> 0 VVVV 1 VVVV ... 7 VVVV </pre>

<i>dstDiffStep</i>	Step of the destination block, specifying width of the block in bytes.
<i>dstDiffStepU</i>	Step of the destination block, specifying width of the block in bytes (U plane).
<i>dstDiffStepV</i>	Step of the destination block, specifying width of the block in bytes (V plane).
<i>roundControl</i>	Parameter that determines type of rounding for half a pel approximation; may be 0 or 1.

## Description

This function is declared in the `ippvc.h` header file. The function `ippiGetDiff8x4B_8u16s` evaluates the difference between the current block and the mean of two reference blocks of the specified size. One of the reference blocks is called forward and belongs to the previous frame in accordance with the type of motion compensation. The other block is called backward and belongs to one of the following frames. The result is stored in block *pDstDiff* (for `ippiGetDiff8x4B_8u16s_C1`) or blocks *pDstDiffu* and *pDstDiffv* (for `ippiGetDiff8x4B_8u16s_C2P2`). Encoding is performed accurate to half a pel and rounding must be specified.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when at least one input pointer is <code>NULL</code> .

## Sub4x4, Sub8x8, Sub16x16

*Subtract two blocks and store the result in the third block.*

---

## Syntax

```
ippStatus ippiSub4x4_8u16s_C1R(const Ipp8u* pSrc1, int src1Step, const Ipp8u* pSrc2, int src2Step, Ipp16s* pDst, int dstStep);
```

```
ippStatus ippiSub4x4_16u16s_C1R(const Ipp16u* pSrc1, int src1Step, const Ipp16u* pSrc2, int src2Step, Ipp16s* pDst, int dstStep);
```

```
ippStatus ippiSub8x8_8u16s_C1R(const Ipp8u* pSrc1, int src1Step, const Ipp8u* pSrc2, int src2Step, Ipp16s* pDst, int dstStep);
```

```
ippStatus ippiSub8x8_16u16s_C1R(const Ipp16u* pSrc1, Ipp32s src1Step, const Ipp16u* pSrc2, Ipp32s src2Step, Ipp16s* pDst, Ipp32s dstStep);
```

```
ippStatus ippiSub16x16_8u16s_C1R (const Ipp8u* pSrc1, int src1Step, const Ipp8u* pSrc2, int src2Step, Ipp16s* pDst, int dstStep);
```

## Parameters

<i>pSrc1</i>	Pointer to the first source block.
<i>src1Step</i>	Step in bytes through the first source plane.
<i>pSrc2</i>	Pointer to the second source block.
<i>src2Step</i>	Step in bytes through the second source plane.

<i>pDst</i>	Pointer to the destination block.
<i>dstStep</i>	Step in bytes through the destination plane.

## Description

The functions `ippiSub4x4_8u16s_C1R`, `ippiSub4x4_16u16s_C1R`, `ippiSub8x8_8u16s_C1R`, `ippiSub8x8_16u16s_C1R`, and `ippiSub16x16_8u16s_C1R` are declared in the `ippvc.h` file. These functions subtract two blocks and store the result in *pDst* block. The functions can be used in motion estimation process to get a difference of current and reference blocks.

These functions are used in the H.261, H.263, and MPEG-4 encoders included into Intel IPP Samples. See [introduction to this section](#).

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when at least one input pointer is <code>NULL</code> .

## SubSAD8x8

*Subtracts two blocks, stores the result in the third block and computes a sum of absolute differences.*

## Syntax

```
IppStatus ippiSubSAD8x8_8u16s_C1R(const Ipp8u* pSrc1, int src1Step, const Ipp8u* pSrc2,
int src2Step, Ipp16s* pDst, int dstStep, Ipp32s* pSAD);
```

## Parameters

<i>pSrc1</i>	Pointer to the first source block.
<i>src1Step</i>	Step in bytes through the first source plane.
<i>pSrc2</i>	Pointer to the second source block.
<i>src2Step</i>	Step in bytes through the second source plane.
<i>pDst</i>	Pointer to the destination block.
<i>dstStep</i>	Step in bytes through the destination plane.
<i>pSAD</i>	Pointer to the resulting SAD.

## Description

The function `ippiSubSAD8x8_8u16s_C1R` is declared in the `ippvc.h` file. This function subtracts two blocks and stores the result in *pDst* block and stores SAD of the result in *pSAD*. This function can be used in motion estimation process to get a difference of current and reference blocks.

This function is used in the H.261, H.263, and MPEG-4 encoders included into Intel IPP Samples. See [introduction to this section](#).

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
--------------------------	---------------------

`ippStsNullPtrErr`Indicates an error when at least one input pointer is `NULL`.

## Sum of Squares of Differences Evaluation

These functions evaluate sum of squares of differences between current and predicted blocks.

These functions are divided into two groups:

- Functions for predicted block, using one reference block for prediction
- Functions for bi-predicted block, using two reference blocks for prediction; in this case predictions are first calculated for each reference block, and then prediction is calculated as average of two predictions.

## SqrDiff16x16

*Evaluates sum of squares of differences between current and reference 16X16 blocks.*

---

### Syntax

```
ippStatus ippISqrDiff16x16_8u32s(const Ipp8u* pSrc, Ipp32s srcStep, const Ipp8u* pRef,
Ipp32s refStep, Ipp32s mcType, Ipp32s* pSqrDiff);
```

### Parameters

<code>pSrc</code>	Pointer to the current block of specified size.
<code>srcStep</code>	Step of the current block, specifying width of the block in bytes.
<code>pRef</code>	Pointer to the reference block of specified size.
<code>refStep</code>	Step of the reference block, specifying width of the block in bytes.
<code>mcType</code>	MC type <a href="#">IPPVC_MC_APX</a> .
<code>pSqrDiff</code>	Pointer to the sum of square difference between all elements in the current and reference blocks.

### Description

This function is declared in the `ippvc.h` header file. The function `ippISqrDiff16x16_8u32s` evaluates the sum of square difference between all the elements in the current block and corresponding elements in the reference block. The result is stored in integer `*pSqrDiff`.

Let us denote the pel which lies at the crossing of  $i^{\text{th}}$  row and  $j^{\text{th}}$  column of the current block as `block[i, j]` and the pel which lies at the crossing of  $i^{\text{th}}$  row and  $j^{\text{th}}$  column of the reference block as `ref_block[i, j]`. Then

$$*pSqrDiff = \sum_{i=0}^{15} \sum_{j=0}^{15} (\text{block}[i, j] - \text{ref\_block}[i, j])^2.$$

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when at least one input pointer is <code>NULL</code> .
<code>ippStsStepErr</code>	Indicates an error when <code>srcStep</code> or <code>refStep</code> is less than or equal to zero.

## SqrDiff16x16B

*Evaluates sum of squares of differences between current bi-predicted 16X16 block and mean of two reference blocks.*

## Syntax

```
IppStatus ippISqrDiff16x16B_8u32s(const Ipp8u* pSrc, Ipp32s srcStep, const Ipp8u pRefF,
Ipp32s refStepF, Ipp32s mcTypeF, const Ipp8u pRefB, Ipp32s refStepB, Ipp32s mcTypeB,
Ipp32s* pSqrDiff);
```

## Parameters

<code>pSrc</code>	Pointer to the current block of specified size.
<code>srcStep</code>	Step of the current block, specifying width of the block in bytes.
<code>pRefF</code>	Pointer to the forward reference block of specified size.
<code>refStepF</code>	Step of the forward reference block, specifying width of the block in bytes.
<code>mcTypeF</code>	MC type <a href="#">IPPVC_MC_APX</a> for the forward reference block.
<code>pRefB</code>	Pointer to the backward reference block of specified size.
<code>refStepB</code>	Step of the backward reference block, specifying width of the block in bytes.
<code>mcTypeB</code>	MC type <a href="#">IPPVC_MC_APX</a> for the backward reference block.
<code>pSqrDiff</code>	Pointer to the sum of square difference between all elements in the current and reference blocks.

## Description

This function is declared in the `ippvc.h` header file. The function `ippISqrDiff16x16B_8u32s` evaluates the sum of square difference between all the elements in the current block and the mean of corresponding elements in two reference blocks of 8x8 elements. The result is stored in integer `pSqrDiff`.

Let us denote the pel which lies at the crossing of  $i^{\text{th}}$  row and  $j^{\text{th}}$  column of the current block as `block[i,j]`, the pel which lies at the crossing of  $i^{\text{th}}$  row and  $j^{\text{th}}$  column of the forward reference block as `f_block[i,j]`, and the pel which lies at the crossing of  $i^{\text{th}}$  row and  $j^{\text{th}}$  column of the backward reference block as `b_block[i,j]`. Then

$$*pSqrDiff = \sum_{i=0}^{15} \sum_{j=0}^{15} \left( block[i, j] - \frac{f\_block[i, j] + b\_block[i, j]}{2} \right)^2$$

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when at least one input pointer is NULL.
<code>ippStsStepErr</code>	Indicates an error when <i>srcStep</i> , <i>refStepF</i> or <i>refStepB</i> is less than or equal to zero.

## SSD8x8

*Evaluates sum of squares of differences between current and reference 8X8 blocks.*

---

## Syntax

```
IppStatus ippISSD8x8_8u32s_C1R(const Ipp8u* srcStep, int srcCurStep, const Ipp8u* pSrcRef,
int srcRefStep, Ipp32s* pDst, Ipp32s mcType);
```

## Parameters

<i>pSrc</i>	Pointer to the current block of specified size.
<i>srcCurStep</i>	Step of the current block, specifying width of the block in bytes.
<i>pSrcRef</i>	Pointer to the reference block of specified size.
<i>srcRefStep</i>	Step of the reference block, specifying width of the block in bytes.
<i>pDst</i>	Pointer to the sum of square difference between all elements in the current and reference blocks.
<i>mcType</i>	MC type <a href="#">IPPVC_MC_APX</a> .

## Description

This function is declared in the `ippvc.h` header file. The function `ippISSD8x8_8u32s_C1R` evaluates the sum of square difference between all the elements in the current block and corresponding elements in the reference block. The result is stored in integer *pDst*.

Let us denote the pel which lies at the crossing of  $i^{\text{th}}$  row and  $j^{\text{th}}$  column of the current block as *block[i, j]* and the pel which lies at the crossing of  $i^{\text{th}}$  row and  $j^{\text{th}}$  column of the reference block as *ref\_block[i, j]*. Then

$$*pSqrDiff = \sum_{i=0}^{15} \sum_{j=0}^{15} (block[i, j] - ref\_block[i, j])^2.$$

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when at least one input pointer is NULL.

## SSD4x4

*Evaluates sum of squares of differences between current and reference 4X4 blocks.*

---

### Syntax

```
ippStatus ippiSSD4x4_8u32s_C1R(const Ipp8u* pSrcCur, int srcCurStep, const Ipp8u* pSrcRef,
int srcRefStep, Ipp32s* pDst, Ipp32s mcType);
```

### Parameters

<code>pSrcCur</code>	Pointer to the current block of specified size.
<code>srcCurStep</code>	Step of the current block, specifying width of the block in bytes.
<code>pSrcRef</code>	Pointer to the reference block of specified size.
<code>srcRefStep</code>	Step of the reference block, specifying width of the block in bytes.
<code>pDst</code>	Pointer to the sum of square difference between all elements in the current and reference blocks.
<code>mcType</code>	MC type <a href="#">IPFVC_MC_APX</a> .

### Description

This function is declared in the `ippvc.h` header file. The function `ippiSSD4x4_8u32s_C1R` evaluates the sum of square difference between all the elements in the current block and corresponding elements in the reference block. The result is stored in integer `*pDst`.

Let us denote the pel which lies at the crossing of  $i^{\text{th}}$  row and  $j^{\text{th}}$  column of the current block as `block[i, j]` and the pel which lies at the crossing of  $i^{\text{th}}$  row and  $j^{\text{th}}$  column of the reference block as `ref_block[i, j]`. Then

$$*pSqrDiff = \sum_{i=0}^{15} \sum_{j=0}^{15} (block[i, j] - ref\_block[i, j])^2.$$

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when at least one input pointer is NULL.

## Block Variance and Mean Evaluation

These functions evaluate sum and variance of all elements in block.

### VarMean8x8

*Evaluates variance and mean of 8X8 block.*

---

#### Syntax

```

IppStatus ippiVarMean8x8_8u32s_C1R(const Ipp8u* pSrc, Ipp32s srcStep, Ipp32s* pVar,
Ipp32s* pMean);

IppStatus ippiVarMean8x8_16s32s_C1R(const Ipp16s* pSrc, Ipp32s srcStep, Ipp32s* pVar,
Ipp32s* pMean);

```

#### Parameters

<i>pSrc</i>	Pointer to the input block 8x8.
<i>srcStep</i>	Input block step.
<i>pVar</i>	Pointer to the variance.
<i>pMean</i>	Pointer to the mean value.

#### Description

This function is declared in the `ippvc.h` header file. The function evaluates the mean and variance of an 8x8 block of unsigned char values (`ippiVarMean8x8_8u32s_C1R`) or short integer values (`ippiVarMean8x8_16s32s_C1R`).

Mean is evaluated by the following formula:

$$M = \sum_{i=0}^7 \sum_{j=0}^7 block[i, j]$$

The actual value of the  $M$  is sum rather than mean yet this designation is preserved for convenience.

Variance is evaluated by the following formula:

$$V = \frac{\sum_{i=0}^7 \sum_{j=0}^7 block[i, j]^2}{64} - \left(\frac{M}{64}\right)^2.$$

This function is used in the MPEG-2 encoder included into Intel IPP Samples. See introduction to this section.



## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when at least one input pointer is <code>NULL</code> .

## Evaluation of Variances and Means of Blocks of Difference Between Two Blocks

These functions evaluate sum and variance of all elements in block.

## VarMeanDiff16x16

*Evaluates variances and means of four 8x8 blocks of difference between two 16x16 blocks.*

### Syntax

```
ippStatus ippiVarMeanDiff16x16_8u32s_C1R(const Ipp8u* pSrc, Ipp32s srcStep, const Ipp8u*
pRef, Ipp32s srcRefStep, Ipp32s* pSrcSum, Ipp32s* pVar, Ipp32s* pMean, Ipp32s mcType);
```

### Parameters

<code>pSrc</code>	Pointer to the current block 16x16.
<code>srcStep</code>	Current block step.
<code>pRef</code>	Pointer to the reference block 16x16.
<code>srcRefStep</code>	Reference block step.
<code>pSrcSum</code>	Pointer to the sum of pixel values for the current block.
<code>pVar</code>	Pointer to the variance.
<code>pMean</code>	Pointer to the mean value.
<code>mcType</code>	MC type <code>IPPVC_MC_APX</code> .

### Description

This function is declared in the `ippvc.h` header file. The function `ippiVarMeanDiff16x16_8u32s_C1R` evaluates the means and variances of four 8x8 blocks of difference between two 16x16 blocks.

As the function is repeatedly called for the same current block, the parameter `pSrcSum` is used to avoid excessive computation. This parameter should point to an array of four 8x8 blocks comprising the current block. The parameters `pVar`, `pMean` should point to arrays of four elements, since the values of variance and mean are calculated separately for each 8x8 block.

Mean is evaluated by the following formula:

$$M = \frac{\text{Sum} - \sum_{i=0}^7 \sum_{j=0}^7 \text{ref\_block}[i, j]}{8}$$

where `Sum` is the sum of pixel values for the current block taken from `pSrcSum` parameter.

Variance is evaluated by the following formula:

$$V = \frac{\left( \sum_{i=0}^7 \sum_{j=0}^7 (cur\_block[i, j] - ref\_block[i, j])^2 \right) - M^2}{64}.$$

This function is used in the MPEG-2 encoder included into Intel IPP Samples. See introduction to this section.

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when at least one input pointer is NULL.

## VarMeanDiff16x8

*Evaluates variances and means of two 8x8 blocks of difference between two 16x8 blocks.*

---

### Syntax

```
ippStatus ippiVarMeanDiff16x8_8u32s_C1R(const Ipp8u* pSrc, Ipp32s srcStep, const Ipp8u*
pRef, Ipp32s srcStep, Ipp32s* pSrcSum, Ipp32s* pVar, Ipp32s* pMean, Ipp32s mcType);
```

### Parameters

<code>pSrc</code>	Pointer to the current block 16x8.
<code>srcStep</code>	Current block step.
<code>pRef</code>	Pointer to the reference block 16x8.
<code>srcStep</code>	Reference block step.
<code>pSrcSum</code>	Pointer to the sum of pixel values for the current block.
<code>pVar</code>	Pointer to the variance.
<code>pMean</code>	Pointer to the mean value.
<code>mcType</code>	MC type <a href="#">IPPVC_MC_APX</a> .

### Description

This function is declared in the `ippvc.h` header file. The function `ippiVarMeanDiff16x8_8u32s_C1R` evaluates the means and variances of two 8x8 blocks of difference between two 16x8 blocks.

As the function is repeatedly called for the same current block, the parameter `pSrcSum` is used to avoid excessive computation. This parameter should point to an array of four 8x8 blocks comprising the current block. The parameters `pVar`, `pMean` should point to arrays of four elements, since the values of variance and mean are calculated separately for each 8x8 block.

Mean is evaluated by the following formula:

$$M = \frac{\sum_{i=0}^7 \sum_{j=0}^7 \text{ref\_block}[i, j]}{8}$$

Variance is evaluated by the following formula:

$$V = \frac{\left( \sum_{i=0}^7 \sum_{j=0}^7 (\text{cur\_block}[i, j] - \text{ref\_block}[i, j])^2 \right) - M^2}{64}$$

This function is used in the MPEG-2 encoder included into Intel IPP Samples. See introduction to this section.

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when at least one input pointer is <code>NULL</code> .

### Block Variance Evaluation

This function evaluates variance of all elements in block.

## Variance16x16

*Evaluates variance of current block.*

### Syntax

```
IppStatus ippiVariance16x16_8u32s(const Ipp8u* pSrc, Ipp32s srcStep, Ipp32s* pVar);
```

### Parameters

<code>pSrc</code>	Pointer to the current block of specified size.
<code>srcStep</code>	Step of the current block, specifying width of the block in bytes.
<code>pVar</code>	Pointer to block variance.

### Description

This function is declared in the `ippvc.h` header file. The function `ippiVariance16x16_8u32s` evaluates variance of all the elements in the current block. The result is stored in integer `Var`.

Let us denote the pel which lies at the crossing of  $i^{\text{th}}$  row and  $j^{\text{th}}$  column of the current block as `block[i, j]`. Then

$$Var = \sum_{i=0}^{15} \sum_{j=0}^{15} block[i, j]^2 - \frac{\left( \sum_{i=0}^{15} \sum_{j=0}^{15} block[i, j] \right)^2}{256}.$$

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when at least one input pointer is <code>NULL</code> .
<code>ippStsStepErr</code>	Indicates an error when <code>srcStep</code> is less than or equal to zero.

## Evaluation of Block Deviation

These functions evaluate mean absolute deviation of a block.

## MeanAbsDev8x8

Evaluates mean absolute deviation for a 8x8 block.

### Syntax

```
IppStatus ippMeanAbsDev8x8_8u32s_C1R(const Ipp8u* pSrc, int srcStep, Ipp32s* pDst);
```

### Parameters

<code>pSrc</code>	Pointer to the block 8x8.
<code>srcStep</code>	Step of the block.
<code>pDst</code>	Pointer to the deviation.

### Description

The function `ippMeanAbsDev8x8_8u32s_C1R` is declared in the `ippvc.h` file. This function evaluates the mean absolute deviation for a 8x8 block by the following formula:

$$Dev = \sum_{i=0}^7 \sum_{j=0}^7 |block[i, j] - Mean|,$$

where *Mean* is evaluated by the formula

$$Mean = \frac{\sum_{i=0}^7 \sum_{j=0}^7 block[i, j] + 32}{64}.$$

This function is used in the H.261, H.263, and MPEG-4 encoders included into Intel IPP Samples. See [introduction](#) to this section.

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if at least one of the specified pointers is NULL.

## MeanAbsDev16x16

*Evaluates mean absolute deviation for a 16x16 block.*

### Syntax

```
IppStatus ippMeanAbsDev16x16_8u32s_C1R(const Ipp8u* pSrc, int srcStep, Ipp32s* pDst);
```

### Parameters

<code>pSrc</code>	Pointer to the block 16x16.
<code>srcStep</code>	Step of the block.
<code>pDst</code>	Pointer to the deviation.

### Description

The function `ippMeanAbsDev16x16_8u32s_C1R` is declared in the `ippvc.h` file. This function evaluates the mean absolute deviation for a 16x16 block by the following formula:

$$Dev = \sum_{i=0}^{15} \sum_{j=0}^{15} |block[i, j] - Mean|,$$

where *Mean* is evaluated by the formula

$$Mean = \frac{\sum_{i=0}^{15} \sum_{j=0}^{15} block[i, j] + 128}{256}.$$

This function is used in the H.261, H.263, and MPEG-4 encoders included into Intel IPP Samples. See [introduction](#) to this section.

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if at least one of the specified pointers is NULL.

## Edges Detection

This function detects edges inside block.

### EdgesDetect16x16

*Detects edges inside 16X16 block.*

---

#### Syntax

```
IppStatus ippEdgesDetect16x16_8u_C1R(const Ipp8u* pSrc, Ipp32u srcStep, Ipp8u
EdgePelDifference, Ipp8u EdgePelCount, Ipp8u* pRes);
```

#### Parameters

<i>pSrc</i>	Pointer to a 16x16 block in the current plan.
<i>srcStep</i>	Step of the current block, specifying width of the plane in bytes.
<i>EdgePelDifference</i>	The value for estimation of difference between neighboring elements. This value must be within the range of [0, 128].
<i>EdgePelCount</i>	The value for estimation of number of pairs of elements with 'big difference'. This value must be within the range of [0, 128].
<i>pRes</i>	Pointer to output value. ( <i>*pRes</i> ) is equal to 1, if edges are detected, and is equal to 0 if edges are not detected.

#### Description

This function is declared in the `ippvc.h` header file. The function `ippEdgesDetect16x16_8u_C1R` detects edges inside a 16x16 block: finds pair of neighboring (horizontal and vertical) elements with difference greater than *EdgePelDifference*.

If the number of pairs is greater than *EdgePelCount*, edges are detected and flag (*\*pRes*) is set to 1. Otherwise, edges are not detected (*\*pRes*) is set to 0).

```

res = 0  count = 0  row ∈ [0,14]  col ∈ [0,14]

if(Src[row][col] - Src[row][col+1]) > EdgePelDifference → count++

if(Src[row][col] - Src[row+1][col]) > EdgePelDifference → count++

if(count > EdgePelCount) → res = 1.
```

This function is used in the H.264 encoder included into Intel IPP Samples. See [introduction](#) to this section.

#### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when at least one input pointer is <code>NULL</code> .

## SAD Functions

These functions evaluate sum of absolute difference (SAD) between current and predicted blocks.

### SAD16x16

*Evaluates sum of absolute difference between current and reference 16X16 blocks.*

---

#### Syntax

```
IppStatus ippisAD16x16_8u32s(const Ipp8u* pSrc, Ipp32s srcStep, const Ipp8u* pRef, Ipp32s refStep, Ipp32s* pSAD, Ipp32s mcType);
```

```
IppStatus ippisAD16x16_16u32s_C1R(const Ipp16u* pSrc, Ipp32s srcStep, const Ipp16u* pRef, Ipp32s refStep, Ipp32s* pSAD, Ipp32s mcType);
```

#### Parameters

<i>pSrc</i>	Pointer to the current block of specified size.
<i>srcStep</i>	Distance in bytes between starts of the consecutive lines in the source image.
<i>pRef</i>	Pointer to the reference block of specified size.
<i>refStep</i>	Distance in bytes between starts of the consecutive lines in the reference image.
<i>pSAD</i>	Pointer to the SAD value.
<i>mcType</i>	MC type <a href="#">IPPVC_MC_APX</a> .

#### Description

This function is declared in the `ippvc.h` header file. The functions `ippisAD16x16_8u32s` and `ippisAD16x16_16u32s_C1R` evaluate the sum of absolute difference between all the elements in the current block and the corresponding elements in the reference block. The result is stored in *pSAD*.

This function is used in the MPEG-2 and H.264 encoders included into Intel IPP Samples. See [introduction](#) to this section.

#### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when at least one input pointer is <code>NULL</code> .
<code>ippStsStepErr</code>	Indicates an error when <i>srcCurStep</i> or <i>srcRefStep</i> is less than or equal to zero.

## SAD16x16xN, SAD16x16xNI

*Evaluates sum of absolute difference between current and reference 16x16xN and 16x16xNI blocks.*

### Syntax

```
IpplStatus ipplSAD16x16xN_8u16u_C1R(const Ipp8u* pSrc, Ipp32s srcStep, const Ipp8u* pRef,
IpplStatus ipplSAD16x16xNI_8u16u_C1R(const Ipp16u* pSrc, Ipp32s srcStep, const Ipp16u* pRef, Ipp32s refStep, Ipp16u* pSAD, Ipp32s numSAD);
```

```
IpplStatus ipplSAD16x16xNI_8u16u_C1R(const Ipp16u* pSrc, Ipp32s srcStep, const Ipp16u* pRef, Ipp32s refStep, Ipp16u* pSAD, Ipp32s numSAD, Ipp32u* pMinSADIndex);
```

### Parameters

<i>pSrc</i>	Pointer to the current block of specified size.
<i>srcStep</i>	Distance in bytes between starts of the consecutive lines in the source image.
<i>pRef</i>	Pointer to the reference block of specified size.
<i>refStep</i>	Distance in bytes between starts of the consecutive lines in the reference image.
<i>pSAD</i>	Pointer to an array of <i>numSAD</i> size to store calculated results.
<i>numSAD</i>	Number of SAD to calculate; should be a product of 8.
<i>pMinSADIndex</i>	Index of the minimal SAD value in <i>pSAD</i> array.

### Description

These functions are declared in the `ippvc.h` header file. The functions `ipplSAD16x16xN_8u16u_C1R` and `ipplSAD16x16xNI_8u16u_C1R` calculate a series of sums of absolute difference between all the elements in the current block and the corresponding elements in the reference block and store the results in the given array. The functions use the SSE4.1 instructions to enhance motion estimation.

#### Optimization Notice

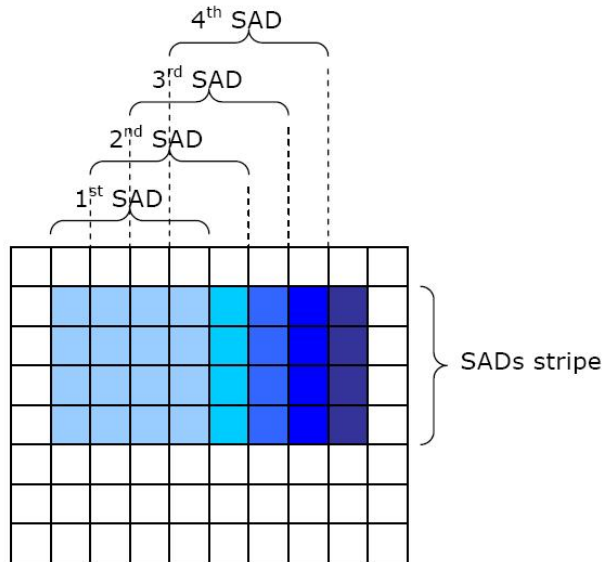
Intel's compilers may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include SSE2, SSE3, and SSSE3 instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on microprocessors not manufactured by Intel. Microprocessor-dependent optimizations in this product are intended for use with Intel microprocessors. Certain optimizations not specific to Intel microarchitecture are reserved for Intel microprocessors. Please refer to the applicable product User and Reference Guides for more information regarding the specific instruction sets covered by this notice.

Notice revision #20110804



See below for a figure that illustrates operation of the functions.

### Operation of `ippiSAD16x16xN_8u16u_C1R` and `ippiSAD16x16xNI_8u16u_C1R`



### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when at least one input pointer is <code>NULL</code> .
<code>ippStsStepErr</code>	Indicates an error if the number of SADs to produce is not a product of 8.

## SAD16x8

*Evaluates sum of absolute difference between current and reference 16X8 blocks.*

### Syntax

```
IppStatus ippiSAD16x8_8u32s_C1R(const Ipp8u* pSrcCur, int srcCurStep, const Ipp8u* pSrcRef,
int srcRefStep, Ipp32s* pSAD, Ipp32s mcType);
```

### Parameters

<code>pSrcCur</code>	Pointer to a 16x8 block in the source plane.
<code>srcCurStep</code>	Distance in bytes between starts of the consecutive lines in the source image.
<code>pSrcRef</code>	Pointer to a 16x8 block in the reference plane.
<code>srcRefStep</code>	Distance in bytes between starts of the consecutive lines in the reference image.

<i>pSAD</i>	Pointer to the SAD value.
<i>mcType</i>	MC type <a href="#">IPPVC_MC_APX</a> .

### Description

This function is declared in the `ippvc.h` header file. The function `ippiSAD16x8_8u32s_C1R` evaluates the sum of absolute difference of all the elements in the current 16x8 block and the corresponding elements in the reference 16x8 block. The result is stored in *pSAD*.

This function is used in the MPEG-4 encoder included into Intel IPP Samples. See [introduction](#) to this section.

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when at least one input pointer is <code>NULL</code> .

## SAD8x16

*Evaluates sum of absolute difference between current and reference 8X16 blocks.*

---

### Syntax

```
ippStatus ippiSAD8x16_8u32s_C1R(const Ipp8u* pSrcCur, int srcCurStep, const Ipp8u* pSrcRef,
int srcRefStep, Ipp32s* pSAD, Ipp32s mcType);
```

### Parameters

<i>pSrcCur</i>	Pointer to an 8x16 block in the source plane.
<i>srcCurStep</i>	Distance in bytes between starts of the consecutive lines in the source image.
<i>pSrcRef</i>	Pointer to an 8x16 block in the reference plane.
<i>srcRefStep</i>	Distance in bytes between starts of the consecutive lines in the reference image.
<i>pSAD</i>	Pointer to the SAD value.
<i>mcType</i>	MC type <a href="#">IPPVC_MC_APX</a> ; reserved and must be 0.

### Description

This function is declared in the `ippvc.h` header file. The function `ippiSAD8x16_8u32s_C1R` evaluates the sum of absolute difference of all the elements in the current 8x16 block and the corresponding elements in the reference 8x16 block. The result is stored in *pSAD*.

This function is used in the H.264 encoder included into Intel IPP Samples. See [introduction](#) to this section.

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when at least one input pointer is <code>NULL</code> .

## SAD8x8

*Evaluates sum of absolute difference between current and reference 8X8 blocks.*

---

### Syntax

```
IppStatus ippiSAD8x8_8u32s_C1R(const Ipp8u* pSrcCur, int srcCurStep, const Ipp8u* pSrcRef,
int srcRefStep, Ipp32s* pSAD, Ipp32s mcType);
```

```
IppStatus ippiSAD8x8_8u32s_C2R(const Ipp8u* pSrcCur, int srcCurStep, const Ipp8u* pSrcRef,
int srcRefStep, Ipp32s* pDstU, Ipp32s* pDstV, Ipp32s mcType);
```

```
IppStatus ippiSAD8x8_16u32s_C1R(const Ipp16u* pSrcCur, Ipp32s srcCurStep, const Ipp16u*
pSrcRef, Ipp32s srcRefStep, Ipp32s* pSAD, Ipp32s mcType);
```

### Parameters

<i>pSrcCur</i>	Pointer to an 8x8 block in the source plane
<i>srcCurStep</i>	Distance in bytes between starts of the consecutive lines in the source image
<i>pSrcRef</i>	Pointer to an 8x8 block in the reference plane
<i>srcRefStep</i>	Distance in bytes between starts of the consecutive lines in the reference image
<i>pDst</i>	Pointer to the SAD value
<i>pDstU</i>	Pointer to the calculated SAD value for U(Cb)
<i>pDstV</i>	Pointer to the calculated SAD value for V(Cr)
<i>mcType</i>	MC type <a href="#">IPPVC_MC_APX</a>

### Description

This function is declared in the `ippvc.h` header file. The functions `ippiSAD8x8_8u32s_C1R`, `ippiSAD8x8_8u32s_C2R`, and `ippiSAD8x8_16u32s_C1R` evaluate the sum of absolute difference of all the elements in the current 8x8 block and the corresponding elements in the reference 8x8 block. The result is stored in *pSAD*.

`ippiSAD8x8_8u32s_C2R` is a clone of `ippiSAD8x8_8u32s_C1R` but calculates SAD for U and V planes at once and for an NV12 chrominance plane:

NV12 Plane:

```
YY YY YY YY
YY YY YY YY
UV UV UV UV
```

These functions are used in the H.261, H.263, H.264 and MPEG-4 encoders included into Intel IPP Samples. See [introduction](#) to this section.

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when at least one input pointer is NULL.
<code>ippStsStepErr</code>	Indicates an error condition if the step value is negative.

## SAD8x8xN, SAD8x8xNI

*Evaluates sum of absolute difference between current and reference 8x8xN and 8x8xNI blocks.*

### Syntax

```
IppStatus ippisAD8x8xN_8u16u_C1R(const Ipp8u* pSrc, Ipp32s srcStep, const Ipp8u* pRef,
Ipp32s refStep, Ipp16u* pSAD, Ipp32s numSAD);
```

```
IppStatus ippisAD8x8xNI_8u16u_C1R(const Ipp16u* pSrc, Ipp32s srcStep, const Ipp16u* pRef,
Ipp32s refStep, Ipp16u* pSAD, Ipp32s numSAD, Ipp32u* pMinSADIndex);
```

### Parameters

<i>pSrc</i>	Pointer to the current block of specified size.
<i>srcStep</i>	Distance in bytes between starts of the consecutive lines in the source image.
<i>pRef</i>	Pointer to the reference block of specified size.
<i>refStep</i>	Distance in bytes between starts of the consecutive lines in the reference image.
<i>pSAD</i>	Pointer to an array of <i>numSAD</i> size to store calculated results.
<i>numSAD</i>	Number of SAD to calculate; should be a product of 8.
<i>pMinSADIndex</i>	Index of the minimal SAD value in <i>pSAD</i> array.

### Description

These functions are declared in the `ippvc.h` header file. The functions `ippisAD8x8xN_8u16u_C1R` and `ippisAD8x8xNI_8u16u_C1R` calculate a series of sums of absolute difference between all the elements in the current block and the corresponding elements in the reference block and store the results in the given array. The functions use the SSE4.1 instructions to enhance motion estimation.

#### Optimization Notice

Intel's compilers may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include SSE2, SSE3, and SSSE3 instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on microprocessors not manufactured by Intel. Microprocessor-dependent optimizations in this product are intended for use with Intel microprocessors. Certain optimizations not specific to Intel microarchitecture are reserved for Intel microprocessors. Please refer to the applicable product User and Reference Guides for more information regarding the specific instruction sets covered by this notice.

Notice revision #20110804

See [Figure "Operation of ippisAD16x16xN\\_8u16u\\_C1R and ippisAD16x16xNI\\_8u16u\\_C1R"](#) that illustrates operation of the functions similar to `ippisAD8x8xN_8u16u_C1R` and `ippisAD8x8xNI_8u16u_C1R`.

### Return Values

`ippStsNoErr` Indicates no error.

<code>ippStsNullPtrErr</code>	Indicates an error when at least one input pointer is <code>NULL</code> .
<code>ippStsStepErr</code>	Indicates an error if the number of SADs to produce is not a product of 8.

## SAD8x4

*Evaluates sum of absolute difference between current and reference 8X4 blocks.*

---

### Syntax

```
IppStatus ippisAD8x4_8u32s_C1R(const Ipp8u* pSrcCur, int srcCurStep, const Ipp8u* pSrcRef,
int srcRefStep, Ipp32s* pSAD, Ipp32s mcType);
```

### Parameters

<code>pSrcCur</code>	Pointer to an 8x4 block in the source plane.
<code>srcCurStep</code>	Distance in bytes between starts of the consecutive lines in the source image.
<code>pSrcRef</code>	Pointer to an 8x4 block in the reference plane.
<code>srcRefStep</code>	Distance in bytes between starts of the consecutive lines in the reference image.
<code>pDst</code>	Pointer to the SAD value.
<code>mcType</code>	MC type <code>IPPVC_MC_APX</code> ; reserved and must be 0.

### Description

This function is declared in the `ippvc.h` header file. The function `ippisAD8x4_8u32s_C1R` evaluates the sum of absolute difference of all the elements in the current 8x4 block and the corresponding elements in the reference 8x4 block. The result is stored in `pSAD`.

This function is used in the H.264 encoder included into Intel IPP Samples. See [introduction](#) to this section.

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when at least one input pointer is <code>NULL</code> .

## SAD4x8

*Evaluates sum of absolute difference between current and reference 4X8 blocks.*

---

### Syntax

```
IppStatus ippisAD4x8_8u32s_C1R(const Ipp8u* pSrcCur, int srcCurStep, const Ipp8u* pSrcRef,
int srcRefStep, Ipp32s* pSAD, Ipp32s mcType);
```

## Parameters

<i>pSrcCur</i>	Pointer to a 4x8 block in the source plane.
<i>srcCurStep</i>	Distance in bytes between starts of the consecutive lines in the source image.
<i>pSrcRef</i>	Pointer to a 4x8 block in the reference plane.
<i>srcRefStep</i>	Distance in bytes between starts of the consecutive lines in the reference image.
<i>pDst</i>	Pointer to the SAD value.
<i>mcType</i>	MC type <a href="#">IPPVC_MC_APX</a> ; reserved and must be 0.

## Description

This function is declared in the `ippvc.h` header file. The function `ippiSAD4x8_8u32s_C1R` evaluates the sum of absolute difference of all the elements in the current 4x8 block and the corresponding elements in the reference 4x8 block. The result is stored in *pSAD*.

This function is used in the H.264 encoder included into Intel IPP Samples. See [introduction](#) to this section.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when at least one input pointer is <code>NULL</code> .

## SAD4x4

*Evaluates sum of absolute difference between current and reference 4X4 blocks.*

---

## Syntax

```
ippStatus ippiSAD4x4_8u32s(const Ipp8u* pSrc, Ipp32s srcStep, const Ipp8u* pRef, Ipp32s refStep, Ipp32s* pSAD, Ipp32s mcType);
```

```
ippStatus ippiSAD4x4_16u32s_C1R(const Ipp16u* pSrc, Ipp32s srcStep, const Ipp16u* pRef, Ipp32s refStep, Ipp32s* pSAD, Ipp32s mcType);
```

## Parameters

<i>pSrc</i>	Pointer to a 4x4 block in the source plane.
<i>srcStep</i>	Distance in bytes between starts of the consecutive lines in the source image.
<i>pRef</i>	Pointer to a 4x4 block in the reference plane.
<i>refStep</i>	Distance in bytes between starts of the consecutive lines in the reference image.
<i>pSAD</i>	Pointer to the SAD value.
<i>mcType</i>	MC type <a href="#">IPPVC_MC_APX</a> ; reserved and must be 0.

### Description

This function is declared in the `ippvc.h` header file. The functions `ippiSAD4x4_8u32s` and `ippiSAD4x4_16u32s_C1R` evaluate the sum of absolute difference of all the elements in the current 4x4 block and the corresponding elements in the reference 4x4 block. The result is stored in `pSAD`.

This function is used in the H.264 encoder included into Intel IPP Samples. See [introduction](#) to this section.

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when at least one input pointer is <code>NULL</code> .

## SAD4x4xN, SAD4x4xNI

*Evaluates sum of absolute difference between current and reference 4x4xN and 4x4xNI blocks.*

### Syntax

```
ippStatus ippiSAD4x4xN_8u16u_C1R(const Ipp8u* pSrc, Ipp32s srcStep, const Ipp8u* pRef,
Ipp32s refStep, Ipp16u* pSAD, Ipp32s numSAD);

ippStatus ippiSAD4x4xNI_8u16u_C1R(const Ipp16u* pSrc, Ipp32s srcStep, const Ipp16u* pRef,
Ipp32s refStep, Ipp16u* pSAD, Ipp32s numSAD, Ipp32u* pMinSADIndex);
```

### Parameters

<code>pSrc</code>	Pointer to the current block of specified size.
<code>srcStep</code>	Distance in bytes between starts of the consecutive lines in the source image.
<code>pRef</code>	Pointer to the reference block of specified size.
<code>refStep</code>	Distance in bytes between starts of the consecutive lines in the reference image.
<code>pSAD</code>	Pointer to an array of <code>numSAD</code> size to store calculated results.
<code>numSAD</code>	Number of SAD to calculate; should be a product of 8.
<code>pMinSADIndex</code>	Index of the minimal SAD value in <code>pSAD</code> array.

### Description

These functions are declared in the `ippvc.h` header file. The functions `ippiSAD4x4xN_8u16u_C1R` and `ippiSAD4x4xNI_8u16u_C1R` calculate a series of sums of absolute difference between all the elements in the current block and the corresponding elements in the reference block and store the results in the given array. The functions use the SSE4.1 instructions to enhance motion estimation.

Optimization Notice

Intel's compilers may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include SSE2, SSE3, and SSSE3 instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any

**Optimization Notice**

optimization on microprocessors not manufactured by Intel. Microprocessor-dependent optimizations in this product are intended for use with Intel microprocessors. Certain optimizations not specific to Intel microarchitecture are reserved for Intel microprocessors. Please refer to the applicable product User and Reference Guides for more information regarding the specific instruction sets covered by this notice.

Notice revision #20110804

See [Figure "Operation of `ippiSAD16x16xN\_8u16u\_C1R` and `ippiSAD16x16xNI\_8u16u\_C1R`"](#) that illustrates operation of the functions similar to `ippiSAD4x4xN_8u16u_C1R` and `ippiSAD4x4xNI_8u16u_C1R`.

**Return Values**

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when at least one input pointer is <code>NULL</code> .
<code>ippStsStepErr</code>	Indicates an error if the number of SADs to produce is not a product of 8.

**SAD16x16Blocks8x8**

*Evaluates four partial sums of absolute differences between current and reference 16X16 blocks.*

**Syntax**

```
ippStatus ippiSAD16x16Blocks8x8_8u16u(const Ipp8u* pSrc, Ipp32s srcStep, const Ipp8u* pRef, Ipp32s refStep, Ipp16u* pDstSAD, Ipp32s mcType);
```

**Parameters**

<code>pSrc</code>	Pointer to a 16x16 block in the source plane.
<code>srcStep</code>	Distance in bytes between starts of the consecutive lines in the source image.
<code>pRef</code>	Pointer to a 16x16 block in the reference plane.
<code>refStep</code>	Distance in bytes between starts of the consecutive lines in the reference image.
<code>pDstSAD</code>	Pointer to an array of size 4 to store the SAD values.
<code>mcType</code>	Reserved and must be 0.

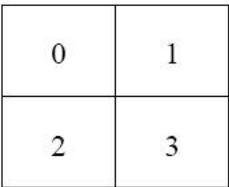
**Description**

This function is declared in the `ippvc.h` header file.



The function `ippiSAD16x16Blocks8x8_8u16u` evaluates the four partial sums of absolute differences of all the elements in the current 16x16 block and the corresponding elements in the reference 16x16 block. The result is stored in `pDstSAD`.

Splitting of 16x16 Block Into Four 8x8 Blocks



This function is used in the H.264 encoder included into Intel IPP Samples. See [introduction](#) to this section.

Return Values

- `ippStsNoErr`
- Indicates no error.
- `ippStsNullPtrErr`
- Indicates an error when at least one input pointer is `NULL`.

SAD16x16Blocks4x4

Evaluates 16 partial sums of absolute differences between current and reference 16X16 blocks.

Syntax

```
IppStatus ippiSAD16x16Blocks4x4_8u16u(const Ipp8u* pSrc, Ipp32s srcStep, Ipp8u* pRef, Ipp32s refStep, Ipp16u* pDstSAD, Ipp32s mcType);
```

Parameters

- `pSrc`
- Pointer to a 16x16 block in the source plane.
- `srcStep`
- Distance in bytes between starts of the consecutive lines in the source image.
- `pRef`
- Pointer to a 16x16 block in the reference plane.
- `refStep`
- Distance in bytes between starts of the consecutive lines in the reference image.
- `pDstSAD`
- Pointer to an array of size 16 to store the SAD values.
- `mcType`
- Reserved and must be 0.

Description

This function is declared in the `ippvc.h` header file.

The function `ippiSAD16x16Blocks4x4_8u16u` evaluates 16 partial sums (for each 4x4 block in the order shown in [Figure "Splitting of 16x16 Block Into 4x4 Blocks"](#)) of absolute differences between all the elements in the current 16x16 block and the corresponding elements in the reference 16x16 block. The result is stored in `pDstSAD`.

## Splitting of 16x16 Block Into 4x4 Blocks

---

0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when at least one input pointer is NULL.

## SATD16x16

*Evaluates sum of absolute transformed differences between current and reference 16X16 blocks using 4x4 transform.*

---

## Syntax

```
ippStatus ippiSATD16x16_8u32s_C1R(const Ipp8u* pSrcCur, int srcCurStep, const Ipp8u* pSrcRef, int srcRefStep, Ipp32s* pSATD);
```

## Parameters

<code>pSrcCur</code>	Pointer to a 16x16 block in the source plane.
<code>srcCurStep</code>	Distance in bytes between starts of the consecutive lines in the source image.
<code>pSrcRef</code>	Pointer to a 16x16 block in the reference plane.
<code>srcRefStep</code>	Distance in bytes between starts of the consecutive lines in the reference image.
<code>pSATD</code>	Pointer to the SATD value.

## Description

This function is declared in the `ippvc.h` header file. The functions `ippiSATD16x16_8u32s_C1R` and `ippiSATD16x16_16u32s_C1R` evaluate the sum of absolute transformed differences of all the elements in the current 16x16 block and the corresponding elements in the reference 16x16 block using a 4x4 transform. The transform matrix is as follows:

$$T = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \\ 1 & -1 & 1 & -1 \end{pmatrix}$$

This matrix is applied to each 4x4 subblock of the 16x16 block.

If  $D$  is a 16x16 block of difference, the result of the function is the sum of absolute values of elements of  $T * D * T$  for each subblock. The result is stored in  $pSATD$ .

This function is used in the H.264 encoder included into Intel IPP Samples. See [introduction](#) to this section.

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when at least one input pointer is NULL.

## SATD16x8

*Evaluates sum of absolute transformed differences between current and reference 16X8 blocks using 4x4 transform.*

### Syntax

```
ippStatus ippISATD16x8_8u32s_C1R(const Ipp8u* pSrcCur, int srcCurStep, const Ipp8u* pSrcRef,
int srcRefStep, Ipp32s* pSATD);
```

### Parameters

<code>pSrcCur</code>	Pointer to a 16x8 block in the source plane.
<code>srcCurStep</code>	Distance in bytes between starts of the consecutive lines in the source image.
<code>pSrcRef</code>	Pointer to a 16x8 block in the reference plane.
<code>srcRefStep</code>	Distance in bytes between starts of the consecutive lines in the reference image.
<code>pSATD</code>	Pointer to the SATD value.

### Description

This function is declared in the `ippvc.h` header file. The functions `ippISATD16x8_8u32s` and `ip-  
piSATD16x8_16u32s_C1R` evaluate the sum of absolute tranformed differences of all the elements in the current 16x8 block and the corresponding elements in the reference 16x8 block using a 4x4 transform. The transform matrix is as follows:

$$T = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \\ 1 & -1 & 1 & -1 \end{pmatrix}$$

This matrix is applied to each 4x4 subblock of the 16x8 block.

If  $D$  is a 16x8 block of difference, the result of the function is the sum of absolute values of elements of  $T * D * T$  for each subblock. The result is stored in  $pSATD$ .

This function is used in the H.264 encoder included into Intel IPP Samples. See [introduction](#) to this section.

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when at least one input pointer is NULL.

## SATD8x16

*Evaluates sum of absolute transformed differences between current and reference 8X16 blocks using 4x4 transform.*

---

### Syntax

```
ippStatus ippisATD8x16_8u32s_C1R(const Ipp8u* pSrcCur, int srcCurStep, const Ipp8u* pSrcRef,
int srcRefStep, Ipp32s* pSATD);
```

### Parameters

<code>pSrcCur</code>	Pointer to an 8x16 block in the source plane.
<code>srcCurStep</code>	Distance in bytes between starts of the consecutive lines in the source image.
<code>pSrcRef</code>	Pointer to an 8x16 block in the reference plane.
<code>srcRefStep</code>	Distance in bytes between starts of the consecutive lines in the reference image.
<code>pSATD</code>	Pointer to the SATD value.

### Description

This function is declared in the `ippvc.h` header file. The functions `ippisATD8x16_8u32s` and `ippisATD8x16_16u32s_C1R` evaluate the sum of absolute transformed differences of all the elements in the current 8x16 block and the corresponding elements in the reference 8x16 block using a 4x4 transform. The transform matrix is as follows:

$$T = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \\ 1 & -1 & 1 & -1 \end{pmatrix}$$

This matrix is applied to each 4x4 subblock of the 8x16 block.

If  $D$  is a 8x16 block of difference, the result of the function is the sum of absolute values of elements of  $T * D * T$  for each subblock. The result is stored in  $pSATD$ .

This function is used in the H.264 encoder included into Intel IPP Samples. See [introduction](#) to this section.

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when at least one input pointer is NULL.

## SATD8x8

*Evaluates sum of absolute transformed differences between current and reference 8X8 blocks using 4x4 transform.*

### Syntax

```
IppStatus ippISATD8x8_8u32s_C1R(const Ipp8u* pSrcCur, int srcCurStep, const Ipp8u* pSrcRef,
int srcRefStep, Ipp32s* pSATD);
```

### Parameters

<code>pSrcCur</code>	Pointer to an 8x8 block in the source plane.
<code>srcCurStep</code>	Distance in bytes between starts of the consecutive lines in the source image.
<code>pSrcRef</code>	Pointer to an 8x8 block in the reference plane.
<code>srcRefStep</code>	Distance in bytes between starts of the consecutive lines in the reference image.
<code>pSATD</code>	Pointer to the SATD value.

### Description

This function is declared in the `ippvc.h` header file. The functions `ippISATD8x8_8u32s` and `ip-  
piSATD8x8_16u32s_C1R` evaluate the sum of absolute transformed differences of all the elements in the current 8x8 block and the corresponding elements in the reference 8x8 block using a 4x4 transform. The transform matrix is as follows:

$$T = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \\ 1 & -1 & 1 & -1 \end{pmatrix}$$

This matrix is applied to each 4x4 subblock of the 8x8 block.

If  $D$  is a 8x8 block of difference, the result of the function is the sum of absolute values of elements of  $T * D * T$  for each subblock. The result is stored in  $pSATD$ .

This function is used in the H.264 encoder included into Intel IPP Samples. See [introduction](#) to this section.

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when at least one input pointer is NULL.

## SATD8x4

*Evaluates sum of absolute transformed differences between current and reference 8X4 blocks using 4x4 transform.*

---

### Syntax

```
IppStatus ippiSATD8x4_8u32s_C1R(const Ipp8u* pSrcCur, int srcCurStep, const Ipp8u* pSrcRef,
int srcRefStep, Ipp32s* pSATD);
```

### Parameters

<code>pSrcCur</code>	Pointer to an 8x4 block in the source plane.
<code>srcCurStep</code>	Distance in bytes between starts of the consecutive lines in the source image.
<code>pSrcRef</code>	Pointer to an 8x4 block in the reference plane.
<code>srcRefStep</code>	Distance in bytes between starts of the consecutive lines in the reference image.
<code>pSATD</code>	Pointer to the SATD value.

### Description

This function is declared in the `ippvc.h` header file. The functions `ippiSATD8x4_8u32s` and `ippiSATD8x4_16u32s_C1R` evaluate the sum of absolute transformed differences of all the elements in the current 8x4 block and the corresponding elements in the reference 8x4 block using a 4x4 transform. The transform matrix is as follows:

$$T = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \\ 1 & -1 & 1 & -1 \end{pmatrix}$$

This matrix is applied to each 4x4 subblock of the 8x4 block.

If  $D$  is a 8x4 block of difference, the result of the function is the sum of absolute values of elements of  $T * D * T$  for each subblock. The result is stored in  $pSATD$ .

This function is used in the H.264 encoder included into Intel IPP Samples. See [introduction](#) to this section.

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when at least one input pointer is NULL.

## SATD4x8

*Evaluates sum of absolute transformed differences between current and reference 4X8 blocks using 4x4 transform.*

### Syntax

```
IppStatus ippiSATD4x8_8u32s_C1R(const Ipp8u* pSrcCur, int srcCurStep, const Ipp8u* pSrcRef,
int srcRefStep, Ipp32s* pSATD);
```

### Parameters

<code>pSrcCur</code>	Pointer to a 4x8 block in the source plane.
<code>srcCurStep</code>	Distance in bytes between starts of the consecutive lines in the source image.
<code>pSrcRef</code>	Pointer to a 4x8 block in the reference plane.
<code>srcRefStep</code>	Distance in bytes between starts of the consecutive lines in the reference image.
<code>pSATD</code>	Pointer to the SATD value.

### Description

This function is declared in the `ippvc.h` header file. The functions `ippiSATD4x4_8u32s` and `ippiSATD4x4_16u32s_C1R` evaluate the sum of absolute transformed differences of all the elements in the current 4x8 block and the corresponding elements in the reference 4x8 block using a 4x4 transform. The transform matrix is as follows:

$$T = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \\ 1 & -1 & 1 & -1 \end{pmatrix}$$

This matrix is applied to each 4x4 subblock of the 8x4 block.

If  $D$  is a 4x8 block of difference, the result of the function is the sum of absolute values of elements of  $T * D * T$  for each subblock. The result is stored in  $pSATD$ .

This function is used in the H.264 encoder included into Intel IPP Samples. See [introduction](#) to this section.

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when at least one input pointer is <code>NULL</code> .

## SATD4x4

*Evaluates sum of absolute transformed differences between current and reference 4X4 blocks.*

---

### Syntax

```
IppStatus ippISATD4x4_8u32s_C1R(const Ipp8u* pSrcCur, int srcCurStep, const Ipp8u* pSrcRef,
int srcRefStep, Ipp32s* pSATD);
```

### Parameters

<code>pSrcCur</code>	Pointer to a 4x4 block in the source plane.
<code>srcCurStep</code>	Distance in bytes between starts of the consecutive lines in the source image.
<code>pSrcRef</code>	Pointer to a 4x4 block in the reference plane.
<code>srcRefStep</code>	Distance in bytes between starts of the consecutive lines in the reference image.
<code>pSATD</code>	Pointer to the SATD value.

### Description

This function is declared in the `ippvc.h` header file. The functions `ippISATD4x4_8u32s` and `ippISATD4x4_16u32s_C1R` evaluate the sum of absolute transformed differences of all the elements in the current 4x4 block and the corresponding elements in the reference 4x4 block. The transform matrix is as follows:

$$T = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \\ 1 & -1 & 1 & -1 \end{pmatrix}$$



If  $D$  is a 4x4 block of difference, the result of the function is the sum of absolute values of elements of  $T * D * T$ . The result is stored in  $pSATD$ .

This function is used in the H.264 encoder included into Intel IPP Samples. See [introduction](#) to this section.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when at least one input pointer is NULL.

## SAT8x8D

*Evaluates sum of absolute transformed differences between current and reference 8X8 blocks using 8x8 transform.*

---

## Syntax

```
ippStatus ippiSAT8x8D_8u32s_C1R(const Ipp8u* pSrcCur, int srcCurStep, const Ipp8u* pSrcRef,
int srcRefStep, Ipp32s* pSATD);
```

```
ippStatus ippiSAT8x8D_16u32s_C1R(const Ipp16u* pSrcCur, int srcCurStep, const Ipp16u*
pSrcRef, int srcRefStep, Ipp32s* pSATD);
```

## Parameters

<code>pSrcCur</code>	Pointer to an 8x8 block in the source plane.
<code>srcCurStep</code>	Distance in bytes between starts of the consecutive lines in the source image.
<code>pSrcRef</code>	Pointer to an 8x8 block in the reference plane.
<code>srcRefStep</code>	Distance in bytes between starts of the consecutive lines in the reference image.
<code>pSATD</code>	Pointer to the SATD value.

## Description

This function is declared in the `ippvc.h` header file. The functions `ippiSAT8x8D_8u32s_C1R` and `ippiSAT8x8D_16u32s_C1R` evaluate the sum of absolute transformed differences of all the elements in the current 8x8 block and the corresponding elements in the reference 8x8 block using an 8x8 transform. The transform matrix is as follows:

$$T = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 & 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 & 1 & -1 & -1 & 1 \\ 1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 \\ 1 & -1 & 1 & -1 & -1 & 1 & -1 & 1 \\ 1 & 1 & -1 & -1 & -1 & -1 & 1 & 1 \\ 1 & -1 & -1 & 1 & -1 & 1 & 1 & -1 \end{pmatrix}$$

If  $D$  is a 8x8 block of difference, the result of the function is the sum of absolute values of elements of  $T * D * T$ . The result is stored in  $pSATD$ .

This function is used in the H.264 encoder included into Intel IPP Samples. See [introduction](#) to this section.

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when at least one input pointer is NULL.

## FrameFieldSAD16x16

*Calculates SAD between field lines and SAD between frame lines of block 16x16.*

---

### Syntax

```

IppStatus ippiFrameFieldSAD16x16_8u32s_C1R(const Ipp8u* pSrc, int srcStep, Ipp32s*
pFrameSAD, Ipp32s* pFieldSAD);

IppStatus ippiFrameFieldSAD16x16_16s32s_C1R(const Ipp16s* pSrc, int srcStep, Ipp32s*
pFrameSAD, Ipp32s* pFieldSAD);

```

### Parameters

<code>pSrc</code>	Pointer to the 16x16 block.
<code>srcStep</code>	Distance in bytes between starts of the consecutive lines in the source image.
<code>pFrameSAD</code>	Pointer to the resulting SAD between frame lines of the block.
<code>pFieldSAD</code>	Pointer to the resulting SAD between field lines of the block.

### Description

This function is declared in the `ippvc.h` header file. The functions `ippiFrameFieldSAD16x16_8u32s_C1R` and `ippiFrameFieldSAD16x16_16s32s_C1R` calculate SAD between field lines and between frame lines of 16x16 blocks. The functions are used for decision on DCT type (Frame or Field) in encoding process of interlaced video.

The computation formulas are as follows:

$$\begin{aligned}
 pFrameSAD &= \sum_{i=0}^6 \sum_{j=0}^{15} |pSrc_{i,j} - pSrc_{i+1,j}| + \sum_{i=8}^{14} \sum_{j=0}^{15} |pSrc_{i,j} - pSrc_{i+1,j}| \\
 pFieldSAD &= \sum_{i=0}^6 \sum_{j=0}^{15} |pSrc_{i*2,j} - pSrc_{i*2+2,j}| + \\
 &+ \sum_{i=0}^6 \sum_{j=0}^{15} |pSrc_{i*2+1,j} - pSrc_{i*2+3,j}|
 \end{aligned}$$

The functions `ippiFrameFieldSAD16x16_8u32s_C1R` and `ippiFrameFieldSAD16x16_16s32s_C1R` are used in the MPEG-4 encoder included into Intel IPP Samples. See [introduction](#) to this section.

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when at least one input pointer is NULL.

### Sum of Differences Evaluation

These functions evaluate difference between current and predicted blocks and calculates sums of elements of all residual blocks.

## SumsDiff16x16Blocks4x4

*Evaluates difference between current and reference 4X4 blocks and calculates sums of 4X4 residual blocks elements for 16X16 blocks.*

### Syntax

```
ippStatus ippiSumsDiff16x16Blocks4x4_8u16s_C1(const Ipp8u* pSrc, Ipp32s srcStep, const Ipp8u* pPred, Ipp32s predStep, Ipp16s* pSums, Ipp16s* pDiff);
```

### Parameters

<code>pSrc</code>	Pointer to a 16x16 block in the current plane.
<code>srcStep</code>	Distance in bytes between starts of the consecutive lines in the source image.
<code>pPred</code>	Pointer to a 16x16 block in the reference plane.
<code>predStep</code>	Distance in bytes between starts of the consecutive lines in the reference image.
<code>pSums</code>	Pointer to an array of size 16 that contains sums of 4x4 difference blocks coefficients. The array is filled by function.
<code>pDiff</code>	If not NULL, pointer to an array of size 256 that contains a sequence of 4x4 residual blocks. The array is filled by function.

## Description

This function is declared in the `ippvc.h` header file. The function `ippiSumsDiff16x16Blocks4x4_8u16s_C1` evaluates difference between current and reference 4x4 blocks and calculates sums of 4x4 residual blocks elements in same order as shown in the figure below.

## Splitting of 16x16 Block Into Sixteen 4x4 Blocks ---

0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15

---


$$pDiff[(k*4 + l)*16 + i*4 + j] =$$

$$pSrc[(k*4 + i)*srcStep + (l*4 + j)] - pPred[(k*4 + i)*predStep + (l*4 + j)]$$

$$k, l, i, j \in [0, 3]$$

$$pSums[n] = \sum_{i=0}^{15} pDiff[n*16 + i]$$

$$n \in [0, 15].$$

## Return Values

`ippStsNoErr` Indicates no error.

`ippStsNullPtrErr` Indicates an error when at least one input pointer is NULL.

## SumsDiff8x8Blocks4x4

*Evaluates difference between current and reference 4X4 blocks and calculates sums of 4X4 residual blocks elements for 8X8 blocks.*

---

## Syntax

```
ippStatus ippiSumsDiff8x8Blocks4x4_8u16s_C1(const Ipp8u* pSrc, Ipp32s srcStep, const Ipp8u*
pPred, Ipp32s predStep, Ipp16s* pSums, Ipp16s* pDiff);
```

```
IppStatus ippiSumsDiff8x8Blocks4x4_8u16s_C2P2(const Ipp8u* pSrcUV, Ipp32s srcStep, const
Ipp8u* pPredU, Ipp32s predStepU, const Ipp8u* pPredV, Ipp32s predStepV, Ipp16s* pSumsU,
Ipp16s* pDiffU, Ipp16s* pSumsV, Ipp16s* pDiffV);
```

## Parameters

<i>pSrc</i>	Pointer to an 8x8 block in the current plane
<i>pSrcUV</i>	Pointer to the source block (chrominance part of NV12 plane) <pre> 0  UV UV UV UV UV UV UV UV 1  UV UV UV UV UV UV UV UV ... 7  UV UV UV UV UV UV UV UV </pre>
<i>srcStep</i>	Distance in bytes between starts of the consecutive lines in the source image
<i>pPred</i>	Pointer to an 8x8 block in the reference plane
<i>predStep</i>	Distance in bytes between starts of the consecutive lines in the reference image
<i>pPredU</i>	Pointer to the reference U block
<i>predStep</i>	Distance in bytes between starts of the consecutive lines in the reference U plane
<i>pPred</i>	Pointer to the reference V block
<i>predStep</i>	Distance in bytes between starts of the consecutive lines in the reference V plane
<i>pSums</i>	Pointer to an array of size 4 that contains sums of 4x4 difference blocks coefficients. The array is filled by function.
<i>pDiff</i>	If not NULL, pointer to an array of size 64 that contains a sequence of 4x4 residual blocks. The array is filled by function
<i>pSumsU</i>	Pointer to an array that contains sums of 4x4 difference blocks coefficients. The array is filled by function.
<i>pDiffU</i>	If not 0, pointer to an array that contains a sequence of 4x4 residual blocks. The array is filled by function if <i>pDiffU</i> is not NULL
<i>pSumsV</i>	Pointer to an array that contains sums of 4x4 difference blocks coefficients. The array is filled by function.
<i>pDiffV</i>	If not 0, pointer to an array that contains a sequence of 4x4 residual blocks. The array is filled by function if <i>pDiffV</i> is not NULL

## Description

The functions are declared in the `ippvc.h` header file. These functions evaluate difference between current and reference 4x4 blocks and calculates sums of 4x4 residual blocks elements in same order as shown in the figure below:

### Splitting of 8x8 Block Into Sixteen 4x4 Blocks ---

0	1
2	3

---

$$\begin{aligned}
 pDiff[(k*2+1)*16+i*4+j] = \\
 pSrc[(k*4+i)*srcStep+(1*4+j)] - pPred[(k*4+i)*predStep+(1*4+j)] \\
 i, j \in [0,3] \\
 k, l \in [0,1] \\
 \\
 pSums[n] = \sum_{i=0}^{15} pDiff[n*16+i] \\
 n \in [0,3].
 \end{aligned}$$

`ippiSumsDiff8x8Blocks4x4_8u16s_C2P2` is a clone of `ippiSumsDiff8x8Blocks4x4_8u16s_C1` but the source image is a chrominance part of an NV12 plane:

```
YY YY YY YY
YY YY YY YY
UV UV UV UV - chrominance part of NV12 plane.
```

This function is used in the H.264 encoder included into Intel IPP Samples. See [introduction](#) to this section.

### ippiSumsDiff8x8Blocks4x4\_8u16s\_C2P2R Usage Example

```
{
    // the pointer to the source block (8x8 block in the current plane)
    // in the chrominance part of NV12 plane).
    Ipp8u *pSrcUV = current_block_in_frame;
    // Distance in bytes between starts of the consecutive lines
    // in the source image
    Ipp32s srcStep = src_step;
    // Pointer to the reference U 8x8 block
    const Ipp8u* pPredU = block_u;
    // Step of the reference U plane, specifying width of the plane in bytes.
    Ipp32s predStepU = step_u;
    // Pointer to the reference V 8x8 block
```

```

const Ipp8u* pPredV = block_v;
Ipp32s predStepV = step_v;
// Pointer to the array that contains sums of U 4x4 difference blocks coefficients.
// The array is filled by the function.
Ipp16s* pSumsU = sums_of_u;
// If it is not zero, pointer to the array that contains a sequence of 4x4 U residual blocks.
// The array is filled by function if pDifUf is not null
Ipp16s* pDiffU = diff_u;
// Pointer to the array that contains sums of V 4x4 difference blocks coefficients.
// The array is filled by the function
Ipp16s* pSumsV = sums_of_v;
// If it is not zero, pointer to array that contains a sequence of 4x4 U residual blocks.
// The array is filled by the function if pDifUf is not null
Ipp16s* pDiffV = diff_v;
ippiSumsDiff8x8Blocks4x4_8u16s_C2P2(pSrcUV,srcStep,
                                     pPredU,PredPitchU,
                                     pPredV,PredPitchV,
                                     pSumsU,pDiffU,
                                     pSumsV,pDiffV);
}

```

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when at least one input pointer is <code>NULL</code> .

## Scanning Functions

Scanning functions convert a two-dimensional 8x8 array into one-dimensional data and vice versa using the predefined scan pattern.

## ScanInv

*Performs classical zigzag, alternate-horizontal, or alternate-vertical inverse scan on a block stored in a compact buffer.*

## Syntax

```
IppStatus ippiScanInv_16s_C1(Ipp16s* pSrc, Ipp16s* pDst, int indxLastNonZero, int scan);
```

## Parameters

<code>pSrc</code>	Pointer to the input block (coefficients in the scan order).
<code>pDst</code>	Pointer to the output block (coefficients in the normal order). The output is in a full buffer that contains 64 elements.
<code>indxLastNonZero</code>	Index of the last non-zero coefficient. Valid within the range of 0 to 63. This parameter provides faster operation. If the value is unknown, set to 63.
<code>scan</code>	Type of the scan, takes one of the following values: <code>IPPVC_SCAN_ZIGZAG</code> , indicating the classical zigzag scan, <code>IPPVC_SCAN_HORIZONTAL</code> , indicating the alternate-horizontal scan,

IPPVC\_SCAN\_VERTICAL, indicating the alternate-vertical scan.

See the corresponding enumerator in the "[Structures and Enumerators](#)" section.

## Description

The function `ippiScanInv_16s_C1` is declared in the `ippvc.h` header file. This function converts a block of coefficients placed in classical zigzag, alternate-horizontal, or alternate-vertical scan order to a block of coefficients placed in the conventional - left-to-right, top-to-bottom raster scan - order. The zigzag and the two alternate scan patterns are shown, for example, in [ [ITUH263](#) ], Figure14 and [ [ITUH263](#) ], Annex I, Figure I.2.

This function is used in the H.263 decoder included into Intel IPP Samples downloadable from <http://www.intel.com/cd/software/products/asmo-na/eng/220046.htm> .

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if at least one of the specified pointers is NULL.
<code>ippStsOutOfRangeErr</code>	Indicates an error condition if <code>indxLastNonZero</code> is out of the range [0, 63].

## ScanFwd

*Performs classical zigzag, alternate-horizontal, or alternate-vertical forward scan on a block.*

---

## Syntax

```
IppStatus ippiScanFwd_16s_C1(Ipp16s* pSrc, Ipp16s* pDst, int countNonZero, int scan);
```

## Parameters

<code>pSrc</code>	Pointer to the input block (coefficients in the normal order).
<code>pDst</code>	Pointer to the output block (coefficients in the scan order).
<code>countNonZero</code>	Number of non-zero coefficients in the block. Valid within the range of 1 to 64. This parameter provides faster operation. If the value is unknown, set to 64.
<code>scan</code>	Type of the scan, takes one of the following values: IPPVC_SCAN_ZIGZAG, indicating the classical zigzag scan, IPPVC_SCAN_HORIZONTAL, indicating the alternate-horizontal scan, IPPVC_SCAN_VERTICAL, indicating the alternate-vertical scan. See the corresponding enumerator in the " <a href="#">Structures and Enumerators</a> " section.



## Description

The function `ippiScanFwd_16s_C1` is declared in the `ippvc.h` header file. This function converts a block of coefficients placed in the conventional - left-to-right, top-to-bottom raster scan - order to a block of coefficients placed in classical zigzag, alternate-horizontal, or alternate-vertical scan order. See [Figure "Scanning Matrix and Sequence"](#) for the scanning process in line with the classical zigzag pattern. The zigzag and the two alternate scan patterns are shown, for example, in [ [ITUH263](#) ], Figure14 and [ [ITUH263](#) ], Annex I, Figure I.2.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if at least one of the specified pointers is NULL.
<code>ippStsOutOfRangeErr</code>	Indicates an error condition if <code>countNonZero</code> is out of the range [1, 64].

## Color Conversion

### CbYCr422ToYCbCr420\_Rotate

*Converts 4:2:2 CbYCr image to 4:2:0 YCbCr image with rotation.*

## Syntax

```
ippStatus ippiCbYCr422ToYCbCr420_Rotate_8u_C2P3R(const Ipp8u* pSrc, int srcStep, IppiSize srcRoi, Ipp8u* pDst[3], int dstStep[3], int rotation);
```

```
ippStatus ippiCbYCr422ToYCbCr420_Rotate_8u_P3R(const Ipp8u* pSrc[3], int srcStep[3], IppiSize srcRoi, Ipp8u* pDst[3], int dstStep[3], int rotation);
```

## Parameters

<code>pSrc</code>	Pointer to the source image for pixel-order data. Array of pointers to the separate source image planes for planar data.
<code>srcStep</code>	Distance in bytes between starts of the consecutive lines in the source image. Array of such values in bytes through the source image.
<code>srcRoi</code>	ROI of the source image, in pixels. The ROI of the destination image is calculated by user.
<code>pDst</code>	Array of pointers to the destination image planes.
<code>dstStep</code>	Distance in bytes between starts of the consecutive lines in the destination image. Array of such values through the destination image planes .
<code>rotation</code>	Rotation control parameter. Possible values: <code>IPPVC_ROTATE_DISABLE</code> no rotation <code>IPPVC_ROTATE_90CCW</code> rotating by 90° counterclockwise <code>IPPVC_ROTATE_90CW</code> rotating by 90° clockwise <code>IPPVC_ROTATE_180</code> rotating by 180°

See the corresponding [enumerator](#) in the introduction to the General Functions section.

## Description

The function `ippiCbYCr422ToYCbCr420_Rotate` is declared in the `ippvc.h` file. This function converts 4:2:2 two-channel or three-plane CbYCr image `pSrc` to the 4:2:0 YCbCr three-plane image `pDst`. The two-channel source image has the following sequence of samples: `Cb0, Y0, Cr0, Y1, Cb1, Y2, Cr1, Y3, Cb2, ...`. Three-plane source image has the following order of pointers: `Cb-plane, Y-plane, Cr-plane`. The destination image has the following order of pointers: `Y-plane, Cb-plane, Cr-plane` (see [Table "Pixel-Order Image Formats"](#) and [Table "Planar Image Formats"](#) ).

The function additionally rotates or flips an image in accordance with the value of the parameter `rotation`.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if any of the specified pointers is <code>null</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if any field of the <code>srcRoi</code> is less than 2.
<code>ippStsDoubleSize</code>	Indicates the condition when the values <code>srcRoi.width</code> and <code>srcRoi.height</code> are not multiples of 2. The function reduces their original values to the nearest multiples of 2 and continues operation.




---

**NOTE.** If `srcRoi.width` and `srcRoi.height` are not multiples of 2, the function reduces the values to the nearest multiples of 2. For example, if the size value is 7, the function approximates this value to 6.

---

## Example

The code example below shows how to call `ippiCbYCr422ToYCbCr420_Rotate_8u_C2P3R`.

```
#define ROTATE IPPVC_ROTATE_90CCW
{
    Ipp8u* pSrcIm;
    Ipp8u* pDstIm[3];
    int    stepSrc;
    int    stepDst[3];
    IppiSize srcRoi = {32,32};
    IppiSize dstRoi;

    switch( ROTATE )
    {
    case IPPVC_ROTATE_DISABLE:
    case IPPVC_ROTATE_180:
        dstRoi.width  = srcRoi.width ;
        dstRoi.height = srcRoi.height;
        break;
    case IPPVC_ROTATE_90CCW:
    case IPPVC_ROTATE_90CW:
        dstRoi.width  = srcRoi.height;
        dstRoi.height = srcRoi.width ;
        break;
    }
    pSrcIm = ippiMalloc_8u_C2(srcRoi.width, srcRoi.height, &stepSrc);
```

```

pDstIm[0] = ippiMalloc_8u_C1(dstRoi.width, dstRoi.height, &(stepDst[0]));
pDstIm[1] = ippiMalloc_8u_C1(dstRoi.width/2, dstRoi.height/2, &(stepDst[1]));
pDstIm[2] = ippiMalloc_8u_C1(dstRoi.width/2, dstRoi.height/2, &(stepDst[2]));
ippiCbYCr422ToYCbCr420_Rotate_8u_C2P3R(pSrcIm, stepSrc, srcRoi, pDstIm, stepDst, ROTATE);
ippiFree(pDstIm[0]);
ippiFree(pDstIm[1]);
ippiFree(pDstIm[2]);
ippiFree(pSrcIm);
}

```

## ResizeCCRotate

*Creates a low-resolution preview image for high-resolution video or still capture applications.*

### Syntax

```

IppStatus ippiResizeCCRotate_8u_C2R(const Ipp8u* pSrc, int srcStep, IppiSize srcRoi, Ipp16u
* pDst, int dstStep, int zoomFactor, int interpolation, int colorConversion, int rotation);

```

### Parameters

<i>pSrc</i>	Pointer to the source image. Input byte order is Cb Y Cr Y.
<i>srcStep</i>	Distance in bytes between starts of the consecutive lines in the source image.
<i>srcRoi</i>	ROI of the source image, in pixels. The ROI of the destination image is calculated by user.
<i>pDst</i>	Pointer to the destination image. As an output parameter, indicates a pointer to the start of the buffer containing the resized and rotated image with completed color conversion.
<i>dstStep</i>	Distance in bytes between starts of the consecutive lines in the destination image.
<i>zoomFactor</i>	Parameter, indicating downscale factor; takes values 2, 4 or 8 for 2:1, 4:1, and 8:1 downscale respectively.
<i>interpolation</i>	Type of interpolation to perform resampling of the input image. The following types are currently supported: <code>IPPI_INTER_NN</code> - nearest neighbor interpolation, <code>IPPI_INTER_LINEAR</code> - linear interpolation.
<i>colorConversion</i>	Color conversion control parameter, must be set to one of the following pre-defined values: <code>IPPVC_CbYCr422ToBGR565</code> and <code>IPPVC_CbYCr422ToBGR555</code> . See the corresponding <a href="#">enumerator</a> in the introduction to the General Functions section
<i>rotation</i>	Rotation control parameter. Possible values: <code>IPPVC_ROTATE_DISABLE</code> no rotation <code>IPPVC_ROTATE_90CCW</code> rotating by 90° counterclockwise <code>IPPVC_ROTATE_90CW</code> rotating by 90° clockwise <code>IPPVC_ROTATE_180</code> rotating by 180° See the corresponding <a href="#">enumerator</a> in the introduction to the General Functions section.

## Description

The function `ippiResizeCCRotate_8u_C2R` is declared in the `ippvc.h` file. This function synthesizes a low-resolution preview image for high-resolution video or still capture applications. The function combines scale reduction 2:1, 4:1 or 8:1, color space conversion, and rotation of an image.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if at least one of the specified pointers is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <code>srcRoi.width</code> or <code>srcRoi.height</code> is less than <code>zoomFactor</code> .
<code>ippStsInterpolationErr</code>	Indicates an invalid value of <i>interpolation</i> control parameter.
<code>ippStsResizeFactorErr</code>	Indicates an invalid value of <i>zoomFactor</i> control parameter.
<code>ippStsBadArgErr</code>	Indicates an invalid value of <i>rotation</i> control parameter.
<code>ippStsDoubleSize</code>	Indicates the condition when the values <code>srcRoi.width</code> and <code>srcRoi.height</code> are not multiples of 2. The function reduces their original values to the nearest multiples of 2 and continues operation.



**NOTE.** If `srcRoi.width` and `srcRoi.height` are not multiples of 2, the function reduces the values to the nearest multiples of 2. For example, if the size value is 7, the function approximates this value to 6.

---

## Example

The code example below shows how to call `ippiResizeCCRotate_8u_C2R`.

```
#define ROTATE IPPVC_ROTATE_90CW
#define ZOOMOUT 8

{
    Ipp8u* pSrcIm;
    Ipp16u* pDstIm;
    int stepSrc;
    int stepDst;
    IppiSize srcRoi = {64,64};
    IppiSize dstRoi;
    /* you should calculate dstROI for memory allocation only.*/
    /* dstROI is a function of parameters zoomFactor and rotation.*/
    switch (ROTATE){
        case IPPVC_ROTATE_DISABLE:
        case IPPVC_ROTATE_180:
            dstRoi.width = srcRoi.width /ZOOMOUT;
            dstRoi.height = srcRoi.height/ZOOMOUT;
            break;
        case IPPVC_ROTATE_90CCW:
        case IPPVC_ROTATE_90CW:
            dstRoi.width = srcRoi.height/ZOOMOUT;
            dstRoi.height = srcRoi.width /ZOOMOUT;
            break;
    }
    pSrcIm = ippiMalloc_8u_C2 (srcRoi.width, srcRoi.height, &stepSrc);
```

```

    pDstIm = ippiMalloc_16u_C1(dstRoi.width, dstRoi.height, &stepDst);
    ippiResizeCCRotate_8u_C2R (pSrcIm, stepSrc, srcRoi, pDstIm, stepDst,
    ZOOMOUT, IPP_INTER_NN, IPPVC_CbYCr422ToBGR555, ROTATE);
    ippiFree(pDstIm);
    ippiFree(pSrcIm);
}

```

## Video Processing

This section describes functions that are used to process decompressed video data.

### Deinterlacing Functions

#### DeinterlaceFilterTriangle

*Deinterlaces video plane.*

##### Syntax

```

IppStatus ippiDeinterlaceFilterTriangle_8u_C1R(const Ipp8u* pSrc, Ipp32s srcStep, Ipp8u*
pDst, Ipp32s dstStep, IppiSize roiSize, Ipp32u centerWeight, Ipp32u layout);

```

```

IppStatus ippiDeinterlaceFilterTriangle_8u_C2R(const Ipp8u* pSrc, Ipp32s srcStep, Ipp8u*
pDst, Ipp32s dstStep, IppiSize roiSize, Ipp32u centerWeight, Ipp32u layout);

```

##### Parameters

<i>pSrc</i>	Pointer to the source video plane. For <code>DeinterlaceFilterTriangle_8u_C2R</code> , in nv12 format.
<i>srcStep</i>	Distance in bytes between starts of the consecutive lines in the source video plane.
<i>pDst</i>	Pointer to the destination video plane.
<i>dstStep</i>	Distance in bytes between starts of the consecutive lines in the destination video plane.
<i>roiSize</i>	Size of ROI; height should be greater than 3.
<i>centerWeight</i>	Weight of filtered pixel, must lie within the range from 0 to 256.
<i>layout</i>	Plane layout, required when the plane is only a part of the frame. Takes the following values:

IPP\_UPPER for the first slice

IPP\_CENTER for the middle slices

IPP\_LOWER for the last slice

IPP\_LOWER&&IPP\_UPPER&&IPP\_CENTER for the image that is not sliced.

## Description

The functions `ippiDeinterlaceFilterTriangle_8u_C1R` and `ippiDeinterlaceFilterTriangle_8u_C2R` are declared in the `ippvc.h` header file. These functions deinterlace a video plane. The functions perform triangle filtering of the image to remove interlacing flicker effect that arises when analogue interlaced TV data is viewed on a computer monitor. The `ippiDeinterlaceFilterTriangle_8u_C2R` function works with the `nv12` format.

Pixels are filtered by the following formula:

$$\text{pixel\_new} = [\text{pixel\_above} * (256 - \text{centerWeight}) / 2 + \text{pixel} * \text{centerWeight} + \text{pixel\_below} * (256 - \text{centerWeight}) / 2] / 256.$$

Use recommended values in range from 128 to 256 as `centerWeight`. Otherwise, the final result may be unpredictable.

When `layout` takes the value of `IPP_CENTER` or `IPP_LOWER`, the last data row from the previous slice should be accessible, that is, `pSrc[- srcStep]` should also be a valid pointer.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when at least one input pointer is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error when <code>roiSize</code> has a field with a zero or negative value.
<code>ippStsBadArgErr</code>	Indicates an invalid argument.

## DeinterlaceFilterCAVT

*Performs deinterlacing of two-field image.*

---

### Syntax

```
ippiStatus ippiDeinterlaceFilterCAVT_8u_C1R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst,
int dstStep, Ipp16u threshold, IppiSize roiSize);
```

### Parameters

<code>pSrc</code>	Pointer to the source image ROI.
<code>srcStep</code>	Distance in bytes between starts of consecutive lines in the source image.
<code>pDst</code>	Pointer to the destination image ROI.
<code>dstStep</code>	Distance in bytes between starts of consecutive lines in the destination image.
<code>threshold</code>	Threshold level value.
<code>roiSize</code>	Size of the source and destination image ROI.

### Description

The function `ippiDeinterlaceFilterCAVT_8u_C1R` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)). This function performs deinterlacing of a two-field image, pointed to by `pSrc`, using content adaptive vertical temporal (CAVT) filtering.

The field pointed to by *pSrc* is copied to *pDst*, while the other field in the destination image, pointed to by *pDst+dstStep*, is the interpolated one. Note that you can set the pointers to the bottom left corner of the images and use negative steps to have the bottom field unchanged and the top one interpolated.

The *threshold* parameter is the edge detection threshold with the valid range [0-2041] regulating the probability of temporal interpolation: 0 means that all the pixels are interpolated only spatially, from the vertically neighbouring pixels of the copied field, 2041 - that combined spatial-temporal interpolation, involving the pixels from the modified field as well, is applied to all the pixels.

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointers is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize.width</i> is less than or equal to 0, or <i>roiSize.height</i> is odd or less than 8.

## Median

*Creates an image consisting of median values of three source images.*

### Syntax

```
ippStatus ippMedian_8u_P3C1R(const Ipp8u* pSrc[3], int srcStep, Ipp8u* pDst, int dstStep,
IppiSize roiSize);
```

### Parameters

<i>pSrc</i>	Array of pointers to the ROI in each plane of the source image.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in each plane of the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>roiSize</i>	Size of the source and destination image ROI.

### Description

The function `ippMedian_8u_P3C1R` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)). This function sets each pixel in the destination image ROI as the median value of correspondent pixels in the each plane of the source image.

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointers is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.

## DeinterlaceMedianThreshold

*Performs median deinterlacing with threshold.*

---

### Syntax

```
IppStatus ippiDeinterlaceMedianThreshold_8u_C1R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst, int dstStep, IppiSize roiSize, int threshold, int fieldNum, int bCopyBorder);
```

### Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>roiSize</i>	Size of the source and destination image ROI.
<i>threshold</i>	Threshold level value.
<i>fieldNum</i>	Specifies field to process. 0 means the top field, 1 means the bottom field.
<i>bCopyBorder</i>	Controls how to process the border. If its value is not equal to 0, the border line is copied to the destination image, otherwise the border line is processed like internal lines.

### Description

The function `ippiDeinterlaceMedianThreshold_8u_C1R` is declared in the `ippvc.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)). This function sets each pixel in the output buffer in the following way:

- Copy an unprocessed field from the source to the destination.
- Copy the border lines from the source to the destination in the case *bCopyBorder* parameter is set.
- Perform processing of the field, chosen by parameter *fieldNum*. If the absolute difference between the current pixel value and the median value (M) of three input pixel values, taken from the current position, upper and lower pixels adjacent to the current one, is greater or equal to the threshold, the output pixel value is set to the median value M, otherwise it is set to the current pixel.

The size of the source image ROI is equal to the size *roiSize* of the destination image ROI.

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointers is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.



## DeinterlaceEdgeDetect

*Generates image field using EdgeDetect filter.*

---

### Syntax

```
IppStatus ippiDeinterlaceEdgeDetect_8u_C1R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst,
int dstStep, IppiSize roiSize, int fieldNum, int bCopyBound);
```

### Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>roiSize</i>	Size of the source and destination image ROI.
<i>fieldNum</i>	Specifies field to process. 0 means the top field, 1 means the bottom field.
<i>bCopyBorder</i>	Controls how to process the border. If its value is not equal to 0, the border line is copied to the destination image, otherwise the border line is processed like internal lines.

### Description

The function `ippiDeinterlaceEdgeDetect_8u_C1R` is declared in the `ippvc.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)). The input of the function must be the field chosen to process. The filter outputs a field of the same parity as the source field.

Parameter *fieldNum* shows the field to generate; the other field must be copied by the user. Border lines are copied if the *bCopyBorder* parameter is enabled.

Processing of the input field is done by the Edge Detect algorithm.

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.

## DeinterlaceMotionAdaptive

*Performs deinterlacing using temporal and spatial interpolations.*

---

### Syntax

```
IppStatus ippiDeinterlaceMotionAdaptive_8u_C1(const Ipp8u* pSrcPlane[4], int srcStep,
Ipp8u* pDst, int dstStep, IppiSize planeSize, int threshold, int topFirst, int topField,
int copyField, int artifactProtection);
```

### Parameters

<i>pSrcPlane</i>	Array of pointers to source images: four consecutive source plane pointers.
<i>srcStep</i>	Distance in bytes between starts of the consecutive lines in the source image buffer.
<i>pDst</i>	Pointer to the destination image.
<i>dstStep</i>	Distance in bytes between starts of the consecutive lines in the destination image buffer.
<i>planeSize</i>	Size of the image plane in pixels.
<i>threshold</i>	Threshold level value [0-256]. The default value is 12. A tradeoff between flickering and residual combing artifacts. You may decrease the value of the threshold to reduce combing artifacts in moving objects, but note that flickering in the static region may increase. Zero value corresponds to the bob-deinterlacing.
<i>topFirst</i>	Defines the field order for the video sequence; can be 0 - bottom field is the first field, 1 - top field is the first field.
<i>topField</i>	Defines the field to process; can be 0 - bottom field, 1 - top field.
<i>copyField</i>	Defines copying of unprocessed field from the source frame to the destination; can be 0 - the unprocessed field is not copied to the destination frame, 1 - the unprocessed field is copied from the source frame to the destination.
<i>artifactProtection</i>	Sets the additional artifact protection to suppress distortion; can be 0 - disabling artifact protection, 1 - enabling artifact protection.

### Description

The function `ippiDeinterlaceMotionAdaptive_8u_C1` is declared in the `ippvc.h` file. The function performs deinterlacing of a two-field image using two consecutive steps - *temporal* and *spatial* interpolations. The first step is used to detect significant temporal difference between pixels from different consecutive frames. The second step is used to detect minor spatial changes between vertical neighbor pixels from opposite fields.

The input sequence of source planes must be set according to the value of the parameters *topFirst* and *topField*. If the processed field is the second field in the frame, the input source planes must be from the previous, current, next, and next after next frames. Otherwise, if the first field is processed, the input source planes must be from the pre-previous, previous, current, and next source frames.

The function can be used to process the source image by parts (slices). To keep the equivalence of slice processing to the processing of the whole image, the following method should be used depending on the parameter *topField*:

- For the top field, the upper part of the image, which begins from the zero line and contains an even number of lines processed by the function in an ordinary way, *topField* is set to 1. Other slices of the source image must contain all necessary neighbor lines to perform deinterlacing. Because of this requirement, other slices of the image that begin from an even line and contain an even number of lines should be extended with one upper neighbor odd line, and *topField* should be set to 0.
- For the bottom field, the lower part of the image, which ends with the last odd line and contains an even number of lines processed by the function in an ordinary way, *topField* is set to 0. Other slices of the source image that end with the odd line and contain an even number of lines should be extended with one lower neighbor even line, and *topField* should be set to 0.

The unprocessed field should be copied to the destination frame separately, if necessary.

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <i>planeSize</i> has a field with zero or negative value.
<code>ippStsMemAllocErr</code>	Indicates a memory allocation error.

## DeinterlaceBlendGetSize

*Calculates the size of an internal structure for DeinterlaceBlend function.*

### Syntax

```
IppStatus ippDeinterlaceBlendGetSize_8u_C1(int* pStateSize);
IppStatus ippDeinterlaceBlendGetSize_8u_C2(int* pSpecSize);
```

### Parameters

<i>pStateSize</i>	Pointer to the computed size in bytes of the <code>IppiDeinterlaceBlendState_8u_C1</code> structure.
<i>pSpecSize</i>	Pointer to the computed size in bytes of the <code>IppiDeinterlaceBlendSpec_8u_C2</code> structure.

### Description

The functions `DeinterlaceBlendGetSize_8u_C1` and `DeinterlaceBlendGetSize_8u_C2` are declared in the `ippvc.h` file. The functions calculates the number of bytes needed to allocate for the structures `IppiDeinterlaceBlendState_8u_C1` and `IppiDeinterlaceBlendSpec_8u_C2` respectively.

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
<code>ippStsNullPtrErr</code>	Indicates an error condition when the specified pointer is <code>NULL</code> .

## Example

The code example below shows how to use `DeinterlaceBlendGetSize_8u_C1` function.

```

Int      stateSize;
Ipp8u*   pMemState;
IppiDeinterlaceBlendState_8u_C1* pState=NULL;
IppiSize planeSize={1664,1088}; /* size of the image plane in pixels, no dependency
                                   on stateSize */

Sts = ippiDeinterlaceBlendGetSize_8u_C1 (&stateSize);
pMemState = malloc(stateSize);

sts = ippiDeinterlaceBlendInit_8u_C1 (planeSize, blendThresh[2], blendConstants[2],
                                     &pState, pMemState);
{
    ...pState processing...
    ...
}
free(pMemState);
pState = NULL;

```

The code example below shows how to use `DeinterlaceBlendGetSize_8u_C2` function.

```

Int      stateSize;
Ipp8u*   pMemState;
IppiDeinterlaceBlendState_8u_C2* pSpec=NULL;
IppiSize planeSize={1664,1088}; /* size of the image plane in pixels, no dependency
                                   on stateSize */

Sts = ippiDeinterlaceBlendGetSize_8u_C2 (&specSize);
pMemState = malloc(specSize);

sts = ippiDeinterlaceBlendInit_8u_C2 (planeSize, blendThresh[2], blendConstants[2],
                                     &pSpec, pMemState);
{
    ...pSpec processing...
    ...
}
free(pMemState);
pSpec = NULL;

```

## DeinterlaceBlendInit

*Initializes an internal structure for DeinterlaceBlend function.*

---

### Syntax

```

IppStatus ippiDeinterlaceBlendInit_8u_C1(IppiSize size, int blendThresh[2], double
blendConstants[2], IppiDeinterlaceBlendState_8u_C1** ppState, Ippi8u* pMemState);

IppStatus ippiDeinterlaceBlendInit_8u_C2(IppiSize size, int blendThresh[2], double
blendConstants[2], IppiDeinterlaceBlendState_8u_C2** ppSpec, Ippi8u* pMemState);

```

### Parameters

<i>size</i>	Size of the image plane in pixels.
-------------	------------------------------------

<i>blendThresh</i>	Array of two thresholds to determine <i>alpha</i> -coefficients; parameter values must be in the range [0,255]. Recommended values are 5, 9.
<i>blendConstants</i>	Array of two values for <i>alpha</i> -coefficients; parameter values must be in the range [0.0, 0.1]. Recommended values are 0.3, 0.7. The <i>blendConstants</i> parameter corresponds to <i>blendThresh</i> .
<i>ppState</i>	Pointer to pointer to an instance of the <code>IppiDeinterlaceBlendState_8u_C1</code> structure initialized in accordance with the values of <i>blendConstants</i> [2] and <i>blendThresh</i> [2]. The function <code>DeinterlaceBlendInit__8u_C1</code> allocates the structure in the <i>pMemState</i> <code>IppiDeinterlaceBlendState_8u_C1</code> buffer.
<i>ppSpec</i>	Pointer to pointer to an instance of the <code>IppiDeinterlaceBlendState_8u_C2</code> structure initialized in accordance with the values of <i>blendConstants</i> [2] and <i>blendThresh</i> [2]. The function <code>DeinterlaceBlendInit__8u_C2</code> allocates the structure in the <i>pMemState</i> buffer.
<i>pMemState</i>	Pointer to an array that you allocated for <code>IppiDeinterlaceBlendState_8u_C1</code> or <code>IppiDeinterlaceBlendState_8u_C2</code> structure. The size of the array does not depend on the image plane size. Call the <code>DeinterlaceBlendGetSize_8u_C1</code> or <code>DeinterlaceBlendGetSize_8u_C2</code> function respectively to calculate it.

## Description

The functions `ippiDeinterlaceBlendInit_8u_C1` and `ippiDeinterlaceBlendInit_8u_C2` are declared in the `ippvc.h` file. The functions initialize a structure pointed to by *ppState* or *ppSpec* respectively in accordance with the values of *blendConstants*[2] and *blendThresh*[2]. You should allocate memory for the structure in the external buffer *pMemState*. If you use these functions, do not call `DeinterlaceBlendFree_8u_C1` or `DeinterlaceBlendFree_8u_C2` to free the *pMemState* buffer.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
<code>ippStsNullPtrErr</code>	Indicates an error condition when the specified pointer is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <i>size</i> has a field with a value smaller than 3.

## Example

The code example below shows how to use `DeinterlaceBlendInit_8u_C1` function.

```
Int      stateSize;
Ipp8u*   pMemState;
IppiDeinterlaceBlendState_8u_C1* pState=NULL;
IppiSize planeSize={1664,1088}; /* size of the image plane in pixels, no dependency
                                on stateSize */

Sts = ippiDeinterlaceBlendGetSize_8u_C1 (&stateSize);
pMemState = malloc(stateSize);
```

```
sts = ippiDeinterlaceBlendInit_8u_C1 (planeSize, blendThresh[2], blendConstants[2],
                                     &pState, pMemState);
{
...pState processing...
...
}
free(pMemState);
pState = NULL;
```

The code example below shows how to use DeinterlaceBlendInit\_8u\_C2 function.

```
Int      stateSize;
Ipp8u*   pMemState;
IppiDeinterlaceBlendState_8u_C2* pSpec=NULL;
IppiSize planeSize={1664,1088}; /* size of the image plane in pixels, no dependency
                                on stateSize */

Sts = ippiDeinterlaceBlendGetSize_8u_C2 (&specSize);
pMemState = malloc(specSize);

sts = ippiDeinterlaceBlendInit_8u_C2 (planeSize, blendThresh[2], blendConstants[2],
                                     &pSpec, pMemState);
{
...pSpec processing...
...
}
free(pMemState);
pSpec = NULL;
```

## DeinterlaceBlendInitAlloc

*Allocates and initializes internal structure for DeinterlaceBlend function.*

---

### Syntax

```
IppStatus ippiDeinterlaceBlendInitAlloc_8u_C1(IppiSize planeSize, int blendThresh[2],
double blendConstants[2], IppiDeinterlaceBlendState_8u_C1** ppState);

IppStatus ippiDeinterlaceBlendInitAlloc_8u_C2(IppiSize size, int blendThresh[2], double
blendConstants[2], IppiDeinterlaceBlendSpec_8u_C2** ppSpec);
```

### Parameters

<i>planeSize, size</i>	Size of the image plane in pixels.
<i>blendThresh</i>	Array of two thresholds to determine <i>alpha</i> -coefficients; parameter values must be in the range [0,255]. Recommended values are 5, 9.
<i>blendConstants</i>	Array of two values for <i>alpha</i> -coefficients; parameter values must be in the range [0.0, 0.1]. Recommended values are 0.3, 0.7. The <i>blendConstants</i> parameter corresponds to <i>blendThresh</i> .
<i>ppState</i>	Pointer to pointer to an instance of the IppiDeinterlaceBlendState_8u_C1 structure containing internal table to store calculated <i>alpha</i> -coefficients.
<i>ppSpec</i>	Pointer to pointer to an instance of the IppiDeinterlaceBlendSpec_8u_C2 structure.

## Description

The functions `ippiDeinterlaceBlendInitAlloc_8u_C1` and `ippiDeinterlaceBlendInitAlloc_8u_C2` are declared in the `ippvc.h` file. The functions perform the following sequence of steps:

- Allocating memory for the internal structure `IppiDeinterlaceBlendState_8u_C1` or `IppiDeinterlaceBlendState_8u_C2` respectively
- Setting a pointer `**ppState/**ppSpec` to the instance of the internal structure
- Initializing the internal structure in accordance with the values of `blendConstants[2]` and `blendThresh[2]`.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
<code>ippStsNullPtrErr</code>	Indicates an error condition when the specified pointer is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <code>planeSize/size</code> has a field with a value smaller than 3.
<code>ippStsMemAllocErr</code>	Indicates a memory allocation error.

## Example

The code example below shows how to use `DeinterlaceBlendInitAlloc_8u_C1` function.

```
IppiDeinterlaceBlendSpec_8u_C1* pState=NULL;
IppiSize planeSize = {1664,1088};

sts = ippiDeinterlaceBlendInitAlloc_8u_C1 (planeSize, blendThresh[2], blendConstants[2],
                                           &pState);
{
    ...pState processing...
}
ippiDeinterlaceBlendFree_8u_C1(pState);
```

The code example below shows how to use `DeinterlaceBlendInitAlloc_8u_C2` function.

```
IppiDeinterlaceBlendSpec_8u_C2* pSpec=NULL;
IppiSize planeSize = {1664,1088};

sts = ippiDeinterlaceBlendInitAlloc_8u_C2 (planeSize, blendThresh[2], blendConstants[2],
                                           &pSpec);
{
    ...pSpec processing...
}
ippiDeinterlaceBlendFree_8u_C2(pSpec);
```

## DeinterlaceBlend

*Calculates output pixels as alpha blends of the results of two filters applied to input pixels.*

---

### Syntax

```
IppStatus ippiDeinterlaceBlend_8u_C1(const Ipp8u* pSrcPlane[3], int srcStep, Ipp8u* pDst,
int dstStep, IppiSize planeSize, int topFirst, int topField, int copyField,
IppiDeinterlaceBlendState_8u_C1* pState);
```

```
IppStatus ippiDeinterlaceBlend_8u_C2(const Ipp8u* pSrcPrev, int srcPrevStep, const Ipp8u*
pSrcCur, int srcCurStep, const Ipp8u* pSrcNext, int srcNextStep, Ipp8u* pDst, int dstStep,
IppiSize size, IppiFrameFieldType fieldFirst, IppiFrameFieldType fieldProcess, IppBool
fieldCopy, IppiDeinterlaceBlendSpec_8u_C2* pSpec);
```

### Parameters

<i>pSrcPlane</i>	Array of pointers to source images: previous, current, and next source plane pointers.
<i>pSrcPrev</i>	Pointer to the previous source images.
<i>pSrcCur</i>	Pointer to the current source images.
<i>pSrcPlane</i>	Pointer to the next source images.
<i>srcStep</i>	Distance in bytes between starts of the consecutive lines in the source image buffer.
<i>srcPrevStep</i>	Distance in bytes between starts of the consecutive lines in the buffer of the previous source image.
<i>srcCurStep</i>	Distance in bytes between starts of the consecutive lines in the buffer of the current source image.
<i>srcNextStep</i>	Distance in bytes between starts of the consecutive lines in the buffer of the next source image.
<i>pDst</i>	Pointer to the destination image.
<i>dstStep</i>	Distance in bytes between starts of the consecutive lines in the destination image buffer.
<i>planeSize, size</i>	Size of the source and destination image planes in pixels.
<i>topFirst</i>	Defines the field order for the video sequence; can be 0 - bottom field is the first field, 1 - top field is the first field.
<i>topField</i>	Defines the field to process; can be 0 - bottom field, 1 - top field.
<i>copyField</i>	Defines copying of unprocessed field from the source frame to the destination; can be 0 - the unprocessed field is not copied to the destination frame, 1 - the unprocessed field is copied from the source frame to the destination.
<i>fieldFirst</i>	Defines the field order for the video sequence; can be BOTTOM_FIELD - bottom field is the first field, TOP_FIELD - top field is the first field.
<i>fieldProcess</i>	Defines the field to process; can be BOTTOM_FIELD - bottom field, TOP_FIELD - top field.



<i>fieldCopy</i>	Defines copying of unprocessed field from the source frame to the destination; can be <code>IppFalse</code> - the unprocessed field is not copied to the destination frame, <code>IppTrue</code> - the unprocessed field is copied from the source frame to the destination.
<i>pState</i>	Pointer to the internal structure <code>IppiDeinterlaceBlendState_8u_C1</code> .
<i>pSpec</i>	Pointer to the internal structure <code>IppiDeinterlaceBlendSpec_8u_C2</code> .

## Description

The functions `ippiDeinterlaceBlend_8u_C1` and `ippiDeinterlaceBlend_8u_C2` are declared in the `ippvc.h` file. The functions calculate each output pixel as an alpha blend of the results of two filters - EdgeDetect and Median 3 - applied to the input pixels. The blending coefficient *alpha* is determined by differences of the corresponding pixels in the previous, current, and subsequent source frames.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
<code>ippStsNullPtrErr</code>	Indicates an error condition when the specified pointer is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <i>planeSize/size</i> has a field with a value smaller than 3.
<code>ippStsMemAllocErr</code>	Indicates a memory allocation error.

## DeinterlaceBlendFree

*Releases memory allocated by DeinterlaceBlendInitAlloc function.*

---

## Syntax

```
IppStatus ippiDeinterlaceBlendFree_8u_C1(IppiDeinterlaceBlendState_8u_C1* pState);
IppStatus ippiDeinterlaceBlendFree_8u_C2(IppiDeinterlaceBlendSpec_8u_C2* pSpec);
```

## Parameters

<i>pState</i>	Pointer to the internal structure <code>IppiDeinterlaceBlendState_8u_C1</code> .
<i>pSpec</i>	Pointer to the internal structure <code>IppiDeinterlaceBlendSpec_8u_C2</code> .

## Description

The functions `ippiDeinterlaceBlendFree_8u_C1` and `ippiDeinterlaceBlendFree_8u_C2` are declared in the `ippvc.h` file. The functions release memory that was allocated by the `ippiDeinterlaceBlendInitAlloc_8u_C1` or `ippiDeinterlaceBlendInitAlloc_8u_C2` function, respectively. Call these functions after finishing deinterlacing of the video sequence.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
<code>ippStsNullPtrErr</code>	Indicates an error condition when the specified pointer is <code>NULL</code> .

`ippStsContextMatchErr` Indicates an error condition when the `pState/pSpec` structure was not allocated by the `ippiDeinterlaceBlendInitAlloc_8u_C1` or `ippiDeinterlaceBlendInitAlloc_8u_C2` function, respectively.

## Denoising Functions

This subsection describes denoising filtering functions. Note that the `FilterDenoiseCASTInit` and `FilterDenoiseCAST` functions use the following structure:

```
typedef struct {
    Ipp8u TemporalDifferenceThreshold;    // default 16 - range [0, 255]
    Ipp8u NumberOfMotionPixelsThreshold; // default 0 - range [0, 16]
    Ipp8u StrongEdgeThreshold;           // default 8 - range [0, 255]
    Ipp8u BlockWidth;                    // default 4 - range [1, 16]
    Ipp8u BlockHeight;                   // default 4 - range [1, 16]
    Ipp8u EdgePixelWeight;                // default 128 - range [0, 255]
    Ipp8u NonEdgePixelWeight;             // default 16 - range [0, 255]
    Ipp8u GaussianThresholdY;            // default 12
    Ipp8u GaussianThresholdUV;           // default 6
    Ipp8u HistoryWeight;                  // default 192 - range [0, 255]
} IppDenoiseCAST;
```

## FilterDenoiseCASTInit

*Initializes a denoise specification structure for content adaptive spatio-temporal noise reduction filtering.*

---

### Syntax

```
IppStatus ippiFilterDenoiseCASTInit(IppDenoiseCAST* pParam);
```

### Parameters

`pParam` Pointer to an instance of the `IppDenoiseCAST` structure that contains input parameters for `ippiFilterDenoiseCAST()`.

### Description

The function `ippiFilterDenoiseCASTInit` is declared in the `ippvc.h` file. The function initializes the denoise specification structure for content adaptive spatio-temporal (CAST) noise reduction filtering.

### Return Values

`ippStsNoErr` Indicates no error. Any other value indicates an error.

`ippStsNullPtrErr` Indicates an error condition when the specified pointer is `NULL`.

## FilterDenoiseCAST

*Performs content adaptive spatio-temporal noise reduction filtering.*

---

### Syntax

```
IppStatus ippiFilterDenoiseCAST_8u_C1R(const Ipp8u* pSrcCur, const Ipp8u* pSrcPrev, Ipp32s srcStep, const Ipp8u* pSrcEdge, Ipp32s srcEdgeStep, IppiSize srcRoiSize, Ipp8u* pDst, Ipp32s dstStep, Ipp8u* pHistoryWeight, IppiDenoiseCAST* pParam);
```

```
IppStatus ippiFilterDenoiseCASTYUV422_8u_C2R(const Ipp8u* pSrcCur, const Ipp8u* pSrcPrev, Ipp32s srcStep, const Ipp8u* pSrcEdge, Ipp32s srcEdgeStep, IppiSize srcRoiSize, Ipp8u* pDst, Ipp32s dstStep, Ipp8u* pHistoryWeight, IppiDenoiseCAST* pParam);
```

### Parameters

<i>pSrcCur</i>	Pointer to the current source image frame.
<i>pSrcPrev</i>	Pointer to the previous processed image frame; can be NULL (see Description below).
<i>srcStep</i>	Distance in bytes between starts of the consecutive lines in the source image.
<i>pSrcEdge</i>	Pointer to the edge-detection filtered source image frame.
<i>srcEdgeStep</i>	Distance in bytes between starts of the consecutive lines in the edge-detection filtered source image frame.
<i>pDst</i>	Pointer to the destination image frame.
<i>dstStep</i>	Distance in bytes between starts of the consecutive lines in the destination image.
<i>srcRoiSize</i>	Size of the source and destination ROI.
<i>pHistoryWeight</i>	Pointer to the array of per-block history weights. Values correspond to blocks in raster order and are modified by the function. If <i>pHistoryWeight</i> == NULL, <i>HistoryWeight</i> from the structure pointed to by <i>pParam</i> , or the default value, is used for all blocks instead.
<i>pParam</i>	Pointer to the structure containing input parameters. If <i>pParam</i> == NULL, the default parameters are used.

### Description

The functions `ippiFilterDenoiseCAST_8u_C1R` and `ippiFilterDenoiseCASTYUV422_8u_C2R` are declared in the `ippvc.h` file. The functions perform content adaptive spatio-temporal (CAST) noise reduction filtering for the packed YUY2 format (`ippiFilterDenoiseCASTYUV422_8u_C2R`) and for any planar format (`ippiFilterDenoiseCAST_8u_C1R`).

When called with `pSrcPrev == NULL` (no previous frame available), the functions perform spatial denoising only. In this case, the values pointed to by `pHistoryWeight` are not used but updated (if `pHistoryWeight != NULL`) by the function.

If the first call of the function is performed with `pSrcPrev != NULL` and `pHistoryWeight != NULL`, the user should fill the array with desired initial values.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
<code>ippStsNullPtrErr</code>	Indicates an error condition when the <code>pSrcCur</code> or <code>pSrcEdge</code> pointers are NULL.
<code>ippStsSizeErr</code>	Indicates an incorrect input size of the image, that is, <code>srcRoiSize</code> has a field with zero or negative value or, in the case of <code>ippiFilterDenoise-CASTYUV422_8u_C2R</code> , <code>srcRoiSize.width</code> is odd.

## FilterDenoiseSmooth

*Performs spatial noise reduction filtering.*

---

### Syntax

```
ippStatus ippiFilterDenoiseSmooth_8u_C1R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst, int dstStep, IppiSize size, IppiRect roi, Ipp8u threshold, Ipp8* pWorkBuffer);
```

### Parameters

<code>pSrc</code>	Pointer to the source image origin.
<code>srcStep</code>	Distance in bytes between starts of consecutive lines in the source image.
<code>pDst</code>	Pointer to the destination image origin.
<code>dstStep</code>	Distance in bytes between starts of consecutive lines in the destination image.
<code>size</code>	Size of the source image. The destination image is of the same size.
<code>roi</code>	Region of interest in the source image (of the <code>IppiRect</code> type). The destination image has the same ROI.
<code>threshold</code>	Parameter of the denoise algorithm for pixel comparison. If difference between the compared pixels is below the <code>threshold</code> value, the pixels are considered to be noised and will be smoothed in line with values of the neighboring pixels. Possible range is (0, 255).
<code>pWorkBuffer</code>	Pointer to the external work buffer of the size $\geq 2 * (roiSize.width + 8) * (roiSize.height + 8)$ .

### Description

The function `ippiFilterDenoiseSmooth_8u_C1R` is declared in the `ippvc.h` file. The function performs Spatial Noise Reduction (SNR) filtering.

The algorithm applies a lowpass filter to an image by using a 3x3 mask and analyzes difference between pixels in accordance with `threshold`. If the difference is smaller than the threshold, the corresponding pixels are noised and will be smoothed. The 3x3 mask is also used for smoothing.

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
--------------------------	---

<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an incorrect input size of the image.

## FilterDenoiseAdaptiveInitAlloc

*Creates and initializes a denoise specification structure for spatio-temporal adaptive noise reduction filtering.*

### Syntax

```
IppStatus ippFilterDenoiseAdaptiveInitAlloc_8u_C1(IppiDenoiseAdaptiveState_8u_C1** ppState,
IppiSize roiSize, IppiSize maskSize);
```

### Parameters

<code>ppState</code>	Pointer to pointer to the denoise specification structure to be created.
<code>roiSize</code>	Size of the source image ROI in pixels that will be processed.
<code>maskSize</code>	Parameter that defines the region used in current pixel transformation.

### Description

The function `ippFilterDenoiseAdaptiveInitAlloc_8u_C1` is declared in the `ippvc.h` file. The function creates and initializes the denoise specification structure for spatio-temporal adaptive noise reduction filtering.

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
<code>ippStsNullPtrErr</code>	Indicates an error condition when the specified pointer is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <code>roiSize</code> or <code>maskSize</code> have a field with zero or negative value.
<code>ippStsMemAllocErr</code>	Indicates an error condition when no memory is allocated.

## FilterDenoiseAdaptiveFree

*Closes a denoise specification structure for spatio-temporal adaptive noise reduction filtering.*

### Syntax

```
IppStatus ippFilterDenoiseAdaptiveFree_8u_C1(IppiDenoiseAdaptiveState_8u_C1* pState);
```

### Parameters

<code>pState</code>	Pointer to the denoise specification structure to be closed.
---------------------	--

### Description

The function `ippFilterDenoiseAdaptiveFree_8u_C1` is declared in the `ippvc.h` file. The function closes the denoise specification structure for spatio-temporal adaptive noise reduction filtering.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
<code>ippStsNullPtrErr</code>	Indicates an error condition when the specified pointer is NULL.

## FilterDenoiseAdaptive

*Performs spatio-temporal adaptive noise reduction filtering.*

---

### Syntax

```
ippStatus ippiFilterDenoiseAdaptive_8u_C1R(const Ipp8u* pSrcPlane[3], int srcStep, Ipp8u*
pDst, int dstStep, IppiSize size, IppiRect roi, IppiSize maskSize, Ipp8u threshold,
IppvcNoiseBlurFlag blurType, IppiDenoiseAdaptiveState_8u_C1* pState);
```

### Parameters

<code>pSrcPlane</code>	Array of pointers to frames that incorporates three source plane pointers: <code>pSrcPlane[0]</code> points to the previous source image origin, <code>pSrcPlane[1]</code> points to the current source image origin, <code>pSrcPlane[2]</code> points to the next source image origin.
<code>srcStep</code>	Distance in bytes between starts of consecutive lines in the source image.
<code>pDst</code>	Pointer to the destination image origin.
<code>dstStep</code>	Distance in bytes between starts of consecutive lines in the destination image.
<code>size</code>	Size of the source image. The destination image is of the same size.
<code>roi</code>	Region of interest in the source image (of the <code>IppiRect</code> type). The destination image has the same ROI. Size of ROI must be the same as <code>roiSize</code> in <code>ippiFilterDenoiseAdaptiveInitAlloc_8u_C1R</code> .
<code>maskSize</code>	Parameter that defines the region used in current pixel transformation; must be the same as <code>maskSize</code> in <code>ippiFilterDenoiseAdaptiveInitAlloc_8u_C1R</code> .
<code>threshold</code>	Parameter of the denoise algorithm for pixel comparison. If difference between the compared pixels is below the <code>threshold</code> value, the pixels are considered to be noised.
<code>blurType</code>	Type of blurring of the noised pixel. Possible modes are as follows: <code>IPPVC_NOISE_BLUR0</code> , <code>IPPVC_NOISE_BLUR1</code> , <code>IPPVC_NOISE_BLUR2</code> , <code>IPPVC_NOISE_BLUR3</code> . See Description for details.
<code>pState</code>	Pointer to the <code>IppiDenoiseAdaptiveState</code> specification structure.

### Description

The function `ippiFilterDenoiseAdaptive_8u_C1R` is declared in the `ippvc.h` file. The function performs spatio-temporal adaptive noise reduction filtering. It requires information about previous, current and next source planes. The algorithm compares differences between the pixel of interest of the current frame and neighbors of the pixel of interest of the previous frame with the threshold value as well as the same differences between the

pixel of interest of the current frame and the neighbors of the pixel of interest of the next frame. The neighbors of the pixel of interest are described by *maskSize*. The pixel is detected as noised if one of the differences is smaller than *threshold*. In that case, the noised pixel will be blurred by using information about the neighbors. There are four methods of blurring depending on *blurType*:

```
case IPPVC_NOISE_BLUR0:
    prev = pSrcROIPrev[posCur];
    next = pSrcROINext[posCur];
    break;
case IPPVC_NOISE_BLUR1:
    prev = sumPrev;
    next = pSrcROINext[posCur];
    break;
case IPPVC_NOISE_BLUR2:
    prev = pSrcROIPrev[posCur];
    next = sumNext;
    break;
case IPPVC_NOISE_BLUR3:
    prev = sumPrev;
    next = sumNext;
    break;
pDstROI[posCur] = (prev + next + sumCur)* scaleFactor,
```

where

*posCur* - position of the interesting pixel

*sumPrev* - sum of neighbours of the interesting pixel of the previous frame

*sumNext* - sum of neighbours of the interesting pixel of the next frame

*scaleFactor* - some normalization factor excluding overflow.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
<code>ippStsNullPtrErr</code>	Indicates an error condition when one of the the specified pointers is NULL.
<code>ippStsSizeErr</code>	Indicates an incorrect input size of the image, mask, or ROI.

## FilterDenoiseMosquitoInitAlloc

*Creates and initializes a denoise specification structure for spatio-temporal motion adaptive mosquito noise reduction filtering.*

---

### Syntax

```
IppStatus ippiFilterDenoiseMosquitoInitAlloc_8u_C1(IppiDenoiseMosquitoState_8u_C1** ppState,
IppiSize roiSize);
```

### Parameters

<i>ppState</i>	Pointer to pointer to the denoise specification structure to be created.
<i>roiSize</i>	Size of the source image ROI in pixels that will be processed.

### Description

The function `ippiFilterDenoiseMosquitoInitAlloc_8u_C1` is declared in the `ippvc.h` file. The function creates and initializes the denoise specification structure for spatio-temporal motion adaptive mosquito noise reduction filtering.

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
<code>ippStsNullPtrErr</code>	Indicates an error condition when the specified pointer is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.
<code>ippStsMemAllocErr</code>	Indicates an error condition when no memory is allocated.

## FilterDenoiseMosquitoFree

*Closes a denoise specification structure for spatio-temporal motion adaptive mosquito noise reduction filtering.*

---

### Syntax

```
IppStatus ippiFilterDenoiseMosquitoFree_8u_C1(IppiDenoiseMosquitoState_8u_C1* pState);
```

### Parameters

<i>pState</i>	Pointer to the denoise specification structure to be closed.
---------------	--

### Description

The function `ippiFilterDenoiseMosquitoFree_8u_C1` is declared in the `ippvc.h` file. The function closes the denoise specification structure for spatio-temporal motion adaptive mosquito noise reduction filtering.

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
--------------------------	---



`ippStsNullPtrErr`Indicates an error condition when the specified pointer is `NULL`.

## FilterDenoiseMosquito

*Performs spatio-temporal motion adaptive mosquito noise reduction filtering.*

### Syntax

```
ippStatus ippiFilterDenoiseMosquito_8u_C1R(const Ipp8u* pSrcPlane[2], int srcStep, Ipp8u* pDst, int dstStep, IppiSize size, IppiRect roi, IppiDenoiseMosquitoState_8u_C1* pState);
```

### Parameters

<code>pSrcPlane</code>	Array of pointers to frames that incorporates two source plane pointers: <code>pSrcPlane[0]</code> points to the previous source image origin, <code>pSrcPlane[1]</code> points to the current source image origin.
<code>srcStep</code>	Distance in bytes between starts of consecutive lines in the source image.
<code>pDst</code>	Pointer to the destination image origin.
<code>dstStep</code>	Distance in bytes between starts of consecutive lines in the destination image.
<code>size</code>	Size of the source image. The destination image is of the same size.
<code>roi</code>	Region of interest in the source image (of the <code>IppiRect</code> type). The destination image has the same ROI. Size of ROI must be the same as <code>roiSize</code> in <code>ippiFilterDenoiseMosquitoInitAlloc_8u_C1R</code> .
<code>pState</code>	Pointer to the denoise specification structure.

### Description

The function `ippiFilterDenoiseMosquito_8u_C1R` is declared in the `ippvc.h` file. The function performs spatio-temporal motion adaptive mosquito noise reduction filtering. It requires information about previous and current source planes.

The Mosquito Noise (MN) manifests itself as fluctuations in luminance/chrominance level around edges and moving objects in a video sequence. MN is mainly the result of the ringing effect and of the inability of the translational motion compensation technique to find the best match.

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
<code>ippStsNullPtrErr</code>	Indicates an incorrect memory address.
<code>ippStsSizeErr</code>	Indicates an incorrect input size of the image.

## MPEG-1 and MPEG-2

This section contains functions for encoding and decoding of video data according to MPEG-1([ISO11172]) and MPEG-2([ISO13818]) standards.

The use of some functions described in this section is demonstrated in Intel® IPP Samples downloadable from <http://www.intel.com/cd/software/products/asmo-na/eng/220046.htm> .

## Structures and Enumerations

The enumeration `IPPVC_MV_TYPE` indicates motion vector types (`FIELD`, `FRAME`).

```
typedef enum _IPPVC_MV_TYPE{
    IPPVC_MV_FIELD =      0x0,
    IPPVC_MV_FRAME =      0x1,
}IPPVC_MV_TYPE;
```

## MPEG-1 and MPEG-2 Video Decoding Functions

Function Short Name	Description
<b>Variable Length Decoding</b>	
<code>HuffmanTableInitAlloc</code>	Allocates memory and initializes a table that contains codes for macroblock address increment, macroblock type, macroblock pattern, or motion vectors.
<code>HuffmanRunLevelTableInitAlloc</code>	Allocates memory and initializes a table that contains codes for DCT coefficients, that is, Run-Level codes.
<code>DecodeHuffmanOne</code>	Decodes a code from the bitstream using a specified table.
<code>DecodeHuffmanPair</code>	Decodes one code using specified table and gets two decoded values.
<code>ReconstructDCTBlock_MPEG1</code>	Decodes 8X8 non-intra block using table of Run-Level codes for standard MPEG1, rearranges and performs inverse quantization.
<code>ReconstructDCTBlockIntra_MPEG1</code>	Decodes 8X8 intra block using table of Run-Level codes for standard MPEG1, rearranges and performs inverse quantization.
<code>ReconstructDCTBlock_MPEG2</code>	Decodes 8X8 non-intra block using table of Run-Level codes for standard MPEG2, rearranges and performs inverse quantization.
<code>ReconstructDCTBlockIntra_MPEG2</code>	Decodes 8X8 intra block using table of Run-Level codes for standard MPEG2, rearranges and performs inverse quantization.
<code>HuffmanTableFree</code>	Frees memory allocated for VLC table.
<b>Inverse Quantization</b>	
<code>QuantInvIntra_MPEG2</code>	Performs inverse quantization for intra frames according to MPEG-2 standard.
<code>QuantInv_MPEG2</code>	Performs inverse quantization for non-intra frames according to MPEG-2 standard.
<b>Inverse Discrete Cosine Transformation</b>	
<code>DCT8x8Inv_AANTransposed_16s_C1R</code>	Performs inverse DCT on pre-transposed block.
<code>DCT8x8Inv_AANTransposed_16s8u_C1R</code>	Performs inverse DCT on pre-transposed block and converts output to unsigned char format.

Function Short Name	Description
<a href="#">DCT8x8Inv_AANTransposed_16s_P2C2R</a>	Performs inverse DCT on pre-transposed data of two input chroma blocks and joins the output data into one array.
<a href="#">DCT8x8Inv_AANTransposed_16s8u_P2C2R</a>	Performs inverse DCT on pre-transposed data of two input chroma blocks and joins the output data into one unsigned char array.
<a href="#">DCT8x8InvOrSet</a>	Performs inverse DCT on an 8x8 block.
<b>Motion Compensation</b>	
<a href="#">MC16x16</a>	Performs motion compensation for a predicted 16X16 block.
<a href="#">MC16x8</a>	Performs motion compensation for a predicted 16X8 block.
<a href="#">MC8x16</a>	Performs motion compensation for a predicted 8X16 block.
<a href="#">MC8x8</a>	Performs motion compensation for a predicted 8X8 block.
<a href="#">MC8x4</a>	Performs motion compensation for a predicted 8X4 block.
<a href="#">MC4x8</a>	Performs motion compensation for a predicted 4X8 block.
<a href="#">MC16x4</a>	Performs motion compensation for predicted UV 16X4 block.
<a href="#">MC16x8UV</a>	Performs motion compensation for predicted UV 16X8 block.
<a href="#">MC16x16B</a>	Performs motion compensation for a bi-predicted 16X16 block.
<a href="#">MC16x8B</a>	Performs motion compensation for a bi-predicted 16X8 block.
<a href="#">MC8x16B</a>	Performs motion compensation for a bi-predicted 8X16 block.
<a href="#">MC8x8B</a>	Performs motion compensation for a bi-predicted 8X8 block.
<a href="#">MC8x4B</a>	Performs motion compensation for a bi-predicted 8X4 block.
<a href="#">MC4x8B</a>	Performs motion compensation for bi-predicted UV 16X4 block.
<a href="#">MC16x8UV</a>	Performs motion compensation for bi-predicted UV 16X8 block.

## MPEG-1 and MPEG-2 Video Encoding Functions

Function Short Name	Description
<b>Motion Estimation</b>	
<a href="#">GetDiff16x16</a>	Evaluates difference between current predicted and reference blocks of 16x16 elements.
<a href="#">GetDiff16x8</a>	Evaluates difference between current predicted and reference blocks of 16x8 elements.
<a href="#">GetDiff8x8</a>	Evaluates difference between current predicted and reference blocks of 8x8 elements.

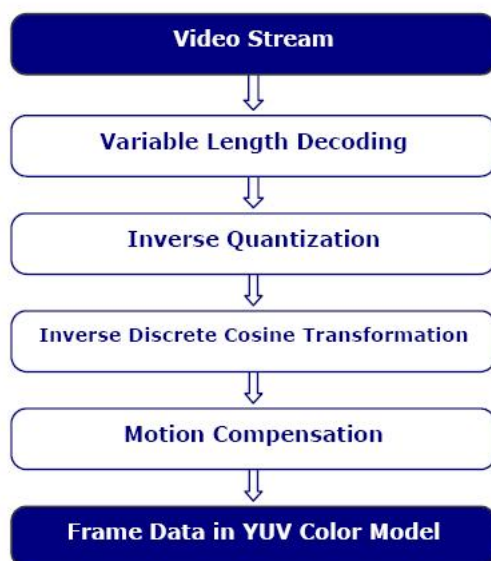
Function Short Name	Description
<a href="#">GetDiff8x16</a>	Evaluates difference between current predicted and reference blocks of 8x16 elements.
<a href="#">GetDiff8x4</a>	Evaluates difference between current predicted and reference blocks of 8x4 elements.
<a href="#">GetDiff16x16B</a>	Evaluates difference between current bi-predicted and mean of two reference blocks of 16x16 elements.
<a href="#">GetDiff16x8B</a>	Evaluates difference between current bi-predicted and mean of two reference blocks of 16x8 elements.
<a href="#">GetDiff8x8B</a>	Evaluates difference between current bi-predicted and mean of two reference blocks of 8x8 elements.
<a href="#">GetDiff8x16B</a>	Evaluates difference between current bi-predicted and mean of two reference blocks of 8x16 elements.
<a href="#">GetDiff8x4B</a>	Evaluates difference between current bi-predicted and mean of two reference blocks of 8x4 elements.
<a href="#">SqrDiff16x16</a>	Evaluates sum of squares of differences between current and reference blocks.
<a href="#">SqrDiff16x16B</a>	Evaluates the sum of squares of differences between the current bi-predicted block and the mean of two reference blocks.
<a href="#">VarMean8x8</a>	Evaluates variance and mean of 8X8 block.
<a href="#">VarMeanDiff16x16</a>	Evaluates variances and means of four 8x8 blocks of difference between two 16x16 blocks.
<a href="#">VarMeanDiff16x8</a>	Evaluates variances and means of four 8x8 blocks of difference between two 16x8 blocks.
<a href="#">SAD16x16</a>	Evaluates sum of absolute difference between current and reference blocks.
<b>Quantization</b>	
<a href="#">QuantIntra_MPEG2</a>	Performs quantization on DCT coefficients for intra blocks in-place with specified quantization matrix according to MPEG-2 standard.
<a href="#">Quant_MPEG2</a>	Performs quantization on DCT coefficients for non-intra blocks in-place with specified quantization matrix according to MPEG-2 standard.
<b>Forward Discrete Cosine Transformation</b>	
<a href="#">DCT8x8Fwd</a>	Performs forward DCT on an 8x8 block of a chrominance part of NV12 plane.
<b>Huffman Encoding</b>	
<a href="#">CreateRLEncodeTable</a>	Creates Run-Level Encode Table.
<a href="#">PutIntraBlock</a>	Encodes, rearranges and puts intra block into the bit stream.
<a href="#">PutNonIntraBlock</a>	Encodes, rearranges and puts non-intra block into the bit stream.

## Video Data Decoding

This section describes principal steps of MPEG-1 and MPEG-2 video decoding in accordance with the standard decoding pipeline shown in the figure below.

### Standard Decoding Pipeline

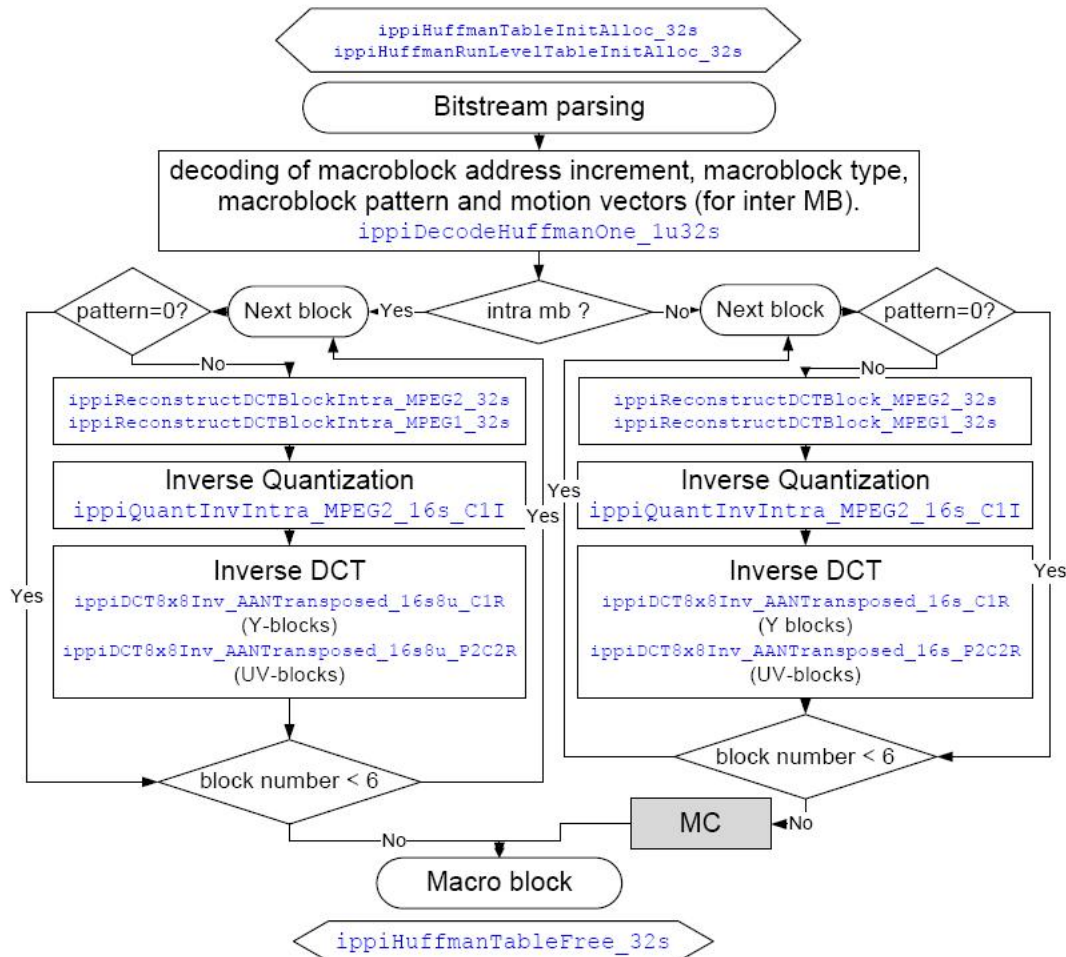
---





**NOTE.** The inverse DCT is one of the most common transformations. It may be performed using Ipp image processing functions: `ippiDCT8x8Inv_16s_C1R` for 8x8 blocks of non-intra type and `ippiDCT8x8Inv_16s8u_C1R` for 8x8 blocks of intra type.

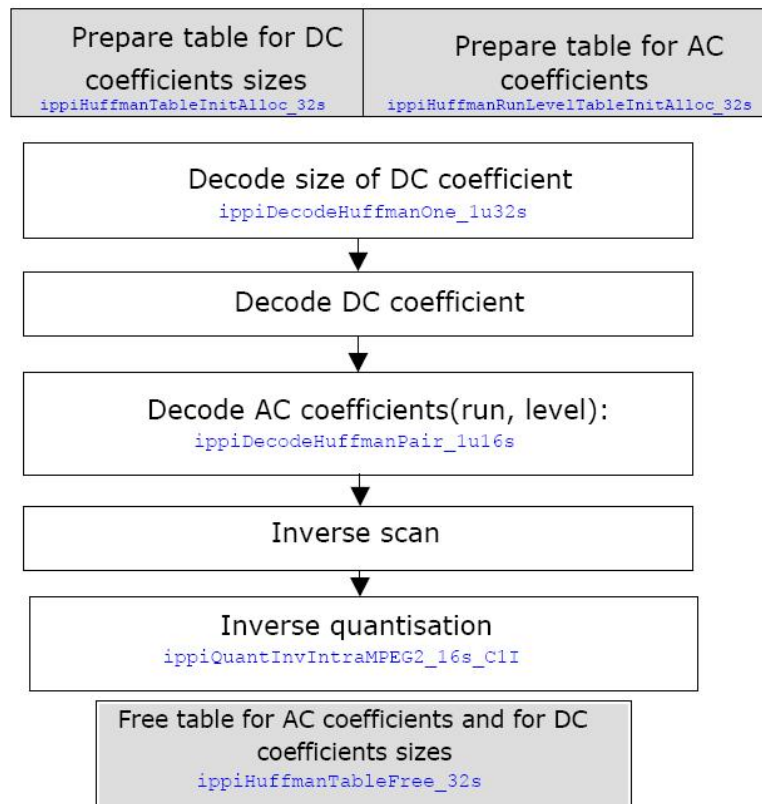
### Macroblock Decoding Scheme



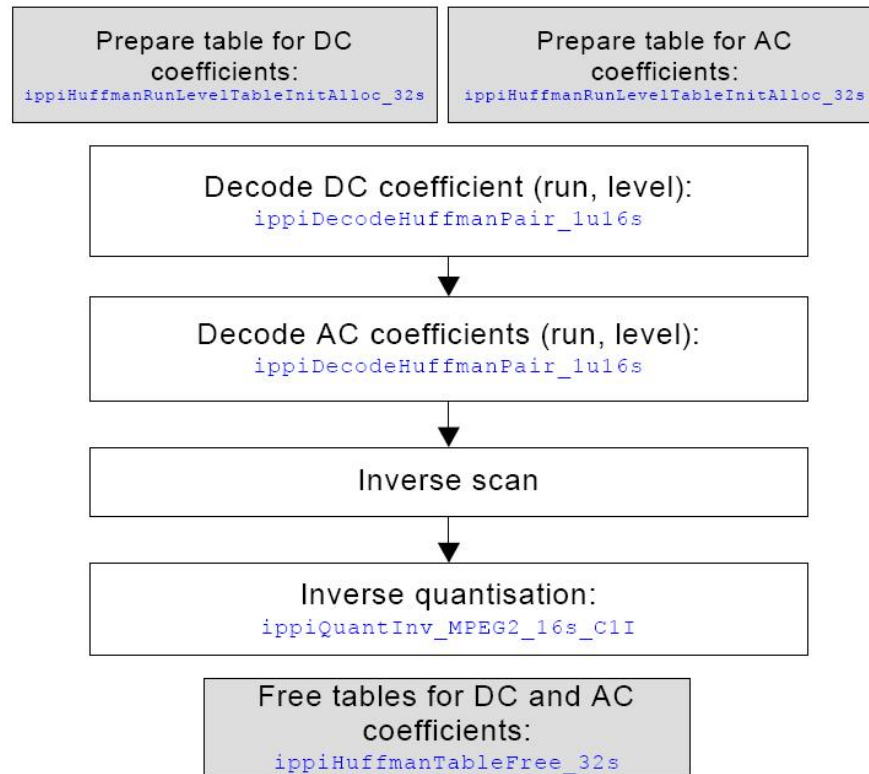


**NOTE.** High-level functions `ippiReconstructDCTBlockIntra_MPEG2_32s` and `ippiReconstructDCTBlockIntra_MPEG1_32s` may be replaced by low-level functions according to the scheme presented in [Figure "DCT Intra Block Reconstruction"](#) and high-level functions `ippiReconstructDCTBlock_MPEG2_32s` and `ippiReconstructDCTBlock_MPEG1_32s` may be replaced by low-level functions according to the scheme presented in [Figure "DCT Non-Intra Block Reconstruction"](#).

### DCT Intra Block Reconstruction



## DCT Non-Intra Block Reconstruction



## Variable Length Decoding

Video data in the bitstream is encoded with Variable Length Code (VLC) tables so that the shortest codes correspond to the most frequent values and the longer codes correspond to the less frequent values. MPEG standard tables contain possible codes and their values.

## ReconstructDCTBlock\_MPEG1

*Decodes 8X8 non-intra block using table of Run-Level codes for standard MPEG1, rearranges and performs inverse quantization.*

### Syntax

```

IppStatus ippiReconstructDCTBlock_MPEG1_32s(Ipp32u** ppBitStream, int * pOffset, const
Ipp32s* pDCSizeTable, const Ipp32s* pACTable, Ipp32s* pScanMatrix, int QP, Ipp16s*
pQPMatrix, Ipp16s* pDstBlock, Ipp32s* pDstSize);
  
```



## Parameters

<i>ppBitStream</i>	Double pointer to the current position in the bit stream.
<i>pOffset</i>	Pointer to the offset between the bit that <i>ppBitStream</i> points to and the start of the code.
<i>pDCSizeTable</i>	Pointer to the table containing codes for DC coefficient, that is, the first of the DCT coefficients.
<i>pACTable</i>	Pointer to the table containing Run-Level codes for all DCT coefficients except the first one.
<i>pScanMatrix</i>	Pointer to the matrix containing indices of elements in scanning sequence.
<i>QP</i>	Quantizer scale factor read from the bit stream.
<i>pQPMatrix</i>	Pointer to the weighting matrix imposed by standard or user-defined.
<i>pDstBlock</i>	Pointer to decoded elements.
<i>pDstSize</i>	Position of the last non-zero block coefficient in scanning sequence.

## Description

This function is declared in the `ippvc.h` header file. The function `ippiReconstructDCTBlock_MPEG1_32s` is applied to predicted and bi-predicted blocks and processes all the DCT coefficients in block using the tables of Run-Level codes: *pDCSizeTable* for DC coefficient and *pACTable* for AC coefficients.

The function uses the tables derived with `HuffmanRunLevelTableInitAlloc` function.

The pointer *pScanMatrix* points to the scanning matrix described in MPEG standard. [Figure "Scanning Matrix and Sequence"](#) illustrates a simple matrix and a sequence. After decoding, data is rearranged from scanning sequence to linear sequence. Then inverse quantization is performed: each of the DCT coefficients is multiplied by a corresponding value from the weighting matrix *pQPMatrix* and by the quantizer scale factor, which are read from the bit stream.

The function decodes the block of 8x8 DCT coefficients. The result is sent to *pDstBlock* and *\*pDstSize* is the position of the last non-zero coefficient. After decoding, the pointers to code in the bitstream are reset as shown in [Figure "Getting One Code From Bitstream"](#).

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when at least one input pointer is NULL.
<code>ippStsH263VLCCodeErr</code>	Indicates an error when decoding in accordance with H263 standard.

## ReconstructDCTBlockIntra\_MPEG1

*Decodes 8X8 intra block using table of Run-Level codes for standard MPEG1, rearranges and performs inverse quantization.*

---

### Syntax

```
IppStatus ippiReconstructDCTBlockIntra_MPEG1_32s(Ipp32u** ppBitStream, int * pOffset,
const Ipp32s* pDCSizeTable, const Ipp32s* pACTable, Ipp32s* pScanMatrix, int QP, Ipp16s*
pQPMatrix, Ipp16s* pDCPred, Ipp16s* pDstBlock, Ipp32s* pDstSize);
```

### Parameters

<i>ppBitStream</i>	Double pointer to the current position in the bit stream.
<i>pOffset</i>	Pointer to the offset between the bit that <i>ppBitStream</i> points to and the start of the code.
<i>pDCSizeTable</i>	Pointer to the table containing codes for DC coefficient, that is, the first of the DCT coefficients.
<i>pACTable</i>	Pointer to the table containing Run-Level codes for all DCT coefficients except the first one.
<i>pScanMatrix</i>	Pointer to the scanning matrix imposed by standard or user-defined.
<i>QP</i>	Quantizer scale factor read from the bit stream.
<i>pQPMatrix</i>	Pointer to the weighting matrix imposed by standard or user-defined.
<i>pDCPred</i>	Pointer to the value to be added to the DC coefficient. This value is read from the special table imposed by standard.
<i>pDstBlock</i>	Pointer to the decoded elements.
<i>pDstSize</i>	Position of the last non-zero block coefficient in scanning sequence.

### Description

This function is declared in the `ippvc.h` header file. The function `ippiReconstructDCTBlockIntra_MPEG1_32s` is applied to intra blocks and processes all the DCT coefficients in block using *pACTable*, which contains Run-Level codes for AC coefficients, except for DC coefficient that is to be processed using *pDCTable*.

The function uses the tables derived with [HuffmanTableInitAlloc](#) and [HuffmanRunLevelTableInitAlloc](#) functions.

The pointer *pScanMatrix* points to the scanning matrix described in MPEG standard. [Figure "Scanning Matrix and Sequence"](#) illustrates a simple matrix and a sequence. After decoding, data is rearranged from scanning sequence to linear sequence. Then inverse quantization is performed: each of the DCT coefficients is multiplied by a corresponding value from the weighting matrix *pQPMatrix* and by the quantizer scale factor, which are read from the bit stream.

The DC coefficient is increased by *pDCPred*. After performing the function, the *pDCPred* stores the sum of its initial value and DC coefficient.

The function decodes the block of 8x8 DCT coefficients. The result is sent to *pDstBlock* and *\*pDstSize* is the position of the last non-zero coefficient. After decoding, the pointers to code in the bitstream are reset as shown in [Figure "Getting One Code From Bitstream"](#).

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when at least one input pointer is <code>NULL</code> .
<code>ippStsH263VLCCodeErr</code>	Indicates an error when decoding in accordance with H263 standard.

## ReconstructDCTBlock\_MPEG2

*Decodes 8X8 non-intra block using table of Run-Level codes for standard MPEG2, rearranges and performs inverse quantization.*

### Syntax

```
IppStatus ippReconstructDCTBlock_MPEG2_32s(Ipp32u** ppBitStream, int * pOffset, const
IppVCHuffmanSpec_32s* pDCTable, const IppVCHuffmanSpec_32s* pACTable, Ipp32s* pScanMatrix,
int QP, Ipp16s* pQPMatrix, Ipp16s* pDstBlock, Ipp32s* pDstSize);
```

### Parameters

<code>ppBitStream</code>	Double pointer to the current position in the bit stream.
<code>pOffset</code>	Pointer to the offset between the bit that <code>ppBitStream</code> points to and the start of the code.
<code>pDCTable</code>	Pointer to the table containing codes for DC coefficient, that is, the first of the DCT coefficients.
<code>pACTable</code>	Pointer to the table containing Run-Level codes for all DCT coefficients except the first one.
<code>pScanMatrix</code>	Pointer to the matrix containing indices of elements in scanning sequence.
<code>QP</code>	Quantizer scale factor read from the bit stream.
<code>pQPMatrix</code>	Pointer to the weighting matrix imposed by standard or user-defined; may be <code>NULL</code> .
<code>pDstBlock</code>	Pointer to decoded elements.
<code>pDstSize</code>	Position of the last non-zero block coefficient in scanning sequence.

### Description

This function is declared in the `ippvc.h` header file. The function `ippReconstructDCTBlock_MPEG2_32s` decodes an 8x8 block using tables of Run-Level codes, rearranges the block and performs inverse quantization. The function is applied to predicted and bi-predicted blocks and processes all the DCT coefficients in block.

The function uses the tables derived with [HuffmanRunLevelTableInitAlloc](#) function.

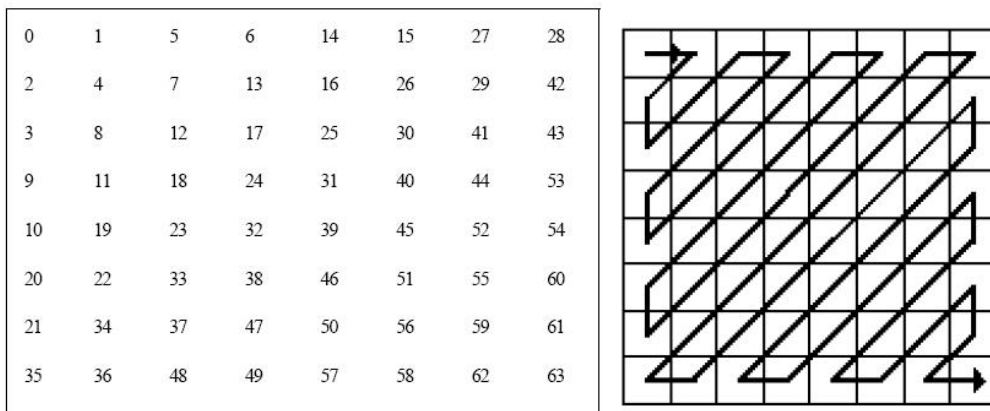
The function decodes the block of 8x8 DCT coefficients. Simultaneous processing of the coefficients enhances performance. The result is sent to `pDstBlock` and `* pDstSize` is the position of the last non-zero coefficient. After decoding, the pointers to code in the bitstream are reset as shown in [Figure "Getting One Code From Bitstream"](#).

The pointer *pScanMatrix* points to one of the pre-defined scanning matrices, depending on the bitstream. [Figure "Scanning Matrix and Sequence"](#) illustrates a simple matrix and a sequence. After decoding, data is rearranged from scanning sequence to linear sequence.

Then inverse quantization is performed: each of the DCT coefficients is multiplied by a corresponding value from the weighting matrix *pQPMatrix* and by the quantizer scale factor, which are read from the bit stream. If *pQPMatrix* is NULL, the default matrix *non\_intra\_quantizer\_matrix* is used, which is described in subclause 6.3.11 of [ISO13818-2].

This function is used in the MPEG-2 decoder included into Intel IPP Samples. See [introduction](#) to this section.

### Scanning Matrix and Sequence



### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when at least one input pointer is NULL.
<code>ippStsH263VLCCodeErr</code>	Indicates an error when decoding in accordance with the H263 standard.

## ReconstructDCTBlockIntra\_MPEG2

*Decodes 8X8 intra block using table of Run-Level codes for standard MPEG2, rearranges and performs inverse quantization.*

### Syntax

```
ippStatus ippiReconstructDCTBlockIntra_MPEG2_32s(Ipp32u** ppBitStream, int * pOffset,
const IppVCHuffmanSpec_32s* pDCSizeTable, const IppVCHuffmanSpec_32s* pACTable, Ipp32s*
pScanMatrix, int QP, Ipp16s* pQPMatrix, Ipp16s* pDCPred, Ipp32s shiftDCVal, Ipp16s*
pDstBlock, Ipp32s* pDstSize);
```

## Parameters

<i>ppBitStream</i>	Double pointer to the current position in the bit stream.
<i>pOffset</i>	Pointer to the offset between the bit that <i>ppBitStream</i> points to and the start of the code.
<i>pDCSizeTable</i>	Pointer to the table containing codes for DC coefficient, that is, the first of the DCT coefficients.
<i>pACTable</i>	Pointer to the table containing Run-Level codes for all DCT coefficients except the first one.
<i>pScanMatrix</i>	Pointer to the scanning matrix imposed by standard or user-defined.
<i>QP</i>	Quantizer scale factor read from the bit stream.
<i>pQPMatrix</i>	Pointer to the weighting matrix imposed by standard or user-defined.
<i>pDCPred</i>	Pointer to the value to be added to the DC coefficient. This value is read from the special table imposed by standard.
<i>shiftDCVal</i>	Integer value. The DC coefficient must be multiplied by $2^{\text{shiftDCVal}}$ .
<i>pDstBlock</i>	Pointer to the decoded elements.
<i>pDstSize</i>	Position of the last non-zero block coefficient in scanning sequence.

## Description

This function is declared in the `ippvc.h` header file. The function `ippiReconstructDCTBlockIntra_MPEG2_32s` decodes an 8x8 intra block using tables of Run-Level codes, rearranges the block and performs inverse quantization. The function is applied to intra blocks and processes all the DCT coefficients in block, except for DC coefficient that is to be processed using the table *pDCSizeTable*.

The function uses the tables derived with `HuffmanTableInitAlloc` and `HuffmanRunLevelTableInitAlloc` functions.

The function decodes the block of 8x8 DCT coefficients. Simultaneous processing of the coefficients enhances performance. The result is sent to *pDstBlock* and *\*pDstSize* is the position of the last non-zero coefficient. After decoding, the pointers to code in the bitstream are reset as shown in [Figure "Getting One Code From Bitstream"](#).

The pointer *pScanMatrix* points to the scanning matrix that is read from the bit stream. [Figure "Scanning Matrix and Sequence"](#) illustrates a simple matrix and a sequence. After decoding, data is rearranged from scanning sequence to linear sequence.

Then the inverse quantization is performed: each of the DCT coefficients is multiplied by a corresponding value from the weighting matrix *pQPMatrix* and by the quantizer scale factor, which are read from the bit stream.

This function is used in the MPEG-2 decoder included into Intel IPP Samples. See [introduction](#) to this section.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when at least one input pointer is <code>NULL</code> .
<code>ippStsH263VLCCodeErr</code>	Indicates an error when decoding in accordance with the H263 standard.

## Inverse Quantization

Each of the decoded DCT coefficients in block should be inverse quantized through multiplication by the corresponding value from the weighting matrix and by the quantizer scale factor, which are read from the bit stream. Before this operation is performed, the DCT coefficients should be rearranged from zigzag scanning sequence to linear sequence.

For intra-frames all the DCT coefficients in block are processed, except the DC coefficient that should be processed separately in decoder, as described in MPEG standard. The functions for non-intra frames process all the DCT coefficients in block.

## QuantInvIntra\_MPEG2

*Performs inverse quantization for intra frames according to MPEG-2 standard.*

---

### Syntax

```
ippStatus ippiQuantInvIntra_MPEG2_16s_C1I(Ipp16s* pSrcDst, int QP, Ipp16s* pQPMatrix);
```

### Parameters

<i>pSrcDst</i>	Pointer to the start of the block.
<i>QP</i>	Quantizer scale factor read from the bit stream.
<i>pQPMatrix</i>	Pointer to the weighting matrix imposed by the MPEG standard or user-defined.

### Description

This function is declared in the `ippvc.h` header file. The function `ippiQuantInvIntra_MPEG2_16s_C1I` multiplies the DCT coefficients from *pSrcDst* by *QP* and by corresponding values from *pQPMatrix* and sends the result back to *pSrcDst*.

This function is used in the MPEG-2 encoder included into Intel IPP Samples. See [introduction](#) to this section.

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when at least one input pointer is NULL.

## QuantInv\_MPEG2

*Performs inverse quantization for non-intra frames according to MPEG-2 standard.*

---

### Syntax

```
ippStatus ippiQuantInv_MPEG2_16s_C1I(Ipp16s* pSrcDst, int QP, Ipp16s* pQPMatrix);
```

### Parameters

<i>pSrcDst</i>	Pointer to the start of the block.
----------------	------------------------------------

<i>QP</i>	Quantizer read from the bit stream.
<i>pQPMatrix</i>	Pointer to the quantizing matrix imposed by MPEG standard or user-defined.

## Description

This function is declared in the `ippvc.h` header file. The function `ippiQuantInv_MPEG2_16s_C1I` multiplies the DCT coefficients from *pSrcDst* by *QP* and by corresponding values from *pQPMatrix* and sends the result back to *pSrcDst*.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when at least one input pointer is <code>NULL</code> .

## Inverse Discrete Cosine Transformation

### DCT8x8Inv\_AANTransposed\_16s\_C1R

*Performs inverse DCT on pre-transposed block.*

## Syntax

```
ippiDCT8x8Inv_AANTransposed_16s_C1R(const Ipp16s* pSrc, Ipp16s* pDst, Ipp32s dstStep,
Ipp32s count);
```

## Parameters

<i>pSrc</i>	Pointer to the block of DCT coefficients.
<i>pDst</i>	Pointer to the destination array.
<i>dstStep</i>	Distance in bytes between starts of the consecutive lines in the destination image.
<i>count</i>	Number of the last non-zero coefficient in zig-zag order. If the block contains no non-zero coefficients, pass the value -1.

## Description

This function is declared in the `ippvc.h` header file. The function `ippiDCT8x8Inv_AANTransposed_16s_C1R` is used for non-intra macroblocks and performs inverse DCT on a transposed block. The block is transposed during the rearranging stage of VLC decoding, using the transposed scanning matrix as *scanMatrix* argument of the functions [ReconstructDCTBlock\\_MPEG1](#) and [ReconstructDCTBlock\\_MPEG2](#).

This function is used in the MPEG-2 decoder included into Intel IPP Samples. See [introduction](#) to this section.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when at least one input pointer is <code>NULL</code> .

## DCT8x8Inv\_AANTransposed\_16s8u\_C1R

*Performs inverse DCT on pre-transposed block and converts output to unsigned char format.*

---

### Syntax

```
ippiDCT8x8Inv_AANTransposed_16s8u_C1R(const Ipp16s* pSrc, Ipp8u* pDst, Ipp32s dstStep,
Ipp32s count);
```

### Parameters

<i>pSrc</i>	Pointer to the block of DCT coefficients.
<i>pDst</i>	Pointer to the destination array.
<i>dstStep</i>	Distance in bytes between starts of the consecutive lines in the destination image.
<i>count</i>	Number of the last non-zero coefficient in zig-zag order. If the block contains no non-zero coefficients, pass the value -1.

### Description

This function is declared in the `ippvc.h` header file. The function `ippiDCT8x8Inv_AANTransposed_16s8u_C1R` is used for intra macroblocks. The function performs inverse DCT on a transposed block and converts the output to unsigned char format. The block is transposed during the rearranging stage of VCL decoding, using the transposed scanning matrix as *scanMatrix* argument of the functions [ReconstructDCTBlockIntra\\_MPEG1](#) and [ReconstructDCTBlockIntra\\_MPEG2](#).

This function is used in the MPEG-2 decoder included into Intel IPP Samples. See [introduction](#) to this section.

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when at least one input pointer is NULL.

## DCT8x8Inv\_AANTransposed\_16s\_P2C2R

*Performs inverse DCT on pre-transposed data of two input chroma blocks and joins the output data into one array.*

---

### Syntax

```
ippiDCT8x8Inv_AANTransposed_16s_P2C2R(const Ipp16s* pSrcU, const Ipp16s* pSrcV, Ipp16s*
pDstUV, Ipp32s dstStep, Ipp32s countU, Ipp32s countV);
```

### Parameters

<i>pSrcU</i>	Pointer to the block of DCT coefficients for <i>U</i> component.
<i>pSrcV</i>	Pointer to the block of DCT coefficients for <i>V</i> component.
<i>pDstUV</i>	Pointer to the destination array.



<i>dstStep</i>	Distance in bytes between starts of the consecutive lines in the destination image.
<i>countU</i>	Number of the last non-zero <i>U</i> coefficient in zig-zag order. If the block contains no non-zero coefficients, pass the value -1.
<i>countV</i>	Number of the last non-zero <i>V</i> coefficient in zig-zag order. If the block contains no non-zero coefficients, pass the value -1.

## Description

This function is declared in the `ippvc.h` header file. The function `ippiDCT8x8Inv_AANTransposed_16s_P2C2R` is used for non-intra macroblocks. The function performs inverse DCT on a transposed *U*-block and a transposed *V*-block and joins the output data into one [UV Block](#). The blocks are transposed during the rearranging stage of VCL decoding, using the transposed scanning matrix as *scanMatrix* argument of the functions [ReconstructDCTBlock\\_MPEG1](#) and [ReconstructDCTBlock\\_MPEG2](#).

This function is used in the MPEG-2 decoder included into Intel IPP Samples. See [introduction](#) to this section.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when at least one input pointer is NULL.

## DCT8x8Inv\_AANTransposed\_16s8u\_P2C2R

*Performs inverse DCT on pre-transposed data of two input chroma blocks and joins the output data into one unsigned char array.*

## Syntax

```
ippiDCT8x8Inv_AANTransposed_16s8u_P2C2R(const Ipp16s* pSrcU, const Ipp16s* pSrcV, Ipp8u* pDstUV, Ipp32s dstStep, Ipp32s countU, Ipp32s countV);
```

## Parameters

<i>pSrcU</i>	Pointer to the block of DCT coefficients for <i>U</i> component.
<i>pSrcV</i>	Pointer to the block of DCT coefficients for <i>V</i> component.
<i>pDstUV</i>	Pointer to the destination array.
<i>dstStep</i>	Distance in bytes between starts of the consecutive lines in the destination image.
<i>countU</i>	Number of the last non-zero <i>U</i> coefficient in zig-zag order. If the block contains no non-zero coefficients, pass the value -1.
<i>countV</i>	Number of the last non-zero <i>V</i> coefficient in zig-zag order. If the block contains no non-zero coefficients, pass the value -1.

## Description

This function is declared in the `ippvc.h` header file. The function `ippiDCT8x8Inv_AANTransposed_16s8u_P2C2R` is used for intra macroblocks. The function performs inverse DCT on a transposed U-block and a transposed V-block and joins the output data into one unsigned char **UV Block**. The blocks are transposed during the rearranging stage of VCL decoding, using the transposed scanning matrix as *scanMatrix* argument of the functions `ReconstructDCTBlockIntra_MPEG1` and `ReconstructDCTBlockIntra_MPEG2`.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when at least one input pointer is NULL.

## DCT8x8InvOrSet

*Performs inverse DCT on an 8x8 block.*

---

## Syntax

```
ippStatus ippiDCT8x8InvOrSet_16s8u_P2C2(const Ipp8u* pSrcU, const Ipp16s* pSrcV, Ipp8u*
pDst, Ipp32s dstStep, Ipp32u flag);
```

## Parameters

<i>pSrcU</i>	Pointer to the source of U(Cb) DCT coefficients
<i>pSrcV</i>	Pointer to the source of V(Cr) DCT coefficients
<i>pDst</i>	Pointer to the destination NV12 plane
<i>dstStep</i>	Step of the current destination block; specifies width of the plane in bytes
<i>flag</i>	Flag that defines function operation. Bits from 2 to 31 are reserved; considers only two low bits. <div style="margin-left: 20px;"> <math>(flag \&amp; 0x03) == 0x00</math> Do inverse DCT for U and V  <math>(flag \&amp; 0x03) == 0x01</math> Do inverse DCT for U and SET for V  <math>(flag \&amp; 0x03) == 0x02</math> Do SET for U and inverse DCT for V  <math>(flag \&amp; 0x03) == 0x03</math> Do SET for U and V            SET means that the destination block is filled with <math>(pSrc[0][0]/8)</math> value for U or <math>(pSrc[1][0]/8)</math> value for V.         </div>

## Description

The function `ippiDCT8x8InvOrSet_16s8u_P2C2` is declared in the `ippvc.h` file.

This function performs an inverse discrete cosine transform on an 8x8 block and places the results into the chrominance part of NV12 plane. Otherwise, depending on the *flag* parameter value, the function fills the chrominance part of NV12 plane with constant values.

**NV12 Plane:**

YY YY YY YY  
 YY YY YY YY  
 UV UV UV UV - chrominance part of NV12 plane.

**Return Values**

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if any of the pointers is NULL.
<code>ippStsStepErr</code>	Indicates an error condition if the step value is negative.

**Example**

The code example below illustrates the usage of `ippiDCT8x8InvOrSet_16s8u_P2C2` function.

```
{
    // pSrcU - the pointer to the source of U(Cb) DCT coefficients.
    // pSrcV - the pointer to the source of V (Cr) DCT coefficients.
    const Ipp16s *pSrcU = ptr_to_source_U_block;
    const Ipp16s *pSrcV = ptr_to_source_V_block;
    // pDst - the pointer to the destination NV12 plane.
    Ipp8u* pDst = ptr_to_dst_plane;
    // dstStep - Step of the current destination block, specifying width of the plane in bytes.
    Ipp32s dstStep = pitch_of_dst;
    // flag - Bits from 2 to 31 are reserved. Take into account only two low bits
    // (flag&0x03) == 0x00 - Do iDCT for U and V buffers
    Ipp32u flag = 0;
    ippiDCT8x8InvOrSet_16s8u_P2C2(pSrcU,pSrcV,
                                   pDst,dstStep,flag );
}
```

**Motion Compensation**

This operation is applied to each block of non-intra type. The values of data obtained after inverse DCT are added to the data of predicted block and sent to the destination block. Prediction for half-pixel interpolation is performed by averaging of neighboring pixels with rounding. After processing, the data is converted from `Ipp16s` to `Ipp8u` type.

These functions are divided into two groups:

- Functions for predicted block (for block of P-type)
- Functions for bi-predicted block (for block of B-type).

See the detailed descriptions of these functions in [General Functions](#) section.

The table below lists MC functions used in Video Data decoding.

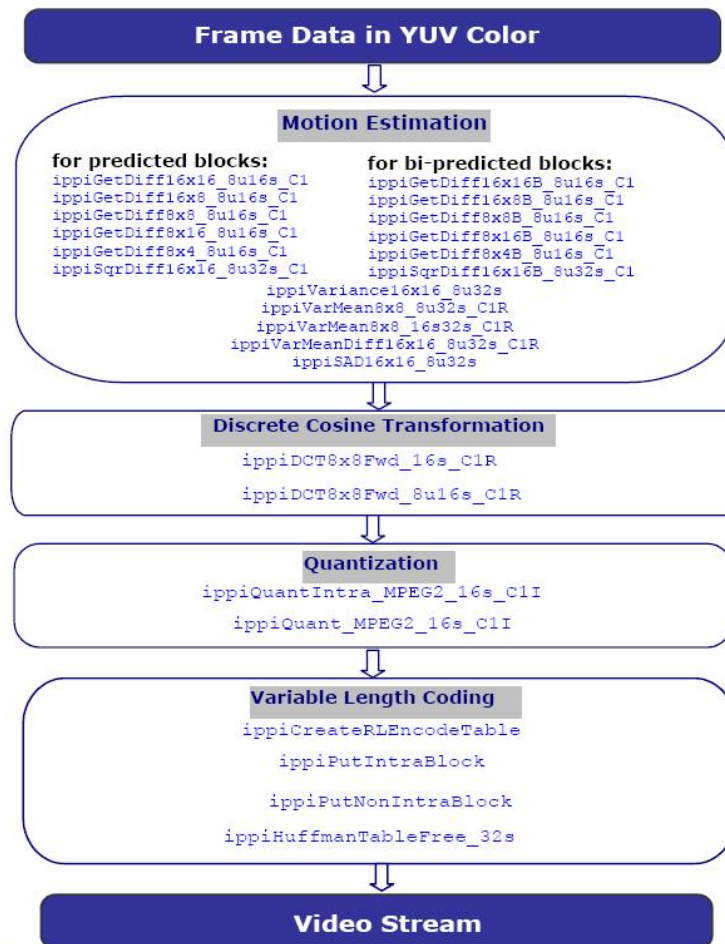
**Motion Compensation Functions**

for predicted blocks:	for bi-predicted blocks:
<a href="#">MC16x4</a>	<a href="#">MC16x4B</a>
<a href="#">MC16x16</a>	<a href="#">MC16x16B</a>
<a href="#">MC16x8</a>	<a href="#">MC16x8B</a>
<a href="#">MC16x8UV</a>	<a href="#">MC16x8UVB</a>
<a href="#">MC8x16</a>	<a href="#">MC8x16B</a>
<a href="#">MC8x8</a>	<a href="#">MC8x8B</a>
<a href="#">MC8x4</a>	<a href="#">MC8x4B</a>

## Video Data Encoding

This section describes the main steps of MPEG video encoding according to the standard encoding pipeline shown in the figure below:

### Encoding Pipeline



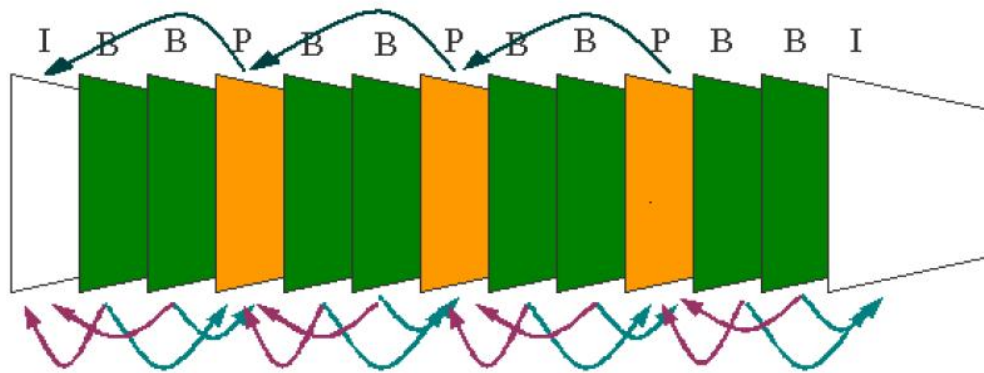
Note that for I and P frames after the Quantization phase reconstruction may be implemented through inverse quantization ([QuantInv\\_MPEG2](#)), inverse discrete cosine transformation ([ippiDCT8x8Inv\\_16s\\_C1R](#), see [DCT8x8Inv](#), [DCT8x8Inv\\_A10](#)), and, for non-intra macroblocks, motion compensation in the same way as in the decoder (see [Figure "Macroblock Decoding Scheme"](#)). The result is further used for motion estimation and compensation.

## Motion Estimation

The process of encoding starts with motion estimation (see [Figure "Motion Compensation Scheme for Predicted Blocks"](#) and [Figure "Motion Compensation Scheme for Bi-Predicted Blocks"](#)). Block *SrcRef* is found for each non-intra block *Dst* to be predicted, or two blocks *SrcRefF* and *SrcRefB* are found if block *Dst* is bi-predicted. The reference blocks are supposed to be similar to block *Dst*.

The figure below shows how reference frames are selected.

### Intra, Predicted and Bi-Predicted Frames With Forward and Backward References



The motion estimation process is performed by the following general Motion Estimation functions:

#### MPEG Video Motion Estimation Functions

##### Evaluation of difference between current predicted and reference blocks

For predicted blocks

`GetDiff16x16`

`GetDiff16x8`

`GetDiff8x8`

`GetDiff8x16`

`GetDiff8x4`

For bi-predicted blocks

`GetDiff16x16B`

`GetDiff16x8B`

`GetDiff8x8B`

`GetDiff8x16B`

`GetDiff8x4B`

##### Evaluation of sum of squares of differences between current and reference blocks

For predicted blocks

`SqrDiff16x16`

For bi-predicted blocks

`SqrDiff16x16B`

##### Evaluation of variance and mean of 8x8 block

`VarMean8x8`

##### Evaluation of variances and means of difference between two blocks

`VarMeanDiff16x16`

`VarMeanDiff16x8`

##### Evaluation of variance of current block

`Variance16x16`

##### Evaluation of sum of absolute difference between current and reference blocks

`SAD16x16`

[SAD16x8](#)[SAD8x8](#)[SAD4x4](#)

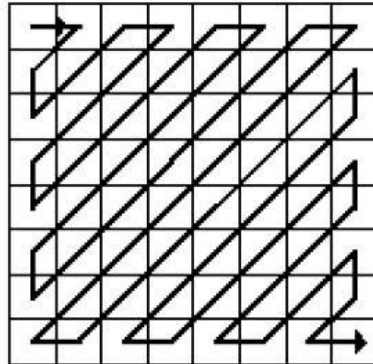
---

## Quantization

Quantization is applied to DCT coefficients to decrease their precision. You may use quantization matrices to vary precision coefficients with different positions. Scale factor is applied to all coefficients in the macroblock in the same way. Quantization values are often inverted to increase performance. Scan matrices specify the way to reorder 8x8 quantized DCT coefficients to an array of 64, so that statistically bigger values are located earlier; the further the element is from the first AC coefficient in the scanning sequence ([Figure "Default Scanning Sequence"](#)) the less important its value is for representing image and for further decoding.

### Default Scanning Sequence

---

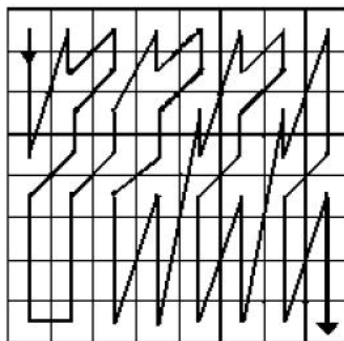


---

The alternate scanning sequence is used for interlaced video and provides better compression for field-coded blocks:

### Alternate Scanning Sequence

---



## QuantIntra\_MPEG2

*Performs quantization on DCT coefficients for intra blocks in-place with specified quantization matrix according to MPEG-2 standard.*

---

### Syntax

```
IppStatus ippiQuantIntra_MPEG2_16s_C1I(Ipp16s* pSrcDst, Ipp32s QP, const Ipp32f* pQPMatrix, Ipp32s* pCount);
```

### Parameters

<i>pSrcDst</i>	Pointer to the block of DCT coefficients.
<i>QP</i>	Quantizer.
<i>pQPMatrix</i>	Pointer to the matrix of inverted quantization coefficients.
<i>pCount</i>	Number of the non-zero AC coefficients after quantization.

### Description

This function is declared in the `ippvc.h` header file. The in-place function `ippiQuantIntra_MPEG2_16s_C1I` multiplies all DCT coefficients in block except DC coefficients by the elements of the inverted quantization matrix and divide them by quantizer. The number of non-zero coefficients after quantization is stored in *pCount* for future considerations. If the pointer *pQPMatrix* is `NULL`, then the default matrix is used.

This function is used in the MPEG-2 encoder included into Intel IPP Samples. See [introduction](#) to this section.

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when at least one input pointer is <code>NULL</code> .
<code>ippStsDivByZeroErr</code>	Indicates an error when quantizer <i>QP</i> is equal to zero.

## Quant\_MPEG2

*Performs quantization on DCT coefficients for non-intra blocks in-place with specified quantization matrix according to MPEG-2 standard.*

---

### Syntax

```
IppStatus ippiQuant_MPEG2_16s_C1I(Ipp16s* pSrcDst, Ipp32s QP, const Ipp32f* pQPMatrix, Ipp32s* pCount);
```

### Parameters

<i>pSrcDst</i>	Pointer to the block of DCT coefficients.
<i>QP</i>	Quantizer.
<i>pQPMatrix</i>	Pointer to the matrix of inverted quantization coefficients.
<i>pCount</i>	Number of the non-zero coefficients after quantization.

## Description

This function is declared in the `ippvc.h` header file. The in-place function `ippiQuant_MPEG2_16s_C1I` multiplies DCT coefficients in block by the elements of the inverted quantization matrix and divide them by quantizer. The number of non-zero coefficients after quantization is stored in `pCount` for future considerations. If the pointer `pQPMatrix` is `NULL`, then the default matrix is used.

This function is used in the MPEG-2 encoder included into Intel IPP Samples. See [introduction](#) to this section.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when at least one input pointer is <code>NULL</code> .
<code>ippStsDivByZeroErr</code>	Indicates an error when quantizer <code>QP</code> is equal to zero.

## Forward Discrete Cosine Transformation

### DCT8x8Fwd

*Performs forward DCT on an 8x8 block of a chrominance part of NV12 plane.*

---

## Syntax

```
ippStatus ippiDCT8x8Fwd_8u16s_C2P2(const Ipp8u* pSrc, Ipp32s srcStep, Ipp16s* pDstU,
Ipp16s* pDstV);
```

## Parameters

<code>pSrc</code>	Pointer to the source image buffer.
<code>srcStep</code>	Distance in bytes between starts of consecutive lines in the source image buffer for operations with ROI.
<code>pDst</code>	Pointer to the destination image buffer.
<code>pSrcDst</code>	Pointer to the source and destination image for in-place operations.

## Description

The function `ippiDCT8x8Fwd_8u16s_C2P2` is declared in the `ippvc.h` file.

This function performs a forward discrete cosine transform on an 8x8 block of a chrominance part of NV12 plane and places the results into separate 8x8 blocks for `U(Cb)` and `V(Cr)` components.

NV12 Plane:

```
YY YY YY YY
YY YY YY YY
UV UV UV UV - chrominance part of NV12 plane.
```

The code example below illustrates the usage of `ippiDCT8x8Fwd_8u16s_C2P2` function.



## ippiDCT8x8Fwd\_8u16s\_C2P2 Usage Example

```
{
    // the pointer to the source block (chrominance part of NV12 plane)
    const Ipp8u *pSrcUV = src_block_ptr;
    // Step of the current source block
    Ipp32s srcStep = src_step;
    // the pointer to the destination buffer for U coefficients of DCT
    Ipp16s dstU[64];
    Ipp16s *pDstU = &dstU[0];
    // the pointer to the destination buffer for V coefficients of DCT
    Ipp16s dstV[64];
    Ipp16s *pDstV = &dstV[0];
    ippiDCT8x8Fwd_8u16s_C2P2(pSrc,srcStep,pDstU,pDstV );
}
```

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if any of the pointers is NULL.
<code>ippStsStepErr</code>	Indicates an error condition if the step value is negative.

## Huffman Encoding Functions

After quantization the DCT coefficients in block should be encoded. Encoding is realized by Variable Length Coding (VLC), which uses standard code tables. These tables are entropy-constrained, that is, non-downloadable and optimized for a limited range of bit rates.

## CreateRLEncodeTable

*Creates Run-Level Encode Table.*

### Syntax

```
IppStatus ippiCreateRLEncodeTable (const Ipp32s* pSrcTable, IppVCHuffmanSpec_32s**
ppDstSpec);
```

### Parameters

<i>pSrcTable</i>	Pointer to the source table.
<i>ppDstSpec</i>	Double pointer to the destination encode table.

### Description

This function is declared in the `ippvc.h` header file. The function `ippiCreateRLEncodeTable` creates the Run-Level Encode Table. The result is stored in block *ppDstSpec*.

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when at least one input pointer is NULL.

## PutIntraBlock

*Encodes, rearranges and puts intra block into bit stream.*

### Syntax

```
IppStatus ippiPutIntraBlock(Ipp32u** ppBitStream, int* pOffset, Ipp16s* pSrcBlock, Ipp32s*
pDCPred, IppVCHuffmanSpec_32u* pDCTable, IppVCHuffmanSpec_32s* pACTable, Ipp32s*
pScanMatrix, Ipp32s EOBLen, Ipp32s EOBCode, Ipp32s count);
```

### Parameters

<i>ppBitStream</i>	Double pointer to the current position in the bit stream.
<i>pOffset</i>	Pointer to the offset between the bit that <i>ppBitStream</i> points to and the start of the code.
<i>pSrcBlock</i>	Pointer to the block.
<i>pDCPred</i>	Pointer to the value to be added to the DC coefficient; this value is read from the special table imposed by standard.
<i>pDCTable</i>	Pointer to the table containing codes for DC coefficient, that is, the first coefficient among the DCT coefficients.
<i>pACTable</i>	Pointer to the table containing Run-Level codes for AC coefficients, that is, all DCT coefficients except the first coefficient.
<i>pScanMatrix</i>	Pointer to the scanning matrix.
<i>EOBLen</i>	Length of the block end code.
<i>EOBCode</i>	Value of the block end code.
<i>count</i>	Number of the non-zero AC coefficients.

### Description

This function is declared in the `ippvc.h` header file. The function `ippiPutIntraBlock` is applied to intra blocks and encodes a block of 8x8 DCT coefficients.

The `ippiPutIntraBlock` function encodes DC coefficient using the value \* *pDCPred* for value computation and *pDCTable* for its encoding. Then it encodes AC coefficients using the *pACTable*.

The pointer *pScanMatrix* points to the scanning matrix that is described in MPEG standard. See [Figure "Default Scanning Sequence"](#) for the simple scanning matrix and sequence. After encoding, the data should be rearranged from linear sequence to scanning sequence.

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when at least one input pointer is <code>NULL</code> .

## PutNonIntraBlock

*Encodes, rearranges and puts non-intra block into bit stream.*

---

### Syntax

```
IppStatus ippiPutNonIntraBlock(Ipp32u** ppBitStream, int* pOffset, Ipp16s* pSrcBlock,
IppVCHuffmanSpec_32s* pACTable, Ipp32s* pScanMatrix, Ipp32s EOBLen, Ipp32s EOBCode, Ipp32s
count);
```

### Parameters

<i>ppBitStream</i>	Double pointer to the current position in the bit stream.
<i>pOffset</i>	Pointer to the offset between the bit that <i>ppBitStream</i> points to and the start of the code.
<i>pSrcBlock</i>	Pointer to the block.
<i>pACTable</i>	Pointer to the table containing Run-Level codes for AC coefficients, that is, all DCT coefficients except the first coefficient.
<i>pScanMatrix</i>	Pointer to the scanning matrix.
<i>EOBLen</i>	Length of the block end code.
<i>EOBCode</i>	Value of the block end code.
<i>count</i>	Number of the non-zero coefficients.

### Description

This function is declared in the `ippvc.h` header file. The function `ippiPutNonIntraBlock` is applied to predicted and bi-predicted non-intra blocks and encodes a block of 8x8 DCT coefficients. The function encodes both DC and AC coefficients using the *pACTable*.

The pointer *pScanMatrix* points to the scanning matrix that is described in MPEG standard. See [Figure "Default Scanning Sequence"](#) for the simple scanning matrix and sequence. After encoding, the data should be rearranged from linear sequence to scanning sequence.

### Return Values

<code>IppStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when at least one input pointer is <code>NULL</code> .

## DV

DV is an international standard for a consumer digital video format.

DV uses a 1/4 inch (6.35mm) metal evaporate tape to record very high quality digital video. The video is sampled at the same rate as D-1, D-5, or Digital Betacam video - 720 pixels per scanline - although the color information is sampled at half the D-1 rate: 4:1:1 in 525-line (NTSC), and 4:2:0 in 625-line (PAL) formats.

DV uses intraframe compression: each compressed frame depends entirely on itself, and not on any data from preceding or following frames. However, it also uses adaptive *interfield* compression; if the compressor detects little difference between the two interlaced fields of a frame, it will compress them together, freeing up

some of the "bit budget" to allow for higher overall quality. In theory, this means that static areas of images will be more accurately represented than areas with a lot of motion; in practice, this can sometimes be observed as a slight degree of "blockiness" in the immediate vicinity of moving objects.

The use of some functions described in this section is demonstrated of Intel® IPP Samples downloadable from <http://www.intel.com/cd/software/products/asm-na/eng/220046.htm> .

The following section discusses the basic DV notions.

## DCT Block

The Y, U, V pixels in one frame shall be divided into *DCT blocks*. All DCT blocks for 625-50 system and DCT blocks except for rightmost DCT blocks in *U* and *V* for 525-60 system are structured with a rectangular area of eight vertical lines and eight horizontal pixels in a frame.

For 525-60 system, the rightmost DCT blocks in *U* and *V* are structured with 16 vertical lines and four horizontal pixels. The rightmost DCT block is reconstructed to eight vertical lines and eight horizontal pixels by moving the lower part of eight vertical lines and four horizontal pixels to the higher part of eight vertical lines and four horizontal pixels.

## DCT Block Structure

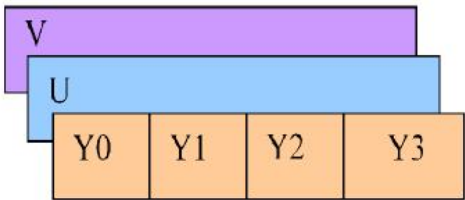
---

		x								
		0	1	2	3	4	5	6	7	
y	0									Field 2
	1									Field 1
	2									Field 2
	3									Field 1
	4									Field 2
	5									Field 1
	6									Field 2
	7									Field 1

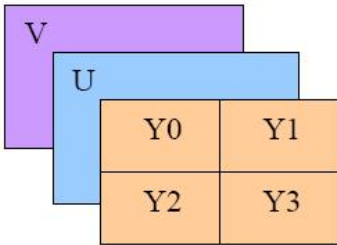
---

Six DCT blocks Y0, Y1, Y2, Y3, U, V form a *macroblock*.

Macroblock Structure for 525-60 System (Except Rightmost Macroblock)



Macroblock Structure for 625-50 System and Rightmost Macroblock of 525-60 System



27 DCT macroblocks form a *superblock*. (Each cell stands for one macroblock).

DCT Superblock Structure for 525-60 System

0	11	12	23	24										8	9	20	21	0	11	12	23	24
1	10	13	22	25										7	10	19	22	1	10	13	22	
2	9	14	21	26										6	11	18	23	2	9	14	21	25
3	8	15	20											0	5	12	17	24	3	8	15	20
4	7	16	19											1	4	13	16	25	4	7	16	19
5	6	17	18											2	3	14	15	26	5	6	17	18

System 525-60 has three types of superblocks.

### DCT Superblock Structure for 625-50 System

0	5	6	11	12	17	18	23	24
1	4	7	10	13	16	19	22	25
2	3	8	9	14	15	20	21	26

All superblocks of 625-50 system are of the same type.

A video *segment* consists of five macroblocks, which are gathered from various areas as below:

### Macroblock Absolute Coordinates

Number of macroblock in the superblock	k	k	k	k	k
Superblock coordinates	$(2, (j+2) \% n)$	$(1, (j+6) \% n)$	$(3, (j+8) \% n)$	$(0, (j) \% n)$	$(4, (j+4) \% n)$

where  $k$  is within the interval  $[0, 26]$ ,  $n = 10$  for 525-60 system,  $n = 12$  for 625-50 system, and  $j$  is the vertical order in the superblock.

Data in a video segment is compressed and transformed to the data of 385 bytes ( *compressed segment* ). A compressed segment consists of five compressed macroblocks. Each compressed macroblock consists of 77 bytes.

### Arrangement of Compressed Macroblock

Byte sequence	H	cbY0	cbY1	cbY2	cbY3	cbU	cbV
Length	4	14	14	14	14	10	10

### Arrangement of Compressed Macroblock Header (H)

Section name	Not used		qn0
Number of bits		4	
Number of bytes	3 bytes		1 byte

### Arrangement of Y compressed blocks cbY0, cbY1, cbY2, cbY3

Section name	D C		m 0	c 1	A C	
Number of bits		1	1	2	4	
Number of bytes	1 byte	1 byte		12 bytes		

### Arrangement Cr compressed blocks cbU, cbV

Section name	D C		m 0	c 1	A C	
Number of bits		1	1	2	4	
Number of bytes	1 byte	1 byte		8 bytes		

Type of block is equal to 1 when the difference between two fields is small ( $m0=0$ ). Type of block is equal to 2 when the difference between two fields is big ( $m0=1$ ).

Each DCT block is classified into four classes. For selecting quantization step, class number is used.

### DV Functions

Function Short Name	Description
<b>Decoding</b>	
<b>Variable Length Decoding</b>	
<a href="#">InitAllocHuffmanTable_DV</a>	Allocates memory and initializes the table that contains codes for DCT coefficients (Run-Level codes).
<a href="#">HuffmanDecodeSegment_DV</a>	Decodes and rearranges a segment block, multiplies the first block element by 2.

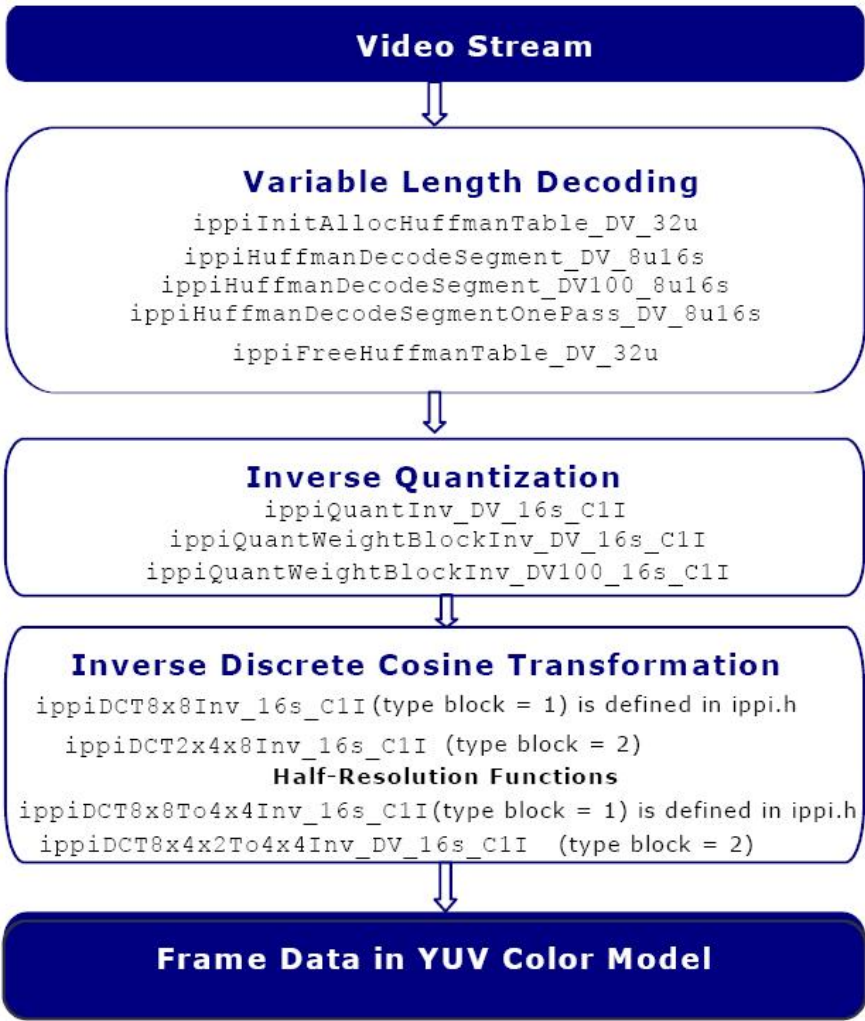
Function Short Name	Description
<a href="#">HuffmanDecodeSegment_DV100</a>	Decodes and rearranges a DV100 segment block, multiplies the first block element by 128.
<a href="#">HuffmanDecodeSegmentOnePass_DV</a>	Performs the first pass of video segment decoding, rearranges the segment block, multiplies the first block element by 128.
<a href="#">FreeHuffmanTable_DV</a>	Frees the memory allocated for VLC table.
<b>Inverse Quantization</b>	
<a href="#">QuantInv_DV</a>	Performs inverse quantization on a block.
<a href="#">QuantWeightBlockInv_DV</a>	Performs inverse quantization and inverse weighting on a block.
<a href="#">QuantWeightBlockInv_DV100</a>	Performs inverse quantization and inverse weighting on a block according to DV100 standard.
<b>Inverse Discrete Cosine Transformation</b>	
<a href="#">DCT2x4x8Inv</a>	Performs the inverse DCT for block of type 2 ( $m0=1$ ).
<a href="#">DCT8x4x2To4x4Inv_DV</a>	Performs the inverse DCT for block 2x4x8 and then creates block 4x4.
<b>Encoding</b>	
<b>Discrete Cosine Transformation</b>	
<a href="#">DCT2x4x8Frw</a>	Performs DCT for a block of type 2.
<a href="#">CountZeros8x8</a>	Evaluates number of zeros in a block.
<b>Color Conversion</b>	
<a href="#">YCrCb411ToYCbCr422_5MBDV</a> , <a href="#">YCrCb411ToY-CbCr422_ZoomOut2_5MBDV</a> , <a href="#">YCrCb411ToY-CbCr422_ZoomOut4_5MBDV</a> , <a href="#">YCrCb411ToY-CbCr422_ZoomOut8_5MBDV</a>	Convert five macroblocks from YUV411 format into YUV2 format.
<a href="#">YCrCb411ToYCbCr422_16x4x5MB_DV</a> , <a href="#">YCrCb411ToY-CbCr422_8x8MB_DV</a>	Convert five reduced YCrCb411 macroblocks into YCrCb422 reduced macroblocks.
<a href="#">YCrCb411ToYCbCr422_EdgeDV</a> , <a href="#">YCrCb411ToY-CbCr422_ZoomOut2_EdgeDV</a> , <a href="#">YCrCb411ToY-CbCr422_ZoomOut4_EdgeDV</a> , <a href="#">YCrCb411ToY-CbCr422_ZoomOut8_EdgeDV</a>	Convert a YCrCb411 macroblock into a YCrCb422 macroblock at the right edge of destination image.
<a href="#">YCrCb420ToYCbCr422_5MBDV</a> , <a href="#">YCrCb420ToY-CbCr422_ZoomOut2_5MBDV</a> , <a href="#">YCrCb420ToY-CbCr422_ZoomOut4_5MBDV</a> , <a href="#">YCrCb420ToY-CbCr422_ZoomOut8_5MBDV</a>	Convert five YCrCb420 macroblocks into YCrCb422 macroblocks.
<a href="#">YCrCb420ToYCbCr422_8x8x5MB_DV</a>	Converts five reduced YCrCb420 macroblocks into YCrCb422 reduced macroblocks.
<a href="#">YCrCb422ToYCbCr422_5MBDV</a> , <a href="#">YCrCb422ToY-CbCr422_ZoomOut2_5MBDV</a> , <a href="#">YCrCb422ToY-CbCr422_ZoomOut4_5MBDV</a> , <a href="#">YCrCb422ToY-CbCr422_ZoomOut8_5MBDV</a>	Convert five YCrCb422 macroblocks into YCbCr422 macroblocks.



Function Short Name	Description
<a href="#">YCrCb422ToYCbCr422_8x4x5MB_DV</a>	Converts five reduced YCrCb422 macroblocks into YCrCb422 reduced macroblocks.
<a href="#">YCrCb422ToYCbCr422_10HalvesMB16x8_DV100</a>	Converts ten YCrCb422 half-macroblocks into YCrCb422 half-macroblocks.

DV Decoding Functions

DV Decoding Pipeline



Variable Length Decoding

Structure of the source tables must be as follows:

## Source Table Structure

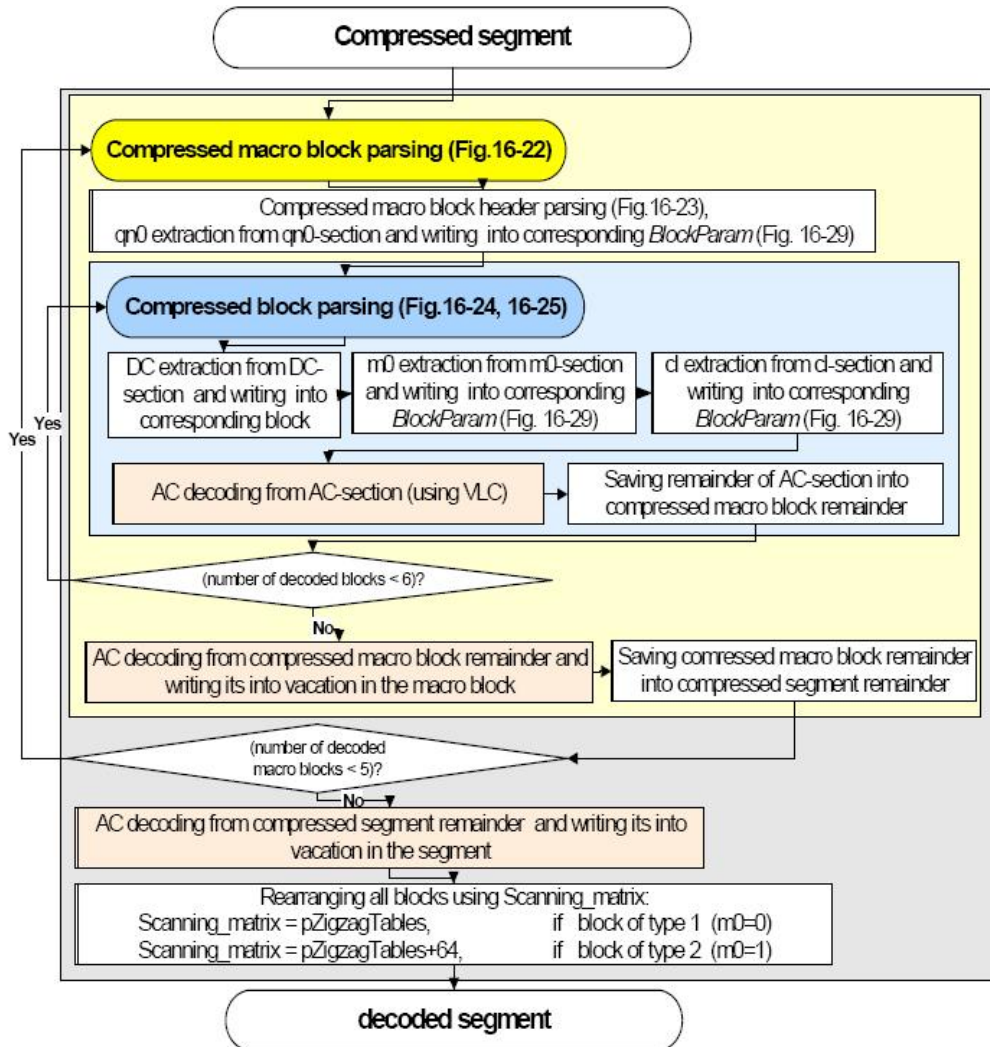
```
static Ipp32s Table[]=
{
    max_bits;           The maximum length of code
    sub_sz1;            Not used

    sub_sz2;
    N1;                 The number of 1-bit codes
    (code1, run1, level1); The 1-bit codes, run values and level values.
    (code2, run2, level2);
    ...;
    (codeN1, runN1, levelN1);
    N2;                 The number of 2-bit codes
    (code1, run1, level1); The 2-bit codes, run values and level values.
    (code2, run2, level2);
    ...;
    (codeN2, runN2, levelN2);
    ...;
    Nm;                 ...
    (code1, run1, level1); The number of maximum length codes
    (code2, run2, level2); The maximum length codes, run values and level values.
    ...;
}
```

```
(codeNm, runNm, levelNm);
-1;
};
```

The significant value to indicate the end of table

### Compressed Segment Decoding



### InitAllocHuffmanTable\_DV

*Allocates memory and initializes table.*

#### Syntax

```
IppStatus ippiInitAllocHuffmanTable_DV_32u(Ipp32s* pSrcTable1, Ipp32s* pSrcTable2, Ipp32u** ppHuffTable);
```

## Parameters

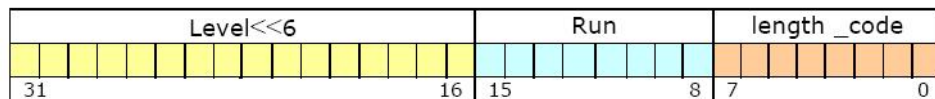
<i>pSrcTable1</i>	Pointer to Source Table 1.
<i>pSrcTable2</i>	Pointer to Source Table 2.
<i>ppHuffTable</i>	Double pointer to the destination decoding table.

## Description

This function is declared in the `ippvc.h` header file. The function `ippiInitAllocHuffmanTable_DV_32u` allocates memory and initializes the table that contains codes for DCT coefficients (Run-Level codes). The function allocates memory at the address that `**ppHuffTable` points to, and fills it with data from Source Table 1 that `pSrcTable1` points to and from Source Table 2 that `pSrcTable2` points to. See ["Source Table Structure"](#) for the source table structure. When filling the destination decoding table the function multiplies the value of `level` by 64 (shift left by 6).

While decoding, the search for code is performed through the first subtable ([Figure "VLC Subtables"](#)), which contains the most frequent and shortest codes. If the code value is not found, then the search continues through the following subtables, containing additional bits for longer codes. Those subtables also forward the search to the next ones, if the code value is not found.

## HuffT Structure



`length_code = length code (if source tables == pSrcTable1),`

`length_code = length code + 4 (if source tables == pSrcTable2)`

This function is used in the DV decoder included into Intel IPP Samples. See [introduction](#) to this section.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when at least one input pointer is NULL.
<code>ippStsMemAllocErr</code>	Indicates an error when no memory is allocated.

## Example

The code example below shows how to use `ippiInitAllocHuffmanTable_DV_32u` function.

```

Ipp32s HuffSrcTable1[] =
{
    9, /* Maximum length of code in this table*/
    0, /* Not used*/
    0, /* Not used*/
    0, /* Number of 1-bit codes*/
    0, /* Number of 2-bit codes*/
    2, /* Number of 3-bit codes*/

```

```

0x0000, 0, 1,      0x0001, 0, -1,
3, /* Number of 4-bit codes*/
0x0004, 0, 2,      0x0005, 0, -2,

0x0006, 100, 100, /* EOB*/

6, /* Number of 5-bit codes*/
0x000e, 1, 1,      0x000f, 1, -1,
0x0010, 0, 3,      0x0011, 0, -3,
0x0012, 0, 4,      0x0013, 0, -4,
8, /* Number of 6-bit codes*/
0x0028, 2, 1,      0x0029, 2, -1,
0x002a, 1, 2,      0x002b, 1, -2,
0x002c, 0, 5,      0x002d, 0, -5,
0x002e, 0, 6,      0x002f, 0, -6,
8, /* Number of 7-bit codes*/
0x0060, 3, 1,      0x0061, 3, -1,
0x0062, 4, 1,      0x0063, 4, -1,
0x0064, 0, 7,      0x0065, 0, -7,
0x0066, 0, 8,      0x0067, 0, -8,
16, /* Number of 8-bit codes*/
0x00d0, 5, 1,      0x00d1, 5, -1,
0x00d2, 6, 1,      0x00d3, 6, -1,
0x00d4, 2, 2,      0x00d5, 2, -2,
0x00d6, 1, 3,      0x00d7, 1, -3,
0x00d8, 1, 4,      0x00d9, 1, -4,
0x00da, 0, 9,      0x00db, 0, -9,
0x00dc, 0, 10,     0x00dd, 0, -10,
0x00de, 0, 11,     0x00df, 0, -11,
32, /* Number of 9-bit codes*/
0x01c0, 7, 1,      0x01c1, 7, -1,
0x01c2, 8, 1,      0x01c3, 8, -1,
0x01c4, 9, 1,      0x01c5, 9, -1,
0x01c6, 10, 1,     0x01c7, 10, -1,
0x01c8, 3, 2,      0x01c9, 3, -2,
0x01ca, 4, 2,      0x01cb, 4, -2,
0x01cc, 2, 3,      0x01cd, 2, -3,
0x01ce, 1, 5,      0x01cf, 1, -5,
0x01d0, 1, 6,      0x01d1, 1, -6,
0x01d2, 1, 7,      0x01d3, 1, -7,
0x01d4, 0, 12,     0x01d5, 0, -12,
0x01d6, 0, 13,     0x01d7, 0, -13,
0x01d8, 0, 14,     0x01d9, 0, -14,
0x01da, 0, 15,     0x01db, 0, -15,
0x01dc, 0, 16,     0x01dd, 0, -16,
0x01de, 0, 17,     0x01df, 0, -17,
-1 /* end of table*/
};

Ipp32s HuffSrcTable2[] =
{
    9, /* Maximum length of code. Note that 4 is added to
        all code lengths in the second table*/
    0, /* Not used*/
    0, /* Not used*/

    0, /* Number of 1(5)-bit codes*/
    0, /* Number of 2(6)-bit codes*/
    0, /* Number of 3(7)-bit codes*/
    0, /* Number of 4(8)-bit codes*/
    0, /* Number of 5(9)-bit codes*/
    32, /* Number of 6(10)-bit codes*/

```

```

0x0000, 11, 1,      0x0001, 11, -1,
0x0002, 12, 1,      0x0003, 12, -1,
0x0004, 13, 1,      0x0005, 13, -1,
0x0006, 14, 1,      0x0007, 14, -1,
0x0008, 5, 2,       0x0009, 5, -2,
0x000a, 6, 2,       0x000b, 6, -2,
0x000c, 3, 3,       0x000d, 3, -3,
0x000e, 4, 3,       0x000f, 4, -3,

0x0010, 2, 4,       0x0011, 2, -4,
0x0012, 2, 5,       0x0013, 2, -5,
0x0014, 1, 8,       0x0015, 1, -8,
0x0016, 0, 18,      0x0017, 0, -18,
0x0018, 0, 19,      0x0019, 0, -19,
0x001a, 0, 20,      0x001b, 0, -20,
0x001c, 0, 21,      0x001d, 0, -21,
0x001e, 0, 22,      0x001f, 0, -22,
16, /* Number of 7(11)-bit codes*/
0x0040, 5, 3,       0x0041, 5, -3,
0x0042, 3, 4,       0x0043, 3, -4,
0x0044, 3, 5,       0x0045, 3, -5,
0x0046, 2, 6,       0x0047, 2, -6,
0x0048, 1, 9,       0x0049, 1, -9,
0x004a, 1, 10,      0x004b, 1, -10,
0x004c, 1, 11,      0x004d, 1, -11,
0x004e, 0, 0,       0x004f, 1, 0,
16, /* Number of 8(12)-bit codes*/
0x00a0, 6, 3,       0x00a1, 6, -3,
0x00a2, 4, 4,       0x00a3, 4, -4,
0x00a4, 3, 6,       0x00a5, 3, -6,
0x00a6, 1, 12,      0x00a7, 1, -12,
0x00a8, 1, 13,      0x00a9, 1, -13,
0x00aa, 1, 14,      0x00ab, 1, -14,
0x00ac, 2, 0,       0x00ad, 3, 0,
0x00ae, 4, 0,       0x00af, 5, 0,

16*2 + 56 + 6 + 2, /* Number of 9(13)-bit codes*/
0x0160, 7, 2,       0x0161, 7, -2,
0x0162, 8, 2,       0x0163, 8, -2,
0x0164, 9, 2,       0x0165, 9, -2,
0x0166, 10, 2,      0x0167, 10, -2,
0x0168, 7, 3,       0x0169, 7, -3,
0x016a, 8, 3,       0x016b, 8, -3,
0x016c, 4, 5,       0x016d, 4, -5,
0x016e, 3, 7,       0x016f, 3, -7,

0x0170, 2, 7,       0x0171, 2, -7,
0x0172, 2, 8,       0x0173, 2, -8,
0x0174, 2, 9,       0x0175, 2, -9,
0x0176, 2, 10,      0x0177, 2, -10,
0x0178, 2, 11,      0x0179, 2, -11,
0x017a, 1, 15,      0x017b, 1, -15,
0x017c, 1, 16,      0x017d, 1, -16,
0x017e, 1, 17,      0x017f, 1, -17,

/* Combinations (0,0) through (6,0) have special codes*/
0x0180 + 0, 0, 0,    0x0180 + 1, 0, 0,
0x0180 + 2, 0, 0,    0x0180 + 3, 0, 0,
0x0180 + 4, 0, 0,    0x0180 + 5, 0, 0, /* 1*/

0x0180 + 6, 6, 0,    0x0180 + 7, 7, 0, 0x0180 + 8, 8, 0, 0x0180 + 9, 9, 0, /* 1*/
0x0180 + 10, 10, 0, 0x0180 + 11, 11, 0, 0x0180 + 12, 12, 0, 0x0180 + 13, 13, 0, /* 2*/

```

```

0x0180 + 14, 14, 0, 0x0180 + 15, 15, 0, 0x0180 + 16, 16, 0, 0x0180 + 17, 17, 0, /* 3*/
0x0180 + 18, 18, 0, 0x0180 + 19, 19, 0, 0x0180 + 20, 20, 0, 0x0180 + 21, 21, 0, /* 4*/
0x0180 + 22, 22, 0, 0x0180 + 23, 23, 0, 0x0180 + 24, 24, 0, 0x0180 + 25, 25, 0, /* 5*/
0x0180 + 26, 26, 0, 0x0180 + 27, 27, 0, 0x0180 + 28, 28, 0, 0x0180 + 29, 29, 0, /* 6*/
0x0180 + 30, 30, 0, 0x0180 + 31, 31, 0, 0x0180 + 32, 32, 0, 0x0180 + 33, 33, 0, /* 7*/
0x0180 + 34, 34, 0, 0x0180 + 35, 35, 0, 0x0180 + 36, 36, 0, 0x0180 + 37, 37, 0, /* 8*/
0x0180 + 38, 38, 0, 0x0180 + 39, 39, 0, 0x0180 + 40, 40, 0, 0x0180 + 41, 41, 0, /* 9*/
0x0180 + 42, 42, 0, 0x0180 + 43, 43, 0, 0x0180 + 44, 44, 0, 0x0180 + 45, 45, 0, /* 10*/
0x0180 + 46, 46, 0, 0x0180 + 47, 47, 0, 0x0180 + 48, 48, 0, 0x0180 + 49, 49, 0, /* 11*/
0x0180 + 50, 50, 0, 0x0180 + 51, 51, 0, 0x0180 + 52, 52, 0, 0x0180 + 53, 53, 0, /* 12*/
0x0180 + 54, 54, 0, 0x0180 + 55, 55, 0, 0x0180 + 56, 56, 0, 0x0180 + 57, 57, 0, /* 13*/
0x0180 + 58, 58, 0, 0x0180 + 59, 59, 0, 0x0180 + 60, 60, 0, 0x0180 + 61, 61, 0, /* 14 (14*4=56)*/
0x0180 + 62, 0, 0, 0x0180 + 63, 0, 0, /* combinations (62,0) and (63,0) are not anticipated*/
-1 /* end of table*/
};

{
    if (ippStsOk != ippiInitAllocHuffmanTable_DV_32u(SrcTable1,
                                                    SrcTable2,
                                                    &pHuffTable))
    {
        /* Process errors*/
    }
}

```

## HuffmanDecodeSegment\_DV

*Decodes and rearranges segment block, multiplies first block element by 128.*

### Syntax

```

IppStatus ippiHuffmanDecodeSegment_DV_8u16s(const Ipp8u* pStream, const Ipp32u*
pZigzagTables, const Ipp32u* pHuffTable, Ipp16s* pBlock, Ipp32u* pBlockParam);

```

### Parameters

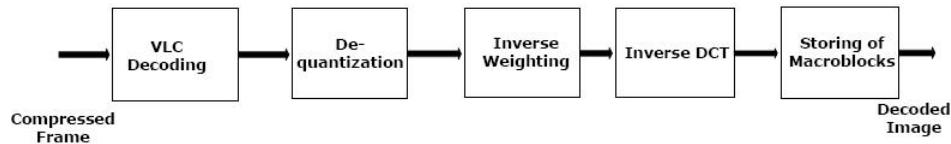
<i>pStream</i>	Pointer to bitstream (compressed segment).
<i>pZigzagTables</i>	Pointer to the array of two scanning matrices.
<i>pHuffTable</i>	Pointer to the decoding Huffman table.
<i>pBlock</i>	Pointer to decoded elements.
<i>pBlockParam</i>	Pointer to output parameters array [30].

### Description

This function is declared in the `ippvc.h` header file. The function `ippiHuffmanDecodeSegment_DV_8u16s` decodes and rearranges segment, multiplies the first block element by 128.

The figure below depicts the DV frame decoding process.

### DV Frame Decoding



At the first step, `ippiHuffmanDecodeSegment_DV_8u16s` function decodes the compressed frame. The output of the function is 6 decompressed macro blocks, which are then de-quantized, de-weighted, processed by inverse DCT, and stored to the appropriate places of the destination image.

AC coefficients are decoded using Variable Length Codes with the help of the destination decoding table created by `InitAllocHuffmanTable_DV`. The pointer to this table is passed to the `ippiHuffmanDecodeSegment_DV` function through the `pHuffTable` pointer. The table contains level values multiplied by 64, and the `ippiHuffmanDecodeSegment_DV` function also multiplies output values of AC coefficients by 64.

DC coefficients are obtained from the first 2 bytes of the compressed blocks (see "[Arrangement of Y compressed blocks cbY0, cbY1, cbY2, cbY3](#)" and "[Arrangement of Cr compressed blocks cbU, cbV](#)") with zeroing of the last 7 bits. The output values of DC coefficients are multiplied by 128 (shift left by 7).

The pointer `pStream` points to a compressed segment, which is a sequence of 5 compressed macroblocks. Each compressed macroblock occupies 80 bytes of data, hence the total length of the compressed video segment is 400 bytes.

The `pZigzagTables` pointer refers to two scanning matrices which are used to de-zigzag AC coefficients.

`Scanning_matrix = pZigzagTables`, if the block is of type 1 (`m0 = 0`). `Scanning_matrix = pZigzagTables + 64`, if the block is of type 2 (`m0 = 1`).

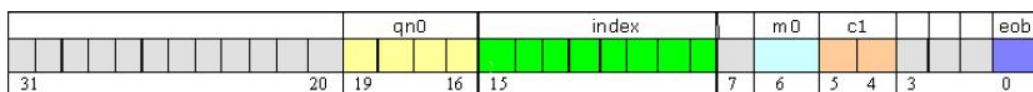
The argument `*pBlockParam` ("[BlockParam Structure](#)") stores values `m0` (block type), `c1` (class number), `index` (index of the last decoded element), and `eob` (end of block indicator) for every block.

Value `qn0` (quantization number) is obtained from the first block of the macroblock. The same value `qn0` is used for all blocks of the macroblock.

With 64 elements in an 8x8 block, if only the DC coefficient (that is, the first block element) is decoded, `index=0`. If all the elements are decoded, `index=63`.

`eob=1` ("true"), if the symbol of the block end EOB is present in the block, otherwise `eob=0` ("false").

### BlockParam Structure





This function is used in the DV decoder included into Intel IPP Samples. See [introduction](#) to this section.

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when at least one input pointer is NULL.

### Example

The code example below shows how to use `HuffmanDecodeSegment_DV` function.

```

Ipp32u DeZigzagTables[] =
{
    0,
    1,

    8,          16, 9,  2,          3, 10, 17,
    24,          32,          25,          18,          11,
    4,           5,           12,          19,           26,
    33,          40,          48,          41,           34,
    27,          20,          13,          6,           7,
    14,          21,          28,          35,           42,
    49,          56,          57,          50,           43,
    36,          29, 22, 15,          23, 30, 37,
    44,          51,
    59,          52,
    38,          31,
    46,          53,
    61,          54,
    55,          62,
    63,

    0, 32, 1, 33, 8, 40, 2, 34,
    9, 41, 16, 48, 24, 56, 17, 49,
    10, 42, 3, 35, 4, 36, 11, 43,
    18, 50, 25, 57, 26, 58, 19, 51,
    12, 44, 5, 37, 6, 38, 13, 45,
    20, 52, 27, 59, 28, 60, 21, 53,
    14, 46, 7, 39, 15, 47, 22, 54,
    29, 61, 30, 62, 23, 55, 31, 63
};

{
    /* Create Huffman Decoding table*/
    ippiInitAllocHuffmanTable_DV_32u( ..., ..., &pHuffTable);

    ...

    /* Huffman Decode video segment

    pEncodedVideoSegment - (Ipp8u*) pointer to compresses video segment
    DeZigzagTables        - (Ipp32u*) pointer to array of two de-zigzag tables
    pHuffTable            - (Ipp32u*) pointer to Huffman decoding table created
                           by the ippiInitAllocHuffmanTable_DV_32u function
    lpsDecodedBlocks      - (Ipp16s*) pointer to 30 decoded DCT blocks (5 macro
                           blocks of 6 DCT blocks)
    BlockParamBuffer      - (Ipp32u*) pointer to array of 30 unsigned double
                           words, containing DC, m0 and cl values for
                           DCT blocks

    */
    ippiHuffmanDecodeSegment_DV_8u16s(pEncodedVideoSegment,

```

```

        DeZigzagTables,
        pHuffTable,
        lpsDecodedBlocks,
        BlockParamBuffer);
}

```

The code example below shows how to use `HuffmanDecodeSegment_DV` with *index* and *eob* values

```

void test_huff2(void)
{
    /* Create Huffman Decoding table*/
    ippiInitAllocHuffmanTable_DV_32u( ..., ..., &pHuffTable);
    ...
    /* Huffman Decode video segment
    pEncodedVideoSegment - (Ipp8u*) pointer to compresses video segment
    DeZigzagTables - (Ipp32u*) pointer to array of two de-zigzag tables
    pHuffTable - (Ipp32u*) pointer to Huffman decoding table created
    by the ippiInitAllocHuffmanTable_DV_32u function
    lpsDecodedBlocks - (Ipp16s*) pointer to 30 decoded DCT blocks (5 macro
    blocks of 6 DCT blocks)
    BlockParamBuffer - (Ipp32u*) pointer to array of 30 unsigned double
    words, containing DC, m0 and cl values for
    DCT blocks
    */
    ippiHuffmanDecodeSegment_DV_8u16s(pEncodedVideoSegment,
        DeZigzagTables,
        pHuffTable,
        lpsDecodedBlocks,
        BlockParamBuffer);
    for (mb_num = 0; mb_num < 5; mb_num++)
    {
        // get quantization number
        qno = (BlParamBuffer[mb_num * 6] >> 16) & 0x0F;

        // decompress each block
        for (b_num = 0; b_num < 6; b_num++)
        {
            block_type = (BlParamBuffer[mb_num * 6 + b_num] >> 6) & 0x01;
            bl_index = (BlParamBuffer[mb_num * 6 + b_num] >> 8) & 0xff;
            eob = BlParamBuffer[mb_num * 6 + b_num] & 0x01;

            // do inverse and adaptive dequantization
            ippiQuantInv_DV_16s_C1I(lpsDecodedBlocks,...);

            // do iDCT
            if (0 == block_type)
            {
                if ((eob == 0) || (10 > bl_index))
                    ippiDCT8x8Inv_4x4_16s_C1I((Ipp16s *) (lpsDecodedBlocks));
                else
                    ippiDCT8x8Inv_16s_C1I((Ipp16s *) (lpsDecodedBlocks));
            }
            else
            {
                ippiDCT2x4x8Inv_16s_C1I((Ipp16s *) (lpsDecodedBlocks));
            }
        }
    }
}

```

## HuffmanDecodeSegment\_DV100

*Decodes and rearranges DV100 segment block, multiplies first block element by 128.*

---

### Syntax

```
IppStatus ippiHuffmanDecodeSegment_DV100_8u16s(const Ipp8u* pStream, const Ipp32u* pZigzagTables, const Ipp32u* pHuffTable, Ipp16s* pBlock, Ipp32u* pBlockParam);
```

### Parameters

<i>pStream</i>	Pointer to bitstream (compressed video segment).
<i>pZigzagTables</i>	Pointer to the de-zigzag table.
<i>pHuffTable</i>	Pointer to the decoding Huffman table.
<i>pBlock</i>	Pointer to the array where DCT decoded blocks should be stored.
<i>pBlockParam</i>	Pointer to output parameters array [40].

### Description

This function is declared in the `ippvc.h` header file. The function `ippiHuffmanDecodeSegment_DV100_8u16s` decodes and rearranges DV100 video segment, multiplies the first block element by 128.

AC coefficients are decoded using Variable Length Codes with the help of the destination decoding table created by `InitAllocHuffmanTable_DV`. Since this table contains level values multiplied by 64, the output values of AC coefficients are also multiplied by 64.

DC coefficients are obtained from the first 2 bytes of the compressed blocks (see "[Arrangement of Y compressed blocks cbY0, cbY1, cbY2, cbY3](#)" and "[Arrangement of Cr compressed blocks cbU, cbV](#)") with zeroing of the last 7 bits. The output values of DC coefficients are multiplied by 128 (shift left by 7).

The pointer *pStream* points to a compressed video segment, which is a sequence of 5 compressed macroblocks. Each macroblock consists of 8 DCT blocks (4 luma and 4 color difference blocks).

The *pZigzagTables* pointer refers to the array that contains the de-zigzag index.

The argument *\*pBlockParam* ("[BlockParam Structure](#)") stores values `m0` (block type), `c1` (class number) for every block. In DV100, the same DCT mode (8-8-frame or 8-8-field) is applied to all DCT blocks in a macroblock. The DCT mode is stored in the first DCT block of the macro block; `m0` values of the other DCT blocks are not used. Value `qn0` (quantization number) is obtained from the first block of the macroblock. The same value `qn0` is used for all blocks of the macroblock.

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when at least one input pointer is <code>NULL</code> .

### Example

The code example below shows how to use `ippiHuffmanDecodeSegment_DV100_8u16s` function.

```
{
    /* Create Huffman Decoding table*/
    /*Dv and DV100 use same Huffman Decoding table*/
```

```

ippiInitAllocHuffmanTable_DV_32u( ..., ..., &pHuffTable);
/*
...
*/
/* Huffman Decode video segment
pEncodedVideoSegment - (Ipp8u*) pointer to compresses video segment
DeZigzagTables - (Ipp32u*) pointer to array of de-zigzag table
pHuffTable - (Ipp32u*) pointer to Huffman decoding table created
by the ippiInitAllocHuffmanTable_DV_32u function
lpsDecodedBlocks - (Ipp16s*) pointer to 40 decoded DCT blocks (5 macro
blocks of 8 DCT blocks, 8* 5 = 40)
BlockParamBuffer - (Ipp32u*) pointer to array of 40 unsigned double
words, containing DC, m0 and cl values for DCT blocks
*/
ippiHuffmanDecodeSegment_DV100_8u16s(pEncodedVideoSegment,
                                     DeZigzagTable,
                                     pHuffTable,
                                     lpsDecodedBlocks,
                                     BlockParamBuffer);

/*
...
*/
};

```

## HuffmanDecodeSegmentOnePass\_DV

*Performs first pass of video segment decoding,  
rearranges segment block, multiplies first block element  
by 128.*

---

### Syntax

```

IppStatus ippiHuffmanDecodeSegmentOnePass_DV_8u16s(const Ipp8u* pStream, const Ipp32u*
pZigzagTables, const Ipp32u* pHuffTable, Ipp16s* pBlock, Ipp32u* pBlockParam, Ipp32s
nNumCoeffs);

```

### Parameters

<i>pStream</i>	Pointer to bitstream (compressed video segment).
<i>pZigzagTables</i>	Pointer to the array of two scanning matrices.
<i>pHuffTable</i>	Pointer to the decoding Huffman table.
<i>pBlock</i>	Pointer to the array where DCT decoded blocks should be stored.
<i>pBlockParam</i>	Pointer to output parameters array [30].
<i>nNumCoeffs</i>	Maximum number of DCT coefficients to extract from the stream.

### Description

This function is declared in the `ippvc.h` header file. The function `ippiHuffmanDecodeSegmentOnePass_DV_8u16s` is designed to be used in DV video decoder in a quarter resolution mode. The function performs only the first pass of video segment VLC decoding and extracts not more than specified number of DCT coefficients.

The function accepts the same arguments as the [HuffmanDecodeSegment\\_DV](#) function and one additional parameter - the maximum number of DCT coefficients to extract from the stream, which is passed through the *nNumCoeffs* argument. If the limit of coefficients is reached during the first pass, the function goes over to the next DCT block. If the limit is not reached during the first pass, the function does not perform the second and the third passes and extracts only the number of DCT coefficients available in the first pass.

The *nNumCoeffs* parameter must fall within the range of 1 to 64.

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when at least one input pointer is NULL.
<code>ippStsBadArgErr</code>	Indicates an error when <i>nNumCoeffs</i> does not fall within the range of 1 through 64.

## FreeHuffmanTable\_DV

*Frees the memory allocated for VLC table.*

---

### Syntax

```
IppStatus ippFreeHuffmanTable_DV_32u(Ipp32u* pHuffTable);
```

### Parameters

*pHuffTable*                      Pointer to the decoding table.

### Description

This function is declared in the `ippvc.h` header file. The function `ippFreeHuffmanTable_DV_32u` frees the memory at *pHuffTable* allocated for VLC table.

This function is used in the DV decoder included into Intel IPP Samples. See [introduction](#) to this section.

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when <i>pHuffTable</i> is NULL.

## Inverse Quantization

## QuantInv\_DV

*Performs inverse quantization on a block.*

---

### Syntax

```
IppStatus ippQuantInv_DV_16s_C1I(Ipp16s* pSrcDst, Ipp16s* pDequantTable);
```

### Parameters

*pSrcDst*                          Pointer to the block.

*pDequantTable*

Pointer to the dequantization table.

### Description

This function is declared in the `ippvc.h` header file. The function `ippiQuantInv_DV_16s_C1I` performs inverse quantization on a block. Each of the decoded DCT coefficients in the block should be inverse quantized through multiplying by a corresponding value from the weighting matrix and division on  $2^{14}$ . Selection of dequantization table depends of the block type.

See [Example "Inverse Quantization Tables"](#) and [Example "ippiQuantInv\\_DV\\_16s\\_C1I Usage"](#) for an example of the function usage.

This function is used in the DV decoder included into Intel IPP Samples. See [introduction](#) to this section.

### Return Values

`ippStsNoErr`

Indicates no error.

`ippStsNullPtrErr`Indicates an error when at least one input pointer is `NULL`.

## QuantWeightBlockInv\_DV

*Performs inverse quantization and inverse weighting on a block.*

---

### Syntax

```
ippStatus ippiQuantWeightBlockInv_DV_16s_C1I(Ipp16s* pSrcDst, constIpp16s* pQuantInvTable,
const Ipp16s* pWeightInvTable);
```

### Parameters

*pSrcDst*

Pointer to the source and destination 8x8 (or 2x4x8) block.

*pQuantInvTable*

Pointer to an array that contains values of quantification table for the current block.

*pWeightInvTable*

Pointer to an array that contains values of weight table for the current block.



**NOTE.** There are two different types of blocks: 8x8 and 2x4x8. Select respective weighting tables for 8x8 or 2x4x8 blocks.

---

### Description

This function is declared in the `ippvc.h` header file. The function `ippiQuantWeightBlockInv_DV_16s_C1I` performs inverse quantization and inverse weighting on a block in accordance with [\[SMPT E314M\]](#) standard.

The DV frame decoding process is depicted in ["BlockParam Structure"](#). The `ippiQuantWeightBlockInv_DV_16s_C1I` function covers de-quantization and inverse weighting steps. Choose appropriate quantization and weighting tables, that is, dequantization and deweight tables for decoding. To choose the quantization table, use the values of two parameters - *qno* (quantization number) and the *block quant class*. For details, see [\[SMPT E314M\]](#), Table 23 - Quantization step. Use different weighting tables for 8x8 and 2x4x8 blocks.

The `ippiQuantWeightBlockInv_DV_16s_C1I` function uses the following three pointers as input arguments:

<code>pDataBlock</code>	Pointer to the current block
<code>pQuantTable</code>	Pointer to the quantization table
<code>pWeightTable</code>	Pointer to the weighting table

The function performs the following operations:

```
for (Ipp32s i = 0; i < 64; i += 1)
    pDataBlock[i] = (Ipp16s) ( (
        ((Ipp16s) pDataBlock [i]>>3      )
        * ((Ipp16s) pQuantTable[i] )
        * ((Ipp16s) pWeightTable[i] ) >>16);
```

The `pWeightTable` weighting table contains floating point data with values varying from 2.0 to 4.0. Scaling is required for integer multiplication through multiplying the table values by 8192 ( $1 << 13$ ). Then back shift is required through division by 65536 to provide for REAL range, that is, to perform inverse scaling.

After `ippiQuantWeightBlockInv_DV_16s_C1I` completes dequantization and inverse weighing, iDCT is performed before creating the macroblock.

Compare [QuantInv\\_DV](#) and `QuantWeightBlockInv_DV`. The `ippiQuantInv_DV_16s_C1I` function has two arguments: `pDataBlock`, which is a pointer to the current block, and `pQuantTable`, a modification of the quantization table with values multiplied by 256 ( $1 << 8$ ) and divided by weight coefficients for `Ipp16s` datatype. The `ippiQuantWeightBlockInv_DV_16s_C1I` function has separate quantization and weighting tables that contain input arguments for the function.

This function is used in the DV decoder included into Intel IPP Samples. See [introduction](#) to this section.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when the <code>pSrcDst</code> pointer is NULL.

## Example

The code example below shows how to use inverse quantization tables.

```
{
//-----
//Several necessary tables

#define _INTERNAL_CS1 0.980785 //cos(1*PI/16)
#define _INTERNAL_CS2 0.923880 //cos(2*PI/16)
#define _INTERNAL_CS3 0.831470 //cos(3*PI/16)
#define _INTERNAL_CS4 0.707107 //cos(4*PI/16)
#define _INTERNAL_CS5 0.555570 //cos(5*PI/16)
#define _INTERNAL_CS6 0.382683 //cos(6*PI/16)
#define _INTERNAL_CS7 0.195090 //cos(7*PI/16)

#define _W0 (1.0000001)
#define _W1 (_INTERNAL_CS4 / (4.0* _INTERNAL_CS7* _INTERNAL_CS2))
#define _W2 (_INTERNAL_CS4 / (2.0* _INTERNAL_CS6))
#define _W3 (1.0 / (2.0* _INTERNAL_CS5))
#define _W4 (7.0 / 8.0)
#define _W5 (_INTERNAL_CS4 / _INTERNAL_CS3)
#define _W6 (_INTERNAL_CS4 / _INTERNAL_CS2)
#define _W7 (_INTERNAL_CS4 / _INTERNAL_CS1)
```

```

tabl1[] =
{
    0,1,8,16,    9,2,3,10,
    17,24,32,25, 18,11,4,5,
    12,19,26,33, 40,48,41,34,
    27,20,13,6,   7,14,21,28,
    35,42,49,56, 57,50,43,36,
    29,22,15,23, 30,37,44,51,
    58,59,52,45, 38,31,39,46,
    53,60,61,54, 47,55,62,63
};

Ipp16s tabl2[] =
{
    0, 32, 1, 33, 8, 40, 2, 34,
    9, 41, 16, 48, 24, 56, 17, 49,
    10, 42, 3, 35, 4, 36, 11, 43,
    18, 50, 25, 57, 26, 58, 19, 51,
    12, 44, 5, 37, 6, 38, 13, 45,
    20, 52, 27, 59, 28, 60, 21, 53,
    14, 46, 7, 39, 15, 47, 22, 54,
    29, 61, 30, 62, 23, 55, 31, 63
};

static double TablW0[] =
{
    (0.5/*2500076*/), (_W0*_W1/2), (_W0*_W2/2), (_W0*_W3/2), (_W0*_W4/2), (_W0*_W5/2), (_W0*_W6/2), (_W0*_W7/2),
    (_W1*_W0/2), (_W1*_W1/2), (_W1*_W2/2), (_W1*_W3/2), (_W1*_W4/2), (_W1*_W5/2), (_W1*_W6/2), (_W1*_W7/2),
    (_W2*_W0/2), (_W2*_W1/2), (_W2*_W2/2), (_W2*_W3/2), (_W2*_W4/2), (_W2*_W5/2), (_W2*_W6/2), (_W2*_W7/2),
    (_W3*_W0/2), (_W3*_W1/2), (_W3*_W2/2), (_W3*_W3/2), (_W3*_W4/2), (_W3*_W5/2), (_W3*_W6/2), (_W3*_W7/2),
    (_W4*_W0/2), (_W4*_W1/2), (_W4*_W2/2), (_W4*_W3/2), (_W4*_W4/2), (_W4*_W5/2), (_W4*_W6/2), (_W4*_W7/2),
    (_W5*_W0/2), (_W5*_W1/2), (_W5*_W2/2), (_W5*_W3/2), (_W5*_W4/2), (_W5*_W5/2), (_W5*_W6/2), (_W5*_W7/2),
    (_W6*_W0/2), (_W6*_W1/2), (_W6*_W2/2), (_W6*_W3/2), (_W6*_W4/2), (_W6*_W5/2), (_W6*_W6/2), (_W6*_W7/2),
    (_W7*_W0/2), (_W7*_W1/2), (_W7*_W2/2), (_W7*_W3/2), (_W7*_W4/2), (_W7*_W5/2), (_W7*_W6/2), (_W7*_W7/2)
};

static Ipp16s TablW0_Scale_ZigZag[] =
{
    (Ipp16s)(8192.0/TablW0[0]), (Ipp16s)(8192.0/TablW0[1]), (Ipp16s)(8192.0/TablW0[8]),
    (Ipp16s)(8192.0/TablW0[16]),
    (Ipp16s)(8192.0/TablW0[9]), (Ipp16s)(8192.0/TablW0[2]), (Ipp16s)(8192.0/TablW0[3]),
    (Ipp16s)(8192.0/TablW0[10]),
    (Ipp16s)(8192.0/TablW0[17]), (Ipp16s)(8192.0/TablW0[24]), (Ipp16s)(8192.0/TablW0[32]), (Ipp16s)(8192.0/TablW0[25]),
    (Ipp16s)(8192.0/TablW0[18]), (Ipp16s)(8192.0/TablW0[11]), (Ipp16s)(8192.0/TablW0[4]), (Ipp16s)(8192.0/TablW0[5]),
    (Ipp16s)(8192.0/TablW0[12]), (Ipp16s)(8192.0/TablW0[19]), (Ipp16s)(8192.0/TablW0[26]), (Ipp16s)(8192.0/TablW0[33]),
    (Ipp16s)(8192.0/TablW0[40]), (Ipp16s)(8192.0/TablW0[48]), (Ipp16s)(8192.0/TablW0[41]), (Ipp16s)(8192.0/TablW0[34]),
    (Ipp16s)(8192.0/TablW0[27]), (Ipp16s)(8192.0/TablW0[20]), (Ipp16s)(8192.0/TablW0[13]), (Ipp16s)(8192.0/TablW0[6]),
    (Ipp16s)(8192.0/TablW0[7]),
    (Ipp16s)(8192.0/TablW0[14]), (Ipp16s)(8192.0/TablW0[21]), (Ipp16s)(8192.0/TablW0[28]),
    (Ipp16s)(8192.0/TablW0[35]), (Ipp16s)(8192.0/TablW0[42]), (Ipp16s)(8192.0/TablW0[49]), (Ipp16s)(8192.0/TablW0[56]),
    (Ipp16s)(8192.0/TablW0[57]), (Ipp16s)(8192.0/TablW0[50]), (Ipp16s)(8192.0/TablW0[43]), (Ipp16s)(8192.0/TablW0[36]),
    (Ipp16s)(8192.0/TablW0[29]), (Ipp16s)(8192.0/TablW0[22]), (Ipp16s)(8192.0/TablW0[15]), (Ipp16s)(8192.0/TablW0[23]),
    (Ipp16s)(8192.0/TablW0[30]), (Ipp16s)(8192.0/TablW0[37]), (Ipp16s)(8192.0/TablW0[44]), (Ipp16s)(8192.0/TablW0[51]),
    (Ipp16s)(8192.0/TablW0[58]), (Ipp16s)(8192.0/TablW0[59]), (Ipp16s)(8192.0/TablW0[52]), (Ipp16s)(8192.0/TablW0[45]),
    (Ipp16s)(8192.0/TablW0[38]), (Ipp16s)(8192.0/TablW0[31]), (Ipp16s)(8192.0/TablW0[39]), (Ipp16s)(8192.0/TablW0[46]),
    (Ipp16s)(8192.0/TablW0[53]), (Ipp16s)(8192.0/TablW0[60]), (Ipp16s)(8192.0/TablW0[61]), (Ipp16s)(8192.0/TablW0[54]),
    (Ipp16s)(8192.0/TablW0[47]), (Ipp16s)(8192.0/TablW0[55]),
    (Ipp16s)(8192.0/TablW0[62]), (Ipp16s)(8192.0/TablW0[63])
};

```



```

static double TablW1[] =
{
(0.5/*2500076*/),(_W0*_W1/2),(_W0*_W2/2),(_W0*_W3/2),(_W0*_W4/2),(_W0*_W5/2),(_W0*_W6/2),(_W0*_W7/2),
(_W2*_W0/2),(_W2*_W1/2),(_W2*_W2/2),(_W2*_W3/2),(_W2*_W4/2),(_W2*_W5/2),(_W2*_W6/2),(_W2*_W7/2),
(_W4*_W0/2),(_W4*_W1/2),(_W4*_W2/2),(_W4*_W3/2),(_W4*_W4/2),(_W4*_W5/2),(_W4*_W6/2),(_W4*_W7/2),
(_W6*_W0/2),(_W6*_W1/2),(_W6*_W2/2),(_W6*_W3/2),(_W6*_W4/2),(_W6*_W5/2),(_W6*_W6/2),(_W6*_W7/2),
(_W0*_W0/2),(_W0*_W1/2),(_W0*_W2/2),(_W0*_W3/2),(_W0*_W4/2),(_W0*_W5/2),(_W0*_W6/2),(_W0*_W7/2),
(_W2*_W0/2),(_W2*_W1/2),(_W2*_W2/2),(_W2*_W3/2),(_W2*_W4/2),(_W2*_W5/2),(_W2*_W6/2),(_W2*_W7/2),
(_W4*_W0/2),(_W4*_W1/2),(_W4*_W2/2),(_W4*_W3/2),(_W4*_W4/2),(_W4*_W5/2),(_W4*_W6/2),(_W4*_W7/2),
(_W6*_W0/2),(_W6*_W1/2),(_W6*_W2/2),(_W6*_W3/2),(_W6*_W4/2),(_W6*_W5/2),(_W6*_W6/2),(_W6*_W7/2),
};

static Ipp16s TablW1_Scale_ZigZag[] =
{
(Ipp16s)(8192.0/TablW1[0]), (Ipp16s)(8192.0/TablW1[32]),(Ipp16s)(8192.0/TablW1[1]),
(Ipp16s)(8192.0/TablW1[33]),
(Ipp16s)(8192.0/TablW1[8]), (Ipp16s)(8192.0/TablW1[40]),(Ipp16s)(8192.0/TablW1[2]),
(Ipp16s)(8192.0/TablW1[34]),
(Ipp16s)(8192.0/TablW1[9]),
(Ipp16s)(8192.0/TablW1[41]),(Ipp16s)(8192.0/TablW1[16]),(Ipp16s)(8192.0/TablW1[48]),
(Ipp16s)(8192.0/TablW1[24]),(Ipp16s)(8192.0/TablW1[56]),(Ipp16s)(8192.0/TablW1[17]),(Ipp16s)(8192.0/TablW1[49]),
(Ipp16s)(8192.0/TablW1[10]),(Ipp16s)(8192.0/TablW1[42]),(Ipp16s)(8192.0/TablW1[3]),
(Ipp16s)(8192.0/TablW1[35]),
(Ipp16s)(8192.0/TablW1[4]),
(Ipp16s)(8192.0/TablW1[36]),(Ipp16s)(8192.0/TablW1[11]),(Ipp16s)(8192.0/TablW1[43]),
(Ipp16s)(8192.0/TablW1[18]),(Ipp16s)(8192.0/TablW1[50]),(Ipp16s)(8192.0/TablW1[25]),(Ipp16s)(8192.0/TablW1[57]),
(Ipp16s)(8192.0/TablW1[26]),(Ipp16s)(8192.0/TablW1[58]),(Ipp16s)(8192.0/TablW1[19]),(Ipp16s)(8192.0/TablW1[51]),
(Ipp16s)(8192.0/TablW1[12]),(Ipp16s)(8192.0/TablW1[44]),(Ipp16s)(8192.0/TablW1[5]),
(Ipp16s)(8192.0/TablW1[37]),
(Ipp16s)(8192.0/TablW1[6]),
(Ipp16s)(8192.0/TablW1[38]),(Ipp16s)(8192.0/TablW1[13]),(Ipp16s)(8192.0/TablW1[45]),
(Ipp16s)(8192.0/TablW1[20]),(Ipp16s)(8192.0/TablW1[52]),(Ipp16s)(8192.0/TablW1[27]),(Ipp16s)(8192.0/TablW1[59]),
(Ipp16s)(8192.0/TablW1[28]),(Ipp16s)(8192.0/TablW1[60]),(Ipp16s)(8192.0/TablW1[21]),(Ipp16s)(8192.0/TablW1[53]),
(Ipp16s)(8192.0/TablW1[14]),(Ipp16s)(8192.0/TablW1[46]),(Ipp16s)(8192.0/TablW1[7]),
(Ipp16s)(8192.0/TablW1[39]),
(Ipp16s)(8192.0/TablW1[15]),(Ipp16s)(8192.0/TablW1[47]),(Ipp16s)(8192.0/TablW1[22]),(Ipp16s)(8192.0/TablW1[54]),
(Ipp16s)(8192.0/TablW1[29]),(Ipp16s)(8192.0/TablW1[61]),(Ipp16s)(8192.0/TablW1[30]),(Ipp16s)(8192.0/TablW1[62]),
(Ipp16s)(8192.0/TablW1[23]),(Ipp16s)(8192.0/TablW1[55]),(Ipp16s)(8192.0/TablW1[31]),(Ipp16s)(8192.0/TablW1[63])
};

```

The code example below shows how to use `ippiQuantInv_DV_16s_C1I` function.

```

{
    ippiHuffmanDecodeSegment_DV_8u16s((short* ) pDataBlock, ...);
    // get pointer to corresponding dequantize table
    // depends on qno & block quant class
    pQuantTable = (...);
    if (0 == block_type)
    {
        for (Ipp32s iI = 0; iI < 64; iI++)
            pQuantTable[iI] = pQuantTable[iI] * (Ipp16s)((double) (1 << 8) /
            TablW0[tabl1[iI]]);

        // do inverse and adaptive dequantization
        ippiQuantInv_DV_16s_C1I((short* ) pDataBlock), (short* ) pQuantTable);
        // do iDCT
        ippiDCT8x8Inv_16s_C1I((short* ) (pDataBlock));
    }

    else
    {

```

```

    for (Ipp32s iI = 0; iI < 64; iI++)
        pQuantTable [iI] = pQuantTable [iI]* (Ipp16s)((double) (1 << 8)) /
        TablWl[tabl2[iI]];

    // do inverse and adaptive dequantization
    ippiQuantInv_DV_16s_C1I((short* ) pDataBlock), (short* ) pQuantTable);
    // do iDCT
    ippiDCT2x4x8Inv_16s_C1I((short* ) (pDataBlock) );
}
};

```

The code example below shows how to use `ippiQuantWeightBlockInv_DV_16s_C1I`.

```

{
    ippiHuffmanDecodeSegment_DV_8ul6s((short *) lpsBlocks,...);
    // get pointer to corresponding dequantization table
    // depends on gno & block quant class
    pQuantTable = (...);

    if (0 == block_type)
    {
        ippiQuantWeightBlockInv_DV_16s_C1I( (Ipp16s *) (pDataBlock),
                                             (Ipp16s *) pQuantTable,
                                             (Ipp16s *) TablW0_Scale_ZigZag );

        // do iDCT
        ippiDCT8x8Inv_16s_C1I((short *) (pDataBlock));
    }

    else
    {
        ippiQuantWeightBlockInv_DV_16s_C1I( (Ipp16s *) (pDataBlock),
                                             (Ipp16s *) pQuantTable,
                                             (Ipp16s *) TablW0_Scale_ZigZag );

        // do iDCT
        ippiDCT2x4x8Inv_16s_C1I((short *) (pDataBlock));
    }
};

```

## QuantWeightBlockInv\_DV100

*Performs inverse quantization and inverse weighting on a block according to DV100 standard.*

---

### Syntax

```

IppStatus ippiQuantWeightBlockInv_DV100_16s_C1I(Ipp16s* pSrcDst, const Ipp16s*
pWeightInvTable, Ipp32s quantValue);

```

### Parameters

<code>pSrcDst</code>	Pointer to the source and destination 8x8 block.
<code>pWeightInvTable</code>	Pointer to an array that contains values of weight table for the current block.
<code>quantValue</code>	Quantification step for the current block.



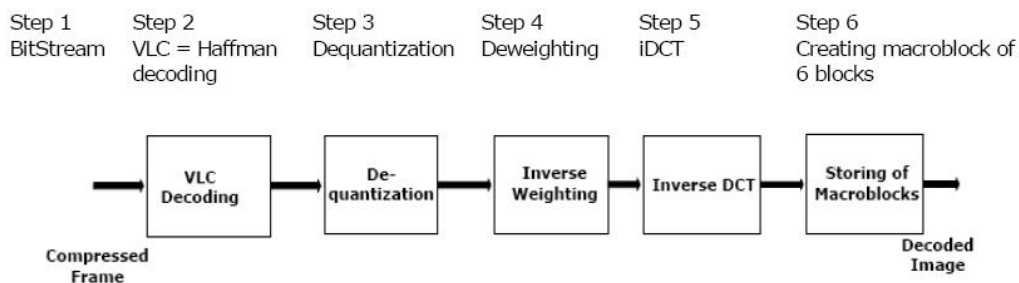
**NOTE.** This standard deals with only one block type - 8x8 block.

## Description

This function is declared in the `ippvc.h` header file. The function `ippiQuantWeightBlockInv_DV100_16s_C1I` performs inverse quantization and inverse weighting on a block in accordance with SMPTE 370M-2006 (DV100) [SMPTE370M-06] standard.

See "DV100 One Block Decoding" for the DV100 frame decoding process.

## DV100 One Block Decoding



Step 1 passes the pointer to the required stream. Step 2 is segment decoding. The `HuffmanDecodeSegment_DV(...)` function decodes the block along with multiplying its elements by 64, that is, performs  $\ll 6$  shift. The `ippiQuantWeightBlockInv_DV100_16s_C1I(...)` function combines Steps 3 and 4. Choose appropriate quantization and weighting tables, that is, dequantization and deweighting tables for decoding. To choose the quantization table, use the values of two parameters - *QNO* (quantization number) and the appropriate block quant class number. For details, see [SMPTE370M-06], Table 25 - Quantization step.

The `ippiQuantWeightBlockInv_DV100_16s_C1I(...)` function uses the following three pointers as input arguments:

<i>pDataBlock</i>	Pointer to the current block
<i>quantValue</i>	Quantification step for the current block
<i>pWeightTable</i>	Pointer to the weighting table

The function performs the following operations:

The function performs the following operations:

```

for (Ipp32s i = 0; i < 64; i+= 1)
    pDataBlock[i] = (Ipp16s) (
        ( pDataBlock[i]* pWeightTable[i] ) )
        * quantValue )>>11 );
  
```

After `ippiQuantWeightBlockInv_DV100_16s_C1I(...)` completes Steps 3 and 4, Step 5 of iDCT is performed before Step 6.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when the <code>pSrcDst</code> pointer is NULL.

## Example

The code example below shows how to use `ippiQuantWeightBlockInv_DV100_16s_C1I` function.

```
{
//For more details see SMPTE 370M-2002 standard.
//You can similarly get tables LumaQuantizeMatrix_720System
//and ChromaQuantizeMatrix_720System.
const Ipp32s LumaQuantizeMatrix_1080System[64] =
{
    128, 16, 17, 18, 18, 19, 42, 44,
    16, 17, 18, 18, 19, 38, 43, 45,
    17, 18, 19, 19, 40, 41, 45, 48,
    18, 18, 19, 40, 41, 42, 46, 49,
    18, 19, 40, 41, 42, 43, 48, 101,
    19, 38, 41, 42, 43, 44, 98, 104,
    42, 43, 45, 46, 48, 98, 109, 116,
    44, 45, 48, 49, 101, 104, 116, 123
};

const Ipp32s ChromaQuantizeMatrix_1080System[64]=
{
    128, 16, 17, 25, 26, 26, 42, 44,
    16, 17, 25, 25, 26, 38, 43, 91,
    17, 25, 26, 27, 40, 41, 91, 96,
    25, 25, 27, 40, 41, 84, 93, 197,
    26, 26, 40, 41, 84, 86, 191, 203,
    26, 38, 41, 84, 86, 177, 197, 209,
    42, 43, 91, 93, 191, 197, 219, 232,
    44, 91, 96, 197, 203, 209, 232, 246
};

//For more details see SMPTE 370M-2002 standard
static const Ipp32s QuantizationSteps[] =
{
    //Class Number
    //0    1    2    3    //Quantization Number (QNO)
    0,    0,    0,    0,    // 0
    1,    2,    4,    8,    // 1
    2,    4,    8,    0,    // 2
    3,    6,   12,    0,    // 3
    4,    8,    0,    0,    // 4
    5,   10,    0,    0,    // 5
    6,   12,    0,    0,    // 6
    7,   14,    0,    0,    // 7
    8,    0,    0,    0,    // 8
    16,   32,   64,    0,    // 9
    18,   36,   72,    0,    // 10
    20,   40,   80,    0,    // 11
    22,   44,   88,    0,    // 12
    24,   48,   96,    0,    // 13
    28,   56,  112,    0,    // 14
    52,  104,    0,    0,    // 15
};
```

```

/*
.....
Before quantization and weighting, you should perform Huffman decoding
.....
*/
//nMBNum - number of the current macroblock
int nMBNum;
//current DCT block in macroblock
int nDCTBlockNum;
//array of BlockParam for each block from 5 macroblock (8 * 5 = 40)
BlockParam BlParam[40];
//Pointer to current DCT block in macroblock
Ipp16s* pCurrDCTBlock;

for(nMBNum = 0; nMBNum < 5; nMBNum++)
{
    Ipp8u QNO = BlParam[nMBNum * 8].qno;

    for(nDCTBlockNum = 0; nDCTBlockNum < 8; nDCTBlockNum++)
    {
        Ipp8u ClassNumber = BlParam[nMBNum * 8 + nDCTBlockNum].cl;
        Ipp32s QuantStep = QuantizationSteps[QNO*4 + ClassNumber];
        const Ipp16s* pQuantizeTable;

        //Dequantize block
        //Select quantization table that depends on system type and type
        //of block (luma or chroma)
        if(m_System == System720_60p)
            pQuantizeTable = (nDCTBlockNum < 4) ?
                LumaQuantizeMatrix_720System : ChromaQuantizeMatrix_720System;
        else
            pQuantizeTable = (nDCTBlockNum < 4) ?
                LumaQuantizeMatrix_1080System : ChromaQuantizeMatrix_1080System;

        //Do quantization and weighting
        //For more performance you should align pCurrDCTBlock and
        //pQuantizeTable pointers on a 16-byte boundary.
        ippiQuantWeightBlockInv_DV100_16s_C1I(pCurrDCTBlock, (Ipp16s
            * )pQuantizeTable, QuantStep);

        //Do inverse DCT
        ippiDCT8x8Inv_16s_C1I(pCurrDCTBlock);

        pCurrDCTBlock += 64;
    } //for nDCTBlockNum = 0 to 7
}
};

```

## Inverse Discrete Cosine Transformation

### DCT2x4x8Inv

*Performs the inverse DCT for block of type 2 (m0=1).*

#### Syntax

```
IppStatus ippiDCT2x4x8Inv_16s_C1I(Ipp16s* pSrcDst);
```

## Parameters

*pSrcDst*                                      Pointer to the block.

## Description

This function is declared in the `ippvc.h` header file. The function `ippiDCT2x4x8Inv_16s_C1I` performs the inverse DCT for a block of type 2 ( $m0 = 1$ ). Formula for inverse DCT is as follows:

$$P(x, 2^*z) = \sum_{u=0}^3 \sum_{h=0}^7 (c(u)c(h)(c(h, u) + c(h, u + 4))KC)$$

$$P(x, 2^*z + 1) = \sum_{u=0}^3 \sum_{h=0}^7 (c(u)c(h)(c(h, u) - c(h, u + 4))KC) ,$$

where,

$$x \text{ in } [0, 7]$$

$$y \text{ in } [0, 7]$$

$$z = \text{int}\left(\frac{Y}{2}\right)$$

$$KC = \cos\left(\frac{\pi u(2z + 1)}{8}\right) \cos\left(\frac{\pi h(2x + 1)}{16}\right)$$

$$c(h) = \frac{0.5}{\text{sqr}(2)} \quad \text{for } h = 0,$$

$$c(h) = 0.5 \quad \text{for } h \text{ in } [1, 7],$$

$$c(u) = \frac{0.5}{\text{sqr}(2)} \quad \text{for } u = 0,$$

$$c(u) = 0.5 \quad \text{for } u \text{ in } [1, 7]$$

This function is used in the DV decoder included into Intel IPP Samples. See [introduction](#) to this section.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when <code>pSrcDst</code> is NULL.

## DCT8x4x2To4x4Inv\_DV

*Performs the inverse DCT for block 2x4x8 and then creates block 4x4.*

---

## Syntax

```
IppStatus ippiDCT8x4x2To4x4Inv_DV_16s_C1I(Ipp16s* pSrcDst);
```

## Parameters

`pSrcDst` Pointer to the source and destination block.



---

**NOTE.** There are two different types of blocks here: 8x8 (or 2x4x8) block as source and 4x4 (or 2x2) as destination.

---

## Description

This function is declared in the `ippvc.h` header file. The function `ippiDCT8x4x2To4x4Inv_DV_16s_C1I` performs the inverse DCT for a 2x4x8 block, then creates a 4x4 block: first, values in rows are averaged in pairs, then values in columns are averaged in pairs too. See "[Creating 4x4 Block from 8x8 Block](#)" for the new block values.

The `pSrcDst` parameter is a pointer to a block of 2x4x8 type. The function performs inverse DCT of the block and averages, first, two adjacent row elements and then two adjacent column elements. The averaging produces 16 elements all in all.

The resulting elements are stored in the memory in a row and without gaps (see "Storing New 4x4 Values" , AverageXX =AveXX) so that after the function is called the *pSrcDst* pointer points to the first element of the sixteen averaged elements.

### Creating 4x4 Block from 8x8 Block

	0	1	2	3	4	5	6	7
0	Average00	Average01	Average02	Average03				
1								
2	Average04	Average05	Average06	Average07				
3								
4	Average08	Average09	Average10	Average11				
5								
6	Average12	Average13	Average14	Average15				
7								

### Storing New 4x4 Values

↓ <i>pSrcDst</i>							
	Ave00	Ave01	Ave02	Ave03	Ave04	...	Ave15

### Return Values

*ippStsNoErr* Indicates no error.

*ippStsNullPtrErr* Indicates an error when *pSrcDst* is NULL.

### Example

The code example below shows how to use *ippiDCT8x4x2To4x4Inv\_DV\_16s\_C1I* function.

```
...
//mb_num = <current MacroBlock>;
//b_num = <current Block>;
//block_type - type of block
//lpBlocks - pointer on current block (8x8 or 2x4x8)
Ipp32s block_type = BlParam[mb_num * 6 + b_num].m0;

// do iDCT, normal decoding, full resolution
```



```

if (0 == block_type)
    ippiDCT8x8Inv_16s_C1I((short *) (lpsBlocks));
else
    ippiDCT2x4x8Inv_16s_C1I((short *) (lpsBlocks));
...

//If you want to get half resolution (1/2), use these functions:

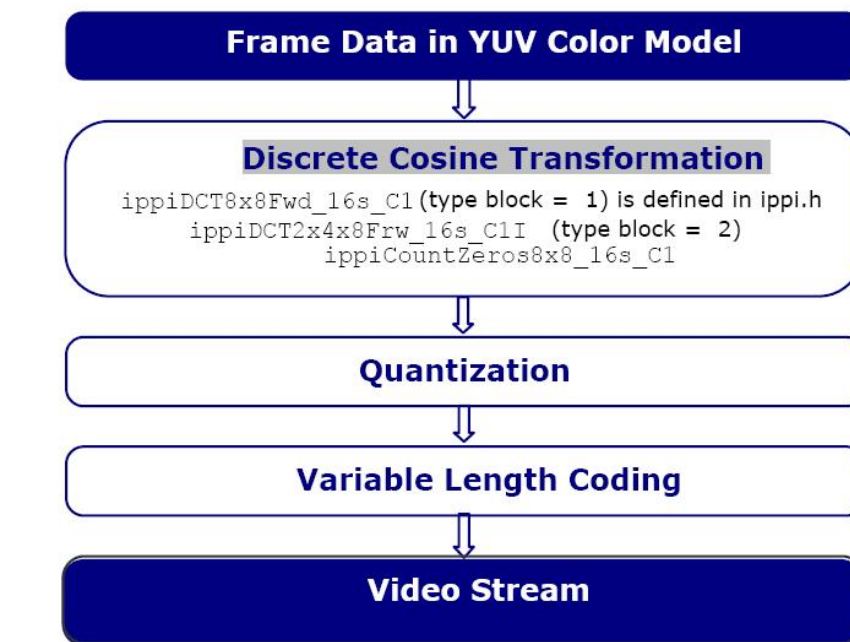
// do iDCT, half (1/2)resolution
if (0 == block_type)
    //ippiDCT8x8Inv_16s_C1I((short *) (lpsBlocks));
    ippiDCT8x8To4x4Inv_16s_C1I((short *) (lpsBlocks));
    // function ippiDCT8x8To4x4Inv_16s_C1I() defined in ippi domain, and not ipmvc!
else
    //ippiDCT8x8Inv_16s_C1I((short *) (lpsBlocks));
    ippiDCT8x4x2To4x4Inv_DV_16s_C1I ((short *) (lpsBlocks));

//After that, you increase performance and get half resolution picture.

```

## DV Encoding Functions

### DV Encoding Pipeline



## Discrete Cosine Transformation

### DCT2x4x8Fw

*Performs DCT for a block of type 2.*

---

#### Syntax

```
IppStatus ippiDCT2x4x8Fw_16s_C1I(Ipp16s* pSrcDst);
```

#### Parameters

*pSrcDst*                                      Pointer to the block.

#### Description

This function is declared in the `ippvc.h` header file. The function `ippiDCT2x4x8Fw_16s_C1I` performs the DCT for block of type 2. Formula for DCT is as follows:

$$C(h, u) = c(u)c(h) \sum_{z=0}^3 \sum_{x=0}^7 ((P(x, 2z) + P(x, 2z + 1))KC)$$

$$C(h, u + 4) = c(u)c(h) \sum_{z=0}^3 \sum_{x=0}^7 ((P(x, 2z) - P(x, 2z + 1))KC) ,$$

where

$$h \text{ in } [0, 7]$$

$$u \text{ in } [0, 7]$$

$$z = \text{int}\left(\frac{Y}{2}\right)$$

$$KC = \cos\left(\frac{\pi u(2z + 1)}{8}\right) \cos\left(\frac{\pi h(2x + 1)}{16}\right)$$

$$\begin{aligned}
 c(h) &= \frac{0.5}{\text{sqr}(2)} && \text{for } h = 0, \\
 c(h) &= 0.5 && \text{for } h \text{ in } [1, 7], \\
 c(u) &= \frac{0.5}{\text{sqr}(2)} && \text{for } u = 0, \\
 c(u) &= 0.5 && \text{for } u \text{ in } [1, 7].
 \end{aligned}$$

### Return Values

`ippStsNoErr` Indicates no error.  
`ippStsNullPtrErr` Indicates an error when `pSrcDst` is NULL.

## CountZeros8x8

*Evaluates number of zeros in a block.*

---

### Syntax

```
IppStatus ippiCountZeros8x8_16s_C1(Ipp16s* pSrc, Ipp32u* pCount);
```

### Parameters

`pSrc` Pointer to the block.  
`pCount` Pointer to the number of zeros.

### Description

This function is declared in the `ippvc.h` header file. The function `ippiCountZeros8x8_16s_C1` is used to identify the type of a block by comparing the results from DCT of the first type and DCT of the second type performed on the block.

### Return Values

`ippStsNoErr` Indicates no error.  
`ippStsNullPtrErr` Indicates an error when at least one of the pointers is NULL.

### Example

The code example below shows how to use `ippiCountZeros8x8` function, where `pBlock` is the pointer to the current block and `WeightTableType0` represents tables of the weight coefficients for discrete cosine transform (see [SMPTE314M] Chapters 5.2 DCT Mode and 5.2.2 Weighting). The coefficients are real numbers that are multiplied by 32767 and reduced to short type through scaling by  $2^{15}$ .

```
Ipp16s sTmp[64];
Ipp32u countzero, countzerol;
```

```

memcpy(sTmp, pBlock->m_psData, 64 * sizeof(Ipp16s));
ippiDCT8x8Fwd_16s_C1I(pBlock->m_psData);
ippsMul_16s_ISfs(WeightTableType0, pBlock->m_psData, 64, 16);

ippiDCT2x4x8Fwd_16s_C1I(sTmp);
ippsMul_16s_ISfs(WeightTableType1, sTmp, 64, 16);

ippiCountZeros8x8_16s_C1((Ipp16s *) (pBlock->m_psData), &countzero);
ippiCountZeros8x8_16s_C1((Ipp16s *) sTmp, &countzero1);
if (countzero >= countzero1)
{
    pBlock->m_cM0 = 0;
}
else
{
    pBlock->m_cM0 = 1;
    memcpy(pBlock->m_psData, sTmp, 64 * sizeof(Ipp16s));
}

```

## DV Color Conversion Functions

These functions convert color format in DV video processing.

The minimal coding unit in DV is a set of five macroblocks, with each macroblock comprising four Y-blocks, one Cb block, and one Cr block. The macroblocks are taken from different parts of the processed frame to decrease loss of quality in case the film is damaged. The data format for a DV frame is

- YUV411 in system 525, or NTSC,
- interlaced YUV420 in system 625, or PAL.

These formats are converted into a more popular YUY2 format after inverse discrete cosine transformation.

The functions described in this subsection are used in the DV decoder included into Intel IPP Samples. See [introduction](#) to the DV section.

## YCrCb411ToYCbCr422\_5MBDV, YCrCb411ToYCbCr422\_ZoomOut2\_5MBDV, YCrCb411ToYCbCr422\_ZoomOut4\_5MBDV, YCrCb411ToYCbCr422\_ZoomOut8\_5MBDV

*Convert five YCrCb411 macroblocks into YCrCb422 macroblocks.*

---

### Syntax

```

IppStatus ippiYCrCb411ToYCbCr422_5MBDV_16s8u_P3C2R(const Ipp16s* pSrc[5], Ipp8u* pDst[5],
int dstStep);

```

```

IppStatus ippiYCrCb411ToYCbCr422_ZoomOut2_5MBDV_16s8u_P3C2R(const Ipp16s* pSrc[5], Ipp8u*
pDst[5], int dstStep);

```

```

IppStatus ippiYCrCb411ToYCbCr422_ZoomOut4_5MBDV_16s8u_P3C2R(const Ipp16s* pSrc[5], Ipp8u*
pDst[5], int dstStep);

```

```

IppStatus ippiYCrCb411ToYCbCr422_ZoomOut8_5MBDV_16s8u_P3C2R(const Ipp16s* pSrc[5], Ipp8u*
pDst);

```

## Parameters

<i>pSrc</i>	Array of pointers to the five source macroblocks.
<i>pDst</i>	Array of pointers to the five destination macroblocks.
<i>dstStep</i>	Distance in bytes between starts of the consecutive lines in the destination image.

## Description

These functions are declared in the `ippvc.h` header file. The functions convert YCrCb411 macroblocks to YCbCr422 macroblocks, that is, they transfer the five macroblocks of uncompressed DV data after IDCT processing from the internal format YUV411 to the destination buffer in YUY2 format with saturation.

The function `ippiYCrCb411ToYCbCr422_ZoomOut2_5MBDV_16s8u_P3C2R` also reduces the size of the destination image by 2 times, `ippiYCrCb411ToYCbCr422_ZoomOut4_5MBDV_16s8u_P3C2R` - by 4 times, and `ippiYCrCb411ToYCbCr422_ZoomOut8_5MBDV_16s8u_P3C2R` - by 8 times.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when at least one of the pointers is <code>NULL</code> .

## YCrCb411ToYCbCr422\_16x4x5MB\_DV, YCrCb411ToYCbCr422\_8x8MB\_DV

*Convert five reduced YCrCb411 macroblocks into YCrCb422 reduced macroblocks.*

## Syntax

```
ippStatus ippiYCrCb411ToYCbCr422_16x4x5MB_DV_16s8u_P3C2R(const Ipp16s* pSrc[5], Ipp8u* pDst[5], int dstStep);

ippStatus ippiYCrCb411ToYCbCr422_8x8MB_DV_16s8u_P3C2R(const Ipp16s* pSrc, Ipp8u* pDst, int dstStep);
```

## Parameters

<i>pSrc</i>	Array of pointers to the five reduced source macroblocks. For the 8x8MB function, <i>pSrc</i> is a single pointer to the reduced source macroblock.
<i>pDst</i>	Array of pointers to the five destination macroblocks. For the 8x8MB function, <i>pDst</i> is a single pointer to the destination macroblock.
<i>dstStep</i>	Distance in bytes between starts of the consecutive lines in the destination image.

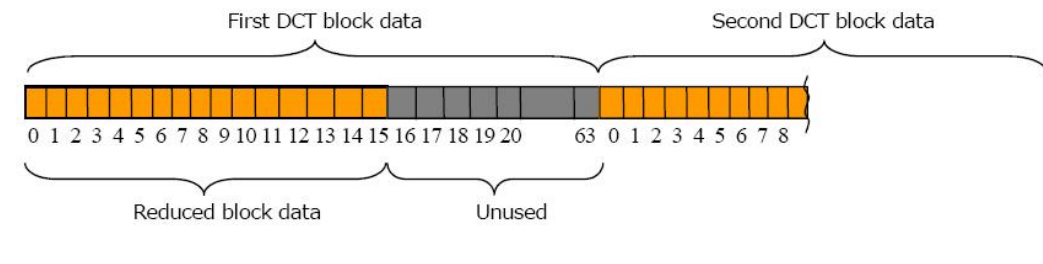
## Description

These functions are declared in the `ippvc.h` header file. The functions convert reduced (quarter resolution) YCrCb411 macroblocks to YCbCr422 macroblocks of the same size. Specifically, the functions transfer the five reduced macroblocks of uncompressed DV data after IDCT processing and block size reduction from the internal format YUV411 to the destination buffer in YUY2 format with saturation.

The function `ippiYCrCb411ToYCbCr422_16x4x5MB_DV_16s8u_P3C2R` converts macroblocks of 16x4 size, and `ippiYCrCb411ToYCbCr422_8x8MB_DV_16s8u_P3C2R` converts a macroblock of 8x8 size located at the right edge of the destination image. The latter function is intended for NTSC standard conversion since all right edge macroblocks have slightly different shapes.

Note that each pointer in the `pSrc` array refers to the uncompressed macroblock data, which consists of 6 uncompressed DCT block data areas. Each DCT block data follows one after another and occupies 64 elements (128 bytes). Reduced uncompressed blocks of size 4x4 occupy first 16 elements of each DCT block data area. See "[Layout of Reduced DCT Blocks in Macroblock Data](#)" for the layout of the reduced blocks in the macroblock data.

### Layout of Reduced DCT Blocks in Macroblock Data



### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when at least one of the pointers is NULL.

### YCrCb411ToYCbCr422\_EdgeDV, YCrCb411ToYCbCr422\_ZoomOut2\_EdgeDV, CrCb411ToYCbCr422\_ZoomOut4\_EdgeDV, YCrCb411ToYCbCr422\_ZoomOut8\_EdgeDV

*Convert a YCrCb411 macroblock into a YCrCb422 macroblock at the right edge of destination image.*

### Syntax

```

IppStatus ippiYCrCb411ToYCbCr422_EdgeDV_16s8u_P3C2R(const Ipp16s* pSrc, Ipp8u* pDst, int
dstStep);

IppStatus ippiYCrCb411ToYCbCr422_ZoomOut2_EdgeDV_16s8u_P3C2R(const Ipp16s* pSrc, Ipp8u*
pDst, int dstStep);

IppStatus ippiYCrCb411ToYCbCr422_ZoomOut4_EdgeDV_16s8u_P3C2R(const Ipp16s* pSrc, Ipp8u*
pDst, int dstStep);

IppStatus ippiYCrCb411ToYCbCr422_ZoomOut8_EdgeDV_16s8u_P3C2R(const Ipp16s* pSrc, Ipp8u*
pDst, int dstStep);

```

## Parameters

<i>pSrc</i>	Pointer to the source macroblock.
<i>pDst</i>	Pointer to the destination macroblock.
<i>dstStep</i>	Distance in bytes between starts of the consecutive lines in the destination image.

## Description

These functions are declared in the `ippvc.h` header file. They convert a YCrCb411 macroblock into a YCrCb422 macroblock at the right edge of destination image. These functions are intended for NTSC standard conversion as all right edge macroblocks in NTSC have slightly different locations.

The function `ippiYCrCb411ToYCbCr422_ZoomOut2_EdgeDV_16s8u_P3C2R` also reduces the size of the destination image by 2 times, `ippiYCrCb411ToYCbCr422_ZoomOut4_EdgeDV_16s8u_P3C2R` - by 4 times, and `ippiYCrCb411ToYCbCr422_ZoomOut8_EdgeDV_16s8u_P3C2R` - by 8 times.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when at least one of the pointers is NULL.

## YCrCb420ToYCbCr422\_5MBDV, YCrCb420ToYCbCr422\_ZoomOut2\_5MBDV, YCrCb420ToYCbCr422\_ZoomOut4\_5MBDV, YCrCb420ToYCbCr422\_ZoomOut8\_5MBDV

*Convert five YCrCb420 macroblocks into YCrCb422 macroblocks.*

## Syntax

```
IppStatus ippiYCrCb420ToYCbCr422_5MBDV_16s8u_P3C2R(const Ipp16s* pSrc[5], Ipp8u* pDst[5], int dstStep);
```

```
IppStatus ippiYCrCb420ToYCbCr422_ZoomOut2_5MBDV_16s8u_P3C2R(const Ipp16s* pSrc[5], Ipp8u* pDst[5], int dstStep);
```

```
IppStatus ippiYCrCb420ToYCbCr422_ZoomOut4_5MBDV_16s8u_P3C2R(const Ipp16s* pSrc[5], Ipp8u* pDst[5], int dstStep);
```

```
IppStatus ippiYCrCb420ToYCbCr422_ZoomOut8_5MBDV_16s8u_P3C2R(const Ipp16s* pSrc[5], Ipp8u* pDst[5], int dstStep);
```

## Parameters

<i>pSrc</i>	Array of pointers to the five source macroblocks.
<i>pDst</i>	Array of pointers to the five destination macroblocks.
<i>dstStep</i>	Distance in bytes between starts of the consecutive lines in the destination image.

## Description

These functions are declared in the `ippvc.h` header file. The functions convert YCrCb420 macroblocks to YCbCr422 macroblocks, that is, they transfer the five macroblocks of uncompressed DV data after IDCT processing from the internal format YUV420 (PAL format) to the destination buffer in YUY2 format with saturation.

The function `ippiYCrCb420ToYCbCr422_ZoomOut2_5MBDV_16s8u_P3C2R` also reduces the size of the destination image by 2 times, `ippiYCrCb420ToYCbCr422_ZoomOut4_5MBDV_16s8u_P3C2R` - by 4 times, and `ippiYCrCb420ToYCbCr422_ZoomOut8_5MBDV_16s8u_P3C2R` - by 8 times.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when at least one of the pointers is <code>NULL</code> .

## YCrCb420ToYCbCr422\_8x8x5MB\_DV

*Converts five reduced YCrCb420 macroblocks into YCrCb422 reduced macroblocks.*

---

## Syntax

```
ippStatus ippiYCrCb420ToYCbCr422_8x8x5MB_DV_16s8u_P3C2R(const Ipp16s* pSrc[5], Ipp8u* pDst[5], int dstStep);
```

## Parameters

<code>pSrc</code>	Array of pointers to the five reduced source macroblocks.
<code>pDst</code>	Array of pointers to the five destination macroblocks.
<code>dstStep</code>	Distance in bytes between starts of the consecutive lines in the destination image.

## Description

This function is declared in the `ippvc.h` header file. The function converts reduced (quarter resolution) YCrCb420 macroblocks of 8x8 size to YCbCr422 macroblocks of the same size. Specifically, the function transfers the five reduced macroblocks of uncompressed DV data after IDCT processing and block size reduction from the internal format YUV420 to the destination buffer in YUY2 format with saturation.

Note that each pointer in the `pSrc` array refers to the uncompressed macroblock data, which consists of 6 uncompressed DCT block data areas. Each DCT block data follows one after another and occupies 64 elements (128 bytes). Reduced uncompressed blocks of size 4x4 occupy first 16 elements of each DCT block data area. See "[Layout of Reduced DCT Blocks in Macroblock Data](#)" for the layout of the reduced blocks in the macroblock data.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when at least one of the pointers is <code>NULL</code> .



## YCrCb422ToYCbCr422\_5MBDV, YCrCb422ToYCbCr422\_ZoomOut2\_5MBDV, YCrCb422ToYCbCr422\_ZoomOut4\_5MBDV, YCrCb422ToYCbCr422\_ZoomOut8\_5MBDV

Convert five YCrCb422 macroblocks into YCrCb422 macroblocks.

### Syntax

```
IppStatus ippiYCrCb422ToYCbCr422_5MBDV_16s8u_P3C2R(const Ipp16s* pSrc[5], Ipp8u* pDst[5],
int dstStep);

IppStatus ippiYCrCb422ToYCbCr422_ZoomOut2_5MBDV_16s8u_P3C2R(const Ipp16s* pSrc[5], Ipp8u*
pDst[5], int dstStep);

IppStatus ippiYCrCb422ToYCbCr422_ZoomOut4_5MBDV_16s8u_P3C2R(const Ipp16s* pSrc[5], Ipp8u*
pDst[5], int dstStep);

IppStatus ippiYCrCb422ToYCbCr422_ZoomOut8_5MBDV_16s8u_P3C2R(const Ipp16s* pSrc[5], Ipp8u*
pDst[5]);
```

### Parameters

<i>pSrc</i>	Array of pointers to the five source macroblocks.
<i>pDst</i>	Array of pointers to the five destination macroblocks.
<i>dstStep</i>	Distance in bytes between starts of the consecutive lines in the destination image.

### Description

These functions are declared in the `ippvc.h` header file. They convert YCrCb422 macroblocks to YCbCr422 macroblocks. These functions may be used in DV50 decoders.

The function `ippiYCrCb422ToYCbCr422_ZoomOut2_5MBDV_16s8u_P3C2R` also reduces the size of the destination image by 2 times, `ippiYCrCb422ToYCbCr422_ZoomOut4_5MBDV_16s8u_P3C2R` - by 4 times, and `ippiYCrCb422ToYCbCr422_ZoomOut8_5MBDV_16s8u_P3C2R` - by 8 times.

Note a specific feature of decoding in this case. In spite of the fact that the input data format is YUV422, the *pSrc* pointer should point to the macroblocks that contain the six blocks in the following order:

```
[Y0 block][empty block][Y1 block][empty block][V block][U block].
```

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when at least one of the pointers is <code>NULL</code> .

### Example

The code example below shows how to use `ippiYCrCb422ToYCbCr422_5MBDV_16s8u_P3C2R` function.

...

```
const Ipp32s lMacroBlockWidth422 = 8 * 2;
```

```

const Ipp32s lMacroBlockHeight422 = 8;
const Ipp32s lSuperBlockWidth422 = (lMacroBlockWidth422 * 9);
const Ipp32s lSuperBlockHeight422 = (lMacroBlockHeight422 * 3);

...
{
    // Storing DV50 decompressed video segment

    // i is the DIF sequence number (from 0 to 19 in a NTSC mode or
    // from 0 to 23 in a PAL mode)
    // k is the compressed video segment number in the DIF sequence (from 0
    // to 26)
    // NumDIFSeq is a number of DIF sequences in a channel (10 in NTSC or
    // 12 in PAL mode)
    // Pitch is the destination image step
    // lpDstImage is the (Ipp8u*) pointer to the destination image buffer
    // lpsBlocks is the (Ipp16s*) pointer to the array of 30 decompressed DCT
    // blocks

    Ipp32s lRow, lCol, lBytesPerPixel;
    Ipp16s* lpSrc[5];
    Ipp8u * lpDst[5];

    // set source pointers
    lpSrc[0] = lpsBlocks + (6 * 64 * 0);
    lpSrc[1] = lpsBlocks + (6 * 64 * 1);
    lpSrc[2] = lpsBlocks + (6 * 64 * 2);
    lpSrc[3] = lpsBlocks + (6 * 64 * 3);
    lpSrc[4] = lpsBlocks + (6 * 64 * 4);

    // get current column
    lCol = k / 3;

    // get current row
    if (lCol & 0x01)
        lRow = 2 - (k % 3);
    else
        lRow = (k % 3);

    // set destination pointers
    // we use the following formula:
    // destination image buffer +
    //     required super block row offset +
    //     required super block column offset +
    //     required macro block row offset +
    //     required macro block column offset
    lBytesPerPixel = 2;
    // We use YUV2 destination format

    lpDst[0] = lpDstImage +
        (i / NumDIFSeq) * Pitch * lSuperBlockHeight422 +
        ((i + 2) % NumDIFSeq) * Pitch * lSuperBlockHeight422 * 2 +
        Pitch * lMacroBlockHeight422 * lRow +
        ((2) * lSuperBlockWidth422 +
        lMacroBlockWidth422 * lCol) * lBytesPerPixel;

```

## YCrCb422ToYCbCr422\_8x4x5MB\_DV

*Converts five reduced YCrCb422 macroblocks into YCrCb422 reduced macroblocks.*

---

### Syntax

```
IppStatus ippiYCrCb422ToYCbCr422_8x4x5MB_DV_16s8u_P3C2R(const Ipp16s* pSrc[5], Ipp8u* pDst[5], int dstStep);
```

### Parameters

<i>pSrc</i>	Array of pointers to the five reduced source macroblocks.
<i>pDst</i>	Array of pointers to the five destination macroblocks.
<i>dstStep</i>	Distance in bytes between starts of the consecutive lines in the destination image.

### Description

This function is declared in the `ippvc.h` header file. The function converts reduced (quarter resolution) YCrCb422 macroblocks of 8x4 size to YCbCr422 macroblocks of the same size. Specifically, the function transfers the five reduced macroblocks of uncompressed DV data after IDCT processing and block size reduction from the internal format YUV422 to the destination buffer in YUY2 format with saturation.

Note that each pointer in the *pSrc* array refers to the uncompressed macroblock data, which consists of 6 uncompressed DCT block data areas. Each DCT block data follows one after another and occupies 64 elements (128 bytes) but the second and the fourth areas are empty and not used. The block data are arranged in the following order:

```
[Y0 block][empty block][Y1 block][empty block][V block][U block].
```

Reduced uncompressed blocks of size 4x4 occupy first 16 elements of corresponding DCT block data area. See "[Layout of Reduced DCT Blocks in Macroblock Data](#)" for the layout of the reduced blocks in the macroblock data.

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when at least one of the pointers is <code>NULL</code> .

## YCrCb422ToYCbCr422\_10HalvesMB16x8\_DV100

*Converts ten YCrCb422 half-macroblocks into YCrCb422 half-macroblocks.*

---

### Syntax

```
IppStatus ippiYCrCb422ToYCbCr422_10HalvesMB16x8_DV100_16s8u_P3C2R(const Ipp16s* pSrc, Ipp8u* pDst[10], Ipp32s dstStep);
```

## Parameters

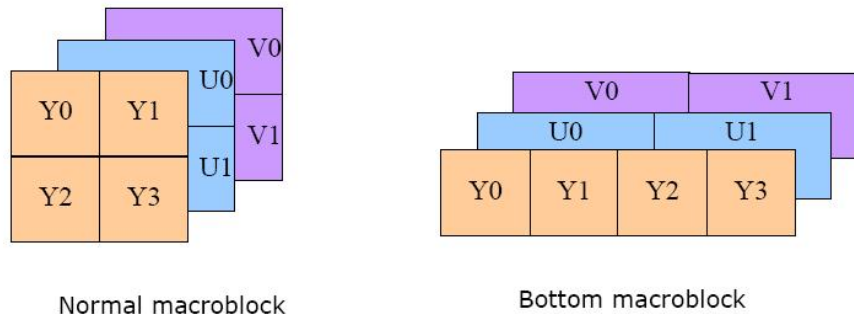
<i>pSrc</i>	Pointer to decompressed DV100 video segment consisting of five macroblocks.
<i>pDst</i>	Array of pointers to ten destinations, where half-macroblocks should be stored.
<i>dstStep</i>	Distance in bytes between starts of the consecutive lines in the destination image.

## Description

This function is declared in the `ippvc.h` header file. The `ippiYCrCb422ToYCbCr422_10HalvesMB16x8_DV100_16s8u_P3C2R` function is intended to be used in DV100 video decoder. It converts ten YCrCb422 half-macroblocks of size 16x8 to YCbCr422 half-macroblocks of the same size. Specifically, the function transfers macroblocks of uncompressed DV data after IDCT processing from the internal format (YUV422) to the destination buffer in YUY2 format with saturation.

There are two types of macro blocks in DV100 standard: normal and bottom macroblock. "DV 100 Macroblocks" depicts these two types. A bottom macroblock is obtained from a normal macroblock by moving its bottom half to the right.

## DV 100 Macroblocks ---



The `ippiYCrCb422ToYCbCr422_10HalvesMB16x8_DV100_16s8u_P3C2R` function can process both types of macroblocks by specifying ten destination pointers that are passed to the function through *pDst* argument. Each pair of pointers (even and odd) is the places on the destination image where two halves of the corresponding macroblock should be stored. For normal macroblock, the odd pointer should refer to the point 8 pixels below the point the even pointer refers to. For the bottom macroblock, the odd pointer should refer to the point 16 pixels to the right of the point the even pointer refers to.

*pSrc* is the pointer to uncompressed DV100 video segment, which consists of 40 DCT blocks following one after another: 4 luma and 4 color difference DCT blocks for each of 5 macro blocks. Each DCT block consists of 64 elements. DCT blocks in each macroblock are arranged in the following order:

Y0, Y1, Y2, Y3, Cr0, Cr1, Cb0, Cb1.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when at least one of the pointers is NULL.

## MPEG-4

This section contains functions for encoding and decoding of video data according to ISO/IEC 14496-2 MPEG-4 standard (see [ISO14496A]).

The use of some functions described in this section is demonstrated in Intel® IPP Samples downloadable from <http://www.intel.com/cd/software/products/asmo-na/eng/220046.htm>.

### MPEG-4 Video Decoder Functions

Function Short Name	Description
<b>Motion Compensation</b>	
<code>Copy8x8QP_MPEG4</code> , <code>Copy16x8QP_MPEG4</code> , <code>Copy16x16QP_MPEG4</code>	Copy a block with quarter-pixel accuracy.
<code>OBMC8x8HP_MPEG4</code> , <code>OBMC16x16HP_MPEG4</code> , <code>OBMC8x8QP_MPEG4</code>	Perform the overlapped block motion compensation (OBMC).
<b>Sprite and Global Motion Compensation</b>	
<code>WarpInit_MPEG4</code>	Initializes <code>IppiWarpSpec_MPEG4</code> structure for further usage in GMC or Sprite reconstruction.
<code>WarpGetSize_MPEG4</code>	Returns size of <code>IppiWarpSpec_MPEG4</code> structure.
<code>WarpLuma_MPEG4</code>	Warps arbitrary rectangular luminance region.
<code>WarpChroma_MPEG4</code>	Warps arbitrary rectangular chrominance region.
<code>CalcGlobalMV_MPEG4</code>	Calculates Global Motion Vector for one macroblock.
<code>ChangeSpriteBrightness_MPEG4</code>	Change brightness after sprite warping.
<b>Inverse Quantization</b>	
<code>QuantInvIntraInit_MPEG4</code> , <code>QuantInvInterInit_MPEG4</code>	Initialize specification structures for <code>QuantInvIntra_MPEG4</code> and <code>QuantInvInter_MPEG4</code> respectively.
<code>QuantInvIntraGetSize_MPEG4</code> , <code>QuantInvInterGetSize_MPEG4</code>	Return the size of the initialized specification structures.
<code>QuantInvIntra_MPEG4</code> , <code>QuantInvInter_MPEG4</code>	Perform inverse quantization on intra/inter coded block.
<b>VLC Decoding</b>	

Function Short Name	Description
<a href="#">DecodeDCIntra_MPEG4</a>	Decodes one DC coefficient for intra coded block.
<a href="#">DecodeCoefIsIntra_MPEG4</a>	Decodes DCT coefficients for intra coded block.
<a href="#">DecodeCoefIsIntraRVLCBack_MPEG4</a>	Decodes DCT coefficients in backward direction for intra coded block using RVLC.
<a href="#">DecodeCoefIsInter_MPEG4</a>	Decodes DCT coefficients for inter coded block.
<a href="#">DecodeCoefIsInterRVLCBack_MPEG4</a>	Decodes DCT coefficients in backward direction for inter coded block using RVLC.
<a href="#">ReconstructCoefIsInter_MPEG4</a>	Decodes DCT coefficients, performs inverse scan and inverse quantization for inter coded block.
<b>Postprocessing</b>	
<a href="#">FilterDeblocking8x8HorEdge_MPEG4</a> , <a href="#">FilterDeblocking8x8VerEdge_MPEG4</a>	Perform deblocking filtering on a horizontal or vertical edge of two adjacent blocks.
<a href="#">FilterDeringingThreshold_MPEG4</a>	Computes threshold values for the deringing filtering through a macroblock.
<a href="#">FilterDeringingSmooth8x8_MPEG4</a>	Performs deringing filtering of a block.
<b>MPEG-4 Video Encoder Functions</b>	
Function Short Name	Description
<b>Quantization</b>	
<a href="#">QuantIntraInit_MPEG4</a> , <a href="#">QuantInterInit_MPEG4</a>	Initialize specification structures for <a href="#">QuantIntra_MPEG4</a> and <a href="#">QuantInter_MPEG4</a> respectively.
<a href="#">QuantIntraGetSize_MPEG4</a> , <a href="#">QuantInterGetSize_MPEG4</a>	Return the size of the initialized specification structures.
<a href="#">QuantIntra_MPEG4</a> , <a href="#">QuantInter_MPEG4</a>	Perform quantization on intra/inter coded block.
<b>VLC Encoding</b>	
<a href="#">EncodeDCIntra_MPEG4</a>	Encodes one DC coefficient for intra coded block.
<a href="#">EncodeCoefIsIntra_MPEG4</a>	Encodes DCT coefficients for intra coded block.
<a href="#">EncodeCoefIsInter_MPEG4</a>	Encodes DCT coefficients for inter coded block.

## MPEG-4 Video Decoder Functions

This section describes Intel IPP functions that are built to support the ISO/IEC 14496-2 MPEG-4 video decoder. MPEG-4 (see [ISO14496]) is a widely used coding method for video signals in various applications such as digital storage media, Internet, various forms of wired or wireless communications, etc. The functions, along with those described in the [General Functions](#) section, cover the following aspects of MPEG-4 decoder:

- block-based variable length code (VLC) decoding and inverse zigzag scan
- Inverse quantization, inverse zigzag positioning, and IDCT
- Block layer coefficient reconstruction, including bitstream parsing, VLC decoding, inverse quantization, inverse zigzag positioning, with appropriate clipping on each step
- overlapped block motion compensation
- motion compensation
- postprocessing for coding noise reduction including deblocking and deringing filters.



---

**NOTE.** All MPEG-4 functions can be applied if `short_video_header` equals 0. If `short_video_header` equals 1, use H.263 decoder functions.

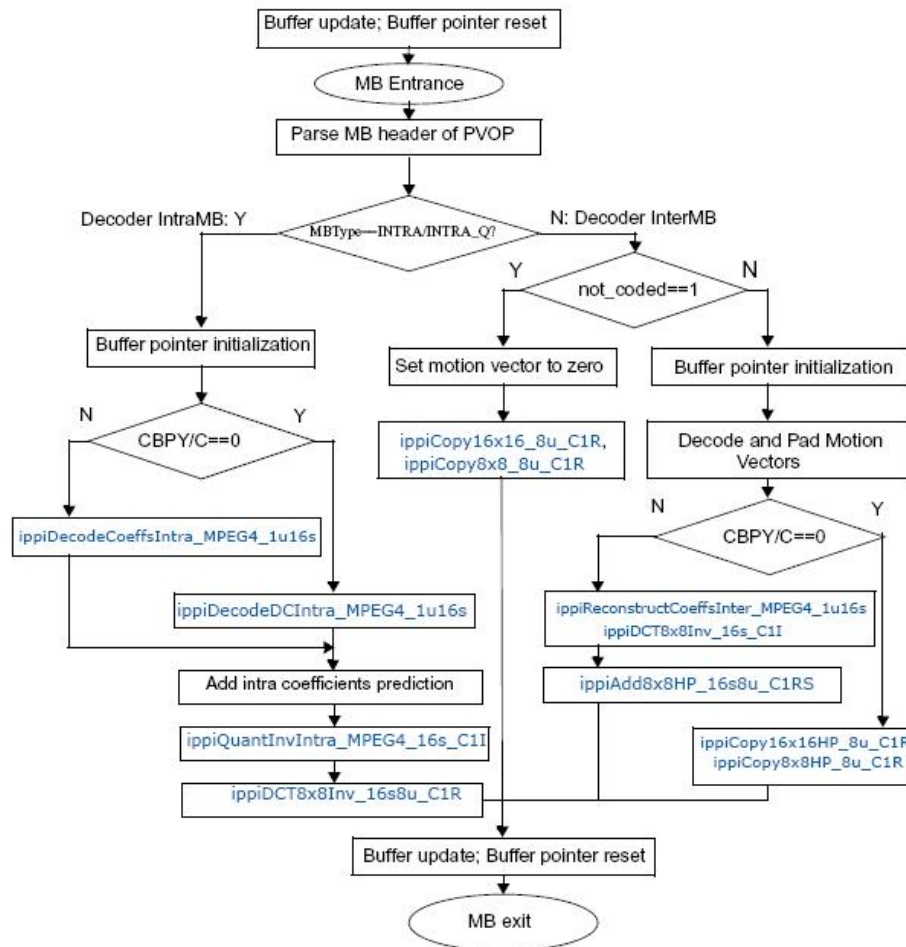
---

Below a high-level description of the MPEG-4 functions usage is given, followed by Intel IPP macro/data structures definition and the detailed descriptions of individual functions.

## High Level Description

"Decoding an MB in P-VOP" shows the general steps that are needed to decode a MacroBlock (MB) in Predictive coded Video Object Plane (P-VOP) and the associated Intel IPP functions.

### Decoding an MB in P-VOP





## Data Types and Structures

### Rectangle Plane

The following structure is defined in Intel IPP to represent a rectangle:

```
typedef structure _IppiRect {  
    int  x;  
    int  y;  
    int  width;  
    int  height;  
  
}
```

### Block Type

The following enumeration indicates the block type:

```
enum {  
    IPPVC_BLOCK_LUMA,  
    IPPVC_BLOCK_CHROMA  
};
```

### Sprite Type

The following enumeration indicates the sprite type:

```
enum {  
    IPPVC_SPRITE_STATIC = 1,  
    IPPVC_SPRITE_GMC = 2  
};
```

## Motion Compensation

### Copy8x8QP\_MPEG4, Copy16x8QP\_MPEG4, Copy16x16QP\_MPEG4

Copy fixed size blocks with quarter-pixel accuracy.

#### Syntax

```
IppStatus ippiCopy8x8QP_MPEG4_8u_C1R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst, int  
dstStep, int acc, int rounding);
```

```
IppStatus ippiCopy16x8QP_MPEG4_8u_C1R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst, int  
dstStep, int acc, int rounding);
```

```
IppStatus ippiCopy16x16QP_MPEG4_8u_C1R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst, int dstStep, int acc, int rounding);
```

## Parameters

<i>pSrc</i>	Pointer to the source block.
<i>srcStep</i>	Distance in bytes between starts of the consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination block.
<i>dstStep</i>	Distance in bytes between starts of the consecutive lines in the destination image.
<i>acc</i>	Parameter that determines quarter-pixel accuracy.
<i>rounding</i>	Parameter that determines type of rounding for pixel approximation; may be 0 or 1.

## Description

The functions *ippiCopy8x8QP\_MPEG4\_8u\_C1R*, *ippiCopy16x8QP\_MPEG4\_8u\_C1R*, and *ippiCopy16x16QP\_MPEG4\_8u\_C1R* are declared in the *ippvc.h* file. These functions copy the source block to the destination block with half-pixel accuracy. Parameter *rounding* has the same meaning as *vop\_rounding\_type* in [ISO14496]. Parameter *acc* is a four-bit value, the bits 0-1 define quarter-pixel offset in horizontal direction and bits 2-3 define quarter-pixel offset in vertical direction. The process of quarter-pixel interpolation is described in [ISO14496] subclause 7.6.2.2.

These functions are used in the MPEG-4 encoder and decoder included into Intel IPP Samples. See [introduction](#) to the MPEG-4 section.

## Return Values

<i>ippStsNoErr</i>	Indicates no error.
<i>ippStsNullPtrErr</i>	Indicates an error when at least one input pointer is NULL.

## OBMCM8x8HP\_MPEG4, OBMCM16x16HP\_MPEG4, OBMCM8x8QP\_MPEG4

*Perform overlapped block motion compensation (OBMC)  
for one luminance block.*

---

## Syntax

```
IppStatus ippiOBMC8x8HP_MPEG4_8u_C1R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst, int dstStep, const IppMotionVector* pMVCur, const IppMotionVector* pMVLeft, const IppMotionVector* pMVRight, const IppMotionVector* pMVAbove, const IppMotionVector* pMVBelow, int rounding);
```

```
IppStatus ippiOBMC16x16HP_MPEG4_8u_C1R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst, int dstStep, const IppMotionVector* pMVCur, const IppMotionVector* pMVLeft, const IppMotionVector* pMVRight, const IppMotionVector* pMVAbove, const IppMotionVector* pMVBelow, int rounding);
```

```
IppStatus ippiOBMC8x8QP_MPEG4_8u_C1R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst, int
dstStep, const IppMotionVector* pMVCur, const IppMotionVector* pMVLeft, const
IppMotionVector* pMVRight, const IppMotionVector* pMVAbove, const IppMotionVector* pMVBelow,
int rounding);
```

## Parameters

<i>pSrc</i>	Pointer to the source blockcoallocated with the destination block.
<i>srcStep</i>	Width in bytes of the source plane.
<i>pDst</i>	Pointer to the destination block.
<i>dstStep</i>	Width in bytes of the destination plane.
<i>pMVCur</i>	Pointer to the motion vector for the current block.
<i>pMVLeft</i>	Pointer to the motion vector for the left block.
<i>pMVRight</i>	Pointer to the motion vector for the right block.
<i>pMVAbove</i>	Pointer to the motion vector for the above block.
<i>pMVBelow</i>	Pointer to the motion vector for the below block.
<i>rounding</i>	Parameter that determines type of rounding for half pixel approximation; may be 0 or 1.

## Description

The functions `ippiOBMC8x8HP_MPEG4_8u_C1R`, `ippiOBMC16x16HP_MPEG4_8u_C1R`, and `ippiOBMC8x8QP_MPEG4_8u_C1R` are declared in the `ippvc.h` file. The functions `ippiOBMC8x8HP_MPEG4_8u_C1R` and `ippiOBMC16x16HP_MPEG4_8u_C1R` perform the overlapped block motion compensation of the *pSrc* with the half-pixel accuracy ([ISO14496], subclause 7.6.2.1) and `ippiOBMC8x8QP_MPEG4_8u_C1R` - with quarter-pixel accuracy ([ISO14496], subclause 7.6.2.2) and store the results in the destination block *pDst*. Overlapped motion compensation is specified in [ISO14496], subclause 7.6.6. `ippiOBMC16x16HP_MPEG4_8u_C1R` should be used for overlapped motion compensation in Reduced Resolution VOPs as specified in [ISO14496], subclause 7.6.10.

The function `ippiOBMC8x8HP_MPEG4_8u_C1R` is used in the H.263 and MPEG-4 encoders and decoders included into Intel IPP Samples. The function `ippiOBMC8x8QP_MPEG4_8u_C1R` is used in the MPEG-4 encoder and decoder included into Intel IPP Samples. See [introduction](#) to MPEG-4 section.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointers is NULL.

## Sprite and Global Motion Compensation

Functions for sprite and Global Motion Compensation (GMC) are used in decoding processes of static sprites and in decoding/encoding processes of Video Object Layers with GMC.

## WarpInit\_MPEG4

*Initializes IppiWarpSpec\_MPEG4 structure for further usage in GMC or Sprite reconstruction.*

---

### Syntax

```
IppStatus ippiWarpInit_MPEG4(IppiWarpSpec_MPEG4* pSpec, const int* pDU, const int* pDV,
int numWarpingPoints, int spriteType, int warpingAccuracy, int roundingType, int
quarterSample, int fcode, const IppiRect* pSpriteRect, const IppiRect* pVopRect);
```

### Parameters

<i>pSpec</i>	Pointer to IppiWarpSpec_MPEG4 structure.
<i>pDU</i>	Pointer to an array of x-coordinate of warping points.
<i>pDV</i>	Pointer to an array of y-coordinate of warping points.
<i>numWarpingPoints</i>	Number of warping points, valid in [0-4].
<i>spriteType</i>	Indicates a sprite coding mode, takes values IPPVC_SPRITE_STATIC, indicating static sprites IPPVC_SPRITE_GMC, indicating GMC.
<i>warpingAccuracy</i>	Accuracy of warping, valid in [0-3].
<i>roundingType</i>	Parameter that determines type of rounding for pixel approximation; may be 0 or 1.
<i>quarterSample</i>	Parameter that indicates a quarter sample mode; may be 0 or 1.
<i>fcode</i>	Parameter that determines the range of motion vector, valid in the range [1,7].
<i>pSpriteRect</i>	Parameter that determines rectangle region for Sprite (or reference VOP for GMC).
<i>pVopRect</i>	Parameter that determines rectangle region for current VOP.

### Description

The function `ippiWarpInit_MPEG4` is declared in the `ippvc.h` file. This function initializes the `IppiWarpSpec_MPEG4` structure for further usage in GMC or Sprite reconstruction. The meaning of parameters is described in [ISO14496], subclause 7.8.

This function is used in the MPEG-4 encoder and decoder included into Intel IPP Samples. See [introduction](#) to the MPEG-4 section.

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when at least one input pointer is NULL.
<code>ippStsSizeErr</code>	Indicates no error when width or height of images is less than or equal to zero.
<code>ippStsOutOfRangeErr</code>	Indicates an error when <i>numWarpingPoints</i> , <i>warpingAccuracy</i> , or <i>fcode</i> are out of the valid range.

## WarpGetSize\_MPEG4

*Returns size of IppiWarpSpec\_MPEG4 structure.*

---

### Syntax

```
IppStatus ippiWarpGetSize_MPEG4(int* pSpecSize);
```

### Parameters

*pSpecSize*                      Pointer to the resulting size of the structure IppiWarpSpec\_MPEG4.

### Description

The function `ippiWarpGetSize_MPEG4` is declared in the `ippvc.h` file. This function returns a size of structure `IppiWarpSpec_MPEG4`.

This function is used in the MPEG-4 encoder and decoder included into Intel IPP Samples. See [introduction](#) to the MPEG-4 section.

### Return Values

`ippStsNoErr`                      Indicates no error.  
`ippStsNullPtrErr`                Indicates an error when pointer *pSpecSize* is NULL.

## WarpLuma\_MPEG4

*Warps arbitrary rectangular luminance region.*

---

### Syntax

```
IppStatus ippiWarpLuma_MPEG4_8u_C1R(const Ipp8u* pSrcY, int srcStepY, Ipp8u* pDstY, int dstStepY, const IppiRect* pDstRect, const IppiWarpSpec_MPEG4* pSpec);
```

### Parameters

*pSrcY*                              Pointer to the origin of the source plane.  
*srcStep*                            Distance in bytes between starts of the consecutive lines in the destination plane.  
*pDst*                                Pointer to the destination region.  
*dstStep*                            Distance in bytes between starts of the consecutive lines in the destination plane.  
*pDstRect*                          Rectangular destination region.  
*pSpec*                              Pointer to the structure with motion parameters.

### Description

The function `ippiWarpLuma_MPEG4_8u_C1R` is declared in the `ippvc.h` file. This function warps an arbitrary rectangular area using motion information from `IppiWarpSpec_MPEG4` structure. This function may be used in GMC or Sprite reconstruction ([[ISO14496](#)], subclause 7.8).

This function is used in the MPEG-4 encoder and decoder included into Intel IPP Samples. See [introduction](#) to the MPEG-4 section.

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when at least one input pointer is <code>NULL</code> .

## WarpChroma\_MPEG4

*Warp arbitrary rectangular chrominance region.*

---

### Syntax

```
ippStatus ippkWarpChroma_MPEG4_8u_P2R(const Ipp8u* pSrcCb, int srcStepCb, const Ipp8u*
pSrcCr, int srcStepCr, Ipp8u* pDstCb, int dstStepCb, Ipp8u* pDstCr, int dstStepCr, const
IppiRect* pDstRect, const IppiWarpSpec_MPEG4* pSpec);
```

### Parameters

<code>pSrcCb</code>	Pointer to the origin of the first source plane.
<code>srcStepCb</code>	Distance in bytes between starts of the consecutive lines in the first source plane.
<code>pSrcCr</code>	Pointer to the origin of the second source plane.
<code>srcStepCr</code>	Distance in bytes between starts of the consecutive lines in the second source plane.
<code>pDstCb</code>	Pointer to the first destination plane.
<code>dstStepCb</code>	Distance in bytes between starts of the consecutive lines in the first destination plane.
<code>pDstCr</code>	Pointer to the second destination plane.
<code>dstStepCr</code>	Distance in bytes between starts of the consecutive lines in the second destination plane.
<code>pDstRect</code>	Rectangular destination region.
<code>pSpec</code>	Pointer to the structure with motion parameters.

### Description

The function `ippkWarpChroma_MPEG4_8u_P2R` is declared in the `ippvc.h` file. This function warps an arbitrary rectangular chrominance area using motion information from `IppiWarpSpec_MPEG4` structure. This function may be used in GMC or Sprite reconstruction ([[ISO14496](#)], subclause 7.8).

This function is used in the MPEG-4 encoder and decoder included into Intel IPP Samples. See [introduction](#) to the MPEG-4 section.

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when at least one input pointer is <code>NULL</code> .

## CalcGlobalMV\_MPEG4

*Calculates Global Motion Vector for one macroblock.*

### Syntax

```
IppStatus ippCalcGlobalMV_MPEG4(int xOffset, int yOffset, IppMotionVector* pGMV, const
IppiWarpSpec_MPEG4* pSpec);
```

### Parameters

<i>xOffset</i>	The left coordinate of top-left corner of luma 16x16 block.
<i>yOffset</i>	The top coordinate of top-left corner of luma 16x16 block.
<i>pGMV</i>	Pointer to the resulting motion vector.
<i>pSpec</i>	Pointer to the structure with motion parameters.

### Description

The function `ippCalcGlobalMV_MPEG4` is declared in the `ippvc.h` file. This function calculates a global motion vector for a macroblock and clips it to lie in range  $[-2^{\text{fcode}+4}, 2^{\text{fcode}+4}-1]$  as it described in [ISO14496], subclause 7.8.7.3.

This function is used in the MPEG-4 encoder and decoder included into Intel IPP Samples. See [introduction](#) to the MPEG-4 section.

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when at least one input pointer is NULL.

## ChangeSpriteBrightness\_MPEG4

*Change brightness after sprite warping.*

### Syntax

```
IppStatus ippChangeSpriteBrightness_MPEG4_8u_C1IR(const Ipp8u* pSrcDst, int srcDstStep,
int width, int height, int brightnessChangeFactor);
```

### Parameters

<i>pSrcDst</i>	Pointer to the video plane.
<i>srcDstStep</i>	Distance in bytes between starts of the consecutive lines in the video plane.
<i>width</i>	The width of the video plane.
<i>height</i>	The height of the video plane.
<i>brightnessChangeFactor</i>	Factor for changing brightness; valid in the range [-112; 1648].

## Description

The function `ippiChangeSpriteBrightness_MPEG4_8u_C1IR` is declared in the `ippvc.h` file. This function changes brightness of video plane after sprite warping ([ISO14496], subclause 7.8).

This function is used in the MPEG-4 decoder included into Intel IPP Samples. See [introduction](#) to the MPEG-4 section.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when at least one input pointer is <code>NULL</code> .
<code>ippStsOutOfRangeErr</code>	Indicates an error when <i>brightnessChangeFactor</i> is out of the valid range.

## Inverse Quantization

### QuantInvIntraInit\_MPEG4, QuantInvInterInit\_MPEG4

*Initialize specification structures.*

---

## Syntax

```

IppStatus ippiQuantInvIntraInit_MPEG4(const Ipp8u* pQuantMatrix,
IppiQuantInvIntraSpec_MPEG4* pSpec, int bitsPerPixel);

IppStatus ippiQuantInvInterInit_MPEG4(const Ipp8u* pQuantMatrix,
IppiQuantInvInterSpec_MPEG4* pSpec, int bitsPerPixel);

```

## Parameters

<i>pQuantMatrix</i>	Pointer to quantization matrix size of 64.
<i>pSpec</i>	Pointer to the initialized specification structure <code>IppiQuantInvIntraSpec_MPEG4</code> or <code>IppiQuantInvInterSpec_MPEG4</code> .
<i>bitsPerPixel</i>	Video data precision used for saturation of the result. Valid within the range [4, 12].

## Description

The functions `ippiQuantInvIntraInit_MPEG4` and `ippiQuantInvInterInit_MPEG4` are declared in the `ippvc.h` file. These functions initialize specification structure `IppiQuantInvIntraSpec_MPEG4` or `IppiQuantInvInterSpec_MPEG4`. If *pQuantMatrix* is `NULL`, the second quantization method is used; otherwise, the first method is used.

These functions are used in the MPEG-4 encoder and decoder included into Intel IPP Samples. See [introduction](#) to the MPEG-4 section.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error pointer <i>pSpec</i> is <code>NULL</code> .



`ippStsOutOfRangeErr` Indicates an error condition when *bitsPerPixel* is out of the range [4, 12].

## QuantInvIntraGetSize\_MPEG4, QuantInvInterGetSize\_MPEG4

*Return size of specification structures.*

### Syntax

```
IppStatus ippiQuantInvIntraGetSize_MPEG4(int* pSpecSize);
IppStatus ippiQuantInvInterGetSize_MPEG4(int* pSpecSize);
```

### Parameters

*pSpecSize* Pointer to the resulting size of the initialized specification structure `IppiQuantInvIntraSpec_MPEG4` or `IppiQuantInvInterSpec_MPEG4`.

### Description

The functions `ippiQuantInvIntraGetSize_MPEG4` and `ippiQuantInvInterGetSize_MPEG4` are declared in the `ippvc.h` file. These functions return a size of specification structure `IppiQuantInvIntraSpec_MPEG4` or `IppiQuantInvInterSpec_MPEG4`.

These functions are used in the MPEG-4 encoder and decoder included into Intel IPP Samples. See [introduction](#) to the MPEG-4 section.

### Return Values

`ippStsNoErr` Indicates no error.  
`ippStsNullPtrErr` Indicates an error pointer *pSpecSize* is NULL.

## QuantInvIntra\_MPEG4, QuantInvInter\_MPEG4

*Perform inverse quantization on intra/inter coded block.*

### Syntax

```
IppStatus ippiQuantInvIntra_MPEG4_16s_C1I(Ipp16s* pCoeffs, int indxLastNonZero, const
IppiQuantIntraSpec_MPEG4* pSpec, int QP, int blockType);
IppStatus ippiQuantInvInter_MPEG4_16s_C1I(Ipp16s* pCoeffs, int indxLastNonZero, const
IppiQuantInterSpec_MPEG4* pSpec, int QP);
```

### Parameters

*pCoeffs* Pointer to the quantized DCT coefficients.  
*indxLastNonZero* Index of the last non-zero coefficient. This parameter provides faster operation. If the value is unknown, set to 63.

<i>pSpec</i>	Pointer to the initialized specification structure IppiQuantInvIntraSpec_MPEG4 or IppiQuantInvInterSpec_MPEG4 initialized by ippiQuantInvIntraInit_MPEG4 or ippiQuantInvInterInit_MPEG4 respectively.
<i>QP</i>	Quantization parameter.
<i>blockType</i>	Indicates the block type. Takes one of the following values: IPPVC_BLOCK_LUMA - for luma and alpha blocks, IPPVC_BLOCK_CHROMA - for chroma blocks.
<i>pSrcDst</i>	Pointer to the coefficient buffer of intra/inter block. Contains quantized coefficients (QF) on input and dequantized coefficients (F) on output.
<i>pQPMatrix</i>	Pointer to the quantization weighting matrix. If <i>pQPMatrix</i> is NULL, this function will use the second inverse quantization method; otherwise, it will use the first method.
<i>videoComp</i>	(Intra version only) Video component type of the current block. It is used for DC coefficient dequantizing. Takes one of the following flags: IPP_VIDEO_LUMINANCE, IPP_VIDEO_CHROMINANCE, IPP_VIDEO_ALPHA.

## Description

The functions `ippiQuantInvIntra_MPEG4_16s_C1I` and `ippiQuantInvInter_MPEG4_16s_C1I` are declared in the `ippvc.h` file. These functions perform inverse quantization on intra/inter coded block, saturation and mismatch control (for the first inverse quantization method only) as it is specified in [ISO14496], subclause 7.4.4.

For `ippiQuantInvIntra_MPEG4_16s_C1I` and `ippiQuantInvInter_MPEG4_16s_C1I` output coefficients are saturated to lie in range  $[-2^{\text{bitsPerPixel}+3}, 2^{\text{bitsPerPixel}+3} - 1]$ .

These functions are used in the MPEG-4 encoder and decoder included into Intel IPP Samples. See [introduction](#) to the MPEG-4 section.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if at least one pointer is NULL.
<code>ippStsQPErr</code>	Indicates an error condition if <i>QP</i> is out of the range $[1, 2^{\text{bitsPerPixel}-3} - 1]$ .

## VLC Decoding

### DecodeDCIntra\_MPEG4

*Decodes one DC coefficient for intra coded block.*

---

## Syntax

```
ippStatus ippiDecodeDCIntra_MPEG4_16s_C1I(Ipp8u** ppBitStream, int* pBitOffset, Ipp16s* pDC,
int blockType);
```

## Parameters

<i>ppBitStream</i>	Pointer to the pointer to the current byte in the bitstream buffer. The pointer is updated by the function.
<i>pBitOffset</i>	Pointer to the bit position in the byte pointed by <i>**ppBitStream</i> . The pointer is updated by the function.
<i>pDC</i>	Pointer to the output coefficient.
<i>blockType</i>	Indicates the block type, takes one of the following values: IPPVC_BLOCK_LUMA - for luma and alpha blocks, IPPVC_BLOCK_CHROMA - for chroma blocks.

## Description

The function `ippiDecodeDCIntra_MPEG4_1u16s` is declared in the `ippvc.h` header file. This function performs VLC decoding of the DC coefficient only for one intra coded block using VLC decoding process as specified in [ISO14496], subclause 7.4.1.1.

This function is used in the MPEG-4 decoder included into Intel IPP Samples. See [introduction](#) to the MPEG-4 section.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when at least one input pointer is <code>NULL</code> .
<code>ippStsBitOffsetErr</code>	Indicates an error condition if <i>*pBitOffset</i> is out of the range [0, 7].
<code>ippStsVLCCodeErr</code>	Indicates an error condition if an illegal code is detected through the DC decoding.

## DecodeCoeffsIntra\_MPEG4

*Decodes DCT coefficients for intra coded block.*

## Syntax

```
ippStatus ippiDecodeCoeffsIntra_MPEG4_1u16s(Ipp8u** ppBitStream, int* pBitOffset, Ipp16s* pCoeffs, int* pIndxLastNonZero, int rvlcFlag, int noDCFlag, int scan);
```

## Parameters

<i>ppBitStream</i>	Pointer to the pointer to the current byte in the bitstream buffer. The pointer is updated by the function.
<i>pBitOffset</i>	Pointer to the bit position in the byte pointed by <i>**ppBitStream</i> . The pointer is updated by the function.
<i>pCoeffs</i>	Pointer to the output coefficients.
<i>pIndxLastNonZero</i>	Pointer to the index of the last non-zero coefficient. In case of error during decoding, the index of the previous successfully decoded coefficient is stored in it.

<i>rvlcFlag</i>	Flag, when set to '0' indicates that VLC tables B.16, B.18, B.19, and B.21 [ISO14496] are used in decoding DCT coefficients, otherwise the reversible variable length tables B.23, B.24, and B.25 [ISO14496] are used.
<i>noDCFlag</i>	Flag, when set to '0' indicates that <i>pCoeffs</i> is set starting with zero element, otherwise - with the first element.
<i>scan</i>	Type of the scan, takes one of the following values: IPPVC_SCAN_NONE,      indicating that no inverse scan is performed, IPPVC_SCAN_ZIGZAG,    indicating the classical zigzag scan, IPPVC_SCAN_HORIZONTAL, indicating the alternate-horizontal scan, IPPVC_SCAN_VERTICAL, indicating the alternate-vertical scan, See the corresponding <a href="#">enumerator</a> in the introduction to the General Functions section.

## Description

The function `ippiDecodeCoeffsIntra_MPEG4_1u16s` is declared in the `ippvc.h` header file. This function performs VLC decoding of the DC coefficient (if *noDCFlag* is 0) and AC coefficients for one intra coded block using VLC decoding process as specified in [ISO14496], subclause 7.4.1. Additionally, the function can perform inverse scan.

This function is used in the MPEG-4 decoder included into Intel IPP Samples. See [introduction](#) to the MPEG-4 section.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when at least one input pointer is NULL.
<code>ippStsBitOffsetErr</code>	Indicates an error condition if * <i>pBitOffset</i> is out of the range [0, 7].
<code>ippStsVLCCodeErr</code>	Indicates an error condition if an illegal code is detected through the decoding.

## DecodeCoeffsIntraRVLCBack\_MPEG4

*Decodes DCT coefficients in backward direction for intra coded block using RVLC.*

---

## Syntax

```
ippStatus ippiDecodeCoeffsIntraRVLCBack_MPEG4_1u16s(Ipp8u** ppBitStream, int* pBitOffset,
Ipp16s* pCoeffs, int* pIdxLastNonZero, int noDCFlag);
```

## Parameters

<i>ppBitStream</i>	Pointer to the pointer to the current byte in the bitstream buffer. The pointer is updated by the function.
<i>pBitOffset</i>	Pointer to the bit position in the byte pointed by <i>**ppBitStream</i> . The pointer is updated by the function.

<i>pCoeffs</i>	Pointer to the output coefficients.
<i>pIndxLastNonZero</i>	Pointer to the index of the last non-zero coefficient. In case of error during decoding, the index of the previous successfully decoded coefficient is stored in it.
<i>noDCFlag</i>	Flag, when set to '0' indicates that <i>pCoeffs</i> is set starting with zero element, otherwise - with the first element.

## Description

The function `ippiDecodeCoeffsIntraRVLCBack_MPEG4_1u16s` is declared in the `ippvc.h` header file. This function performs backward decoding of the DC coefficient (if *noDCFlag* is 0) and AC coefficients using variable length tables B.23, B.24, and B.25 [ISO14496] for one intra coded block. The backward RVLC decoding process is specified in [ISO14496], subclauses E.1.3 and E.1.4.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when at least one input pointer is NULL.
<code>ippStsBitOffsetErr</code>	Indicates an error condition if <i>*pBitOffset</i> is out of the range [0, 7].
<code>ippStsVLCCodeErr</code>	Indicates an error condition if an illegal code is detected through the decoding.

## DecodeCoeffsInter\_MPEG4

*Decodes DCT coefficients for inter coded block.*

## Syntax

```
ippStatus ippiDecodeCoeffsInter_MPEG4_1u16s(Ipp8u** ppBitStream, int* pBitOffset, Ipp16s* pCoeffs, int* pIndxLastNonZero, int rvlcFlag, int scan);
```

## Parameters

<i>ppBitStream</i>	Pointer to the pointer to the current byte in the bitstream buffer. The pointer is updated by the function.
<i>pBitOffset</i>	Pointer to the bit position in the byte pointed by <i>**ppBitStream</i> . The pointer is updated by the function.
<i>pCoeffs</i>	Pointer to the output coefficients.
<i>pIndxLastNonZero</i>	Pointer to the index of the last non-zero coefficient. In case of error during decoding, the index of the previous successfully decoded coefficient is stored in it.
<i>rvlcFlag</i>	Flag, when set to '0' indicates that VLC tables B.17, B.18, B.19, and B.21 [ISO14496] are used in decoding DCT coefficients, otherwise the reversible variable length tables B.23, B.24, and B.25 [ISO14496] are used.
<i>scan</i>	Type of the scan, takes one of the following values: <code>IPPVC_SCAN_NONE</code> ,     indicating that no inverse scan is performed,

IPPVC\_SCAN\_ZIGZAG, indicating the classical zigzag scan,

IPPVC\_SCAN\_VERTICAL, indicating the alternate-vertical scan.

See the corresponding [enumerator](#) in the introduction to the General Functions section

## Description

The function `ippiDecodeCoefInter_MPEG4_1u16s` is declared in the `ippvc.h` header file. This function performs VLC decoding of the DCT coefficients for one inter coded block. Inter DCT VLC decoding process is specified in [ISO14496], subclause 7.4.1. Additionally, the function can perform inverse scan.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when at least one input pointer is <code>NULL</code> .
<code>ippStsBitOffsetErr</code>	Indicates an error condition if <code>*pBitOffset</code> is out of the range <code>[0, 7]</code> .
<code>ippStsVLCCodeErr</code>	Indicates an error condition if an illegal code is detected through the DCT decoding.

## DecodeCoeffsInterRVLCBack\_MPEG4

*Decodes DCT coefficients in backward direction for inter coded block using RVLC.*

---

## Syntax

```
ippStatus ippiDecodeCoeffsIntraRVLCBack_MPEG4_1u16s(Ipp8u** ppBitStream, int* pBitOffset,
Ipp16s* pCoeffs, int* pIndxLastNonZero);
```

## Parameters

<code>ppBitStream</code>	Pointer to the pointer to the current byte in the bitstream buffer. The pointer is updated by the function.
<code>pBitOffset</code>	Pointer to the bit position in the byte pointed by <code>**ppBitStream</code> . The pointer is updated by the function.
<code>pCoeffs</code>	Pointer to the output coefficients.
<code>pIndxLastNonZero</code>	Pointer to the index of the last non-zero coefficient. In case of error during decoding, the index of the previous successfully decoded coefficient is stored in it.

## Description

The function `ippiDecodeCoeffsInterRVLCBack_MPEG4_1u16s` is declared in the `ippvc.h` header file. This function performs backward decoding of the DC coefficient coefficients using variable length tables B.23, B.24 and B.25 [ISO14496] for one inter coded block. The backward RVLC decoding process is specified in [ISO14496], subclauses E.1.3 and E.1.4.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when at least one input pointer is <code>NULL</code> .
<code>ippStsBitOffsetErr</code>	Indicates an error condition if <code>*pBitOffset</code> is out of the range <code>[0, 7]</code> .
<code>ippStsVLCCodeErr</code>	Indicates an error condition if an illegal code is detected through the decoding.

## ReconstructCoeffsInter\_MPEG4

*Decodes DCT coefficients, performs inverse scan and inverse quantization for inter coded block.*

### Syntax

```
IppStatus ippiReconstructCoeffsInter_MPEG4_lul6s(Ipp8u** ppBitStream, int* pBitOffset,
Ipp16s* pCoeffs, int* pIndxLastNonZero, int rvlcFlag, int scan, const
IppiQuantInvInterSpec_MPEG4* pQuantInvInterSpec, int QP);
```

### Parameters

<code>ppBitStream</code>	Pointer to the pointer to the current byte in the bitstream buffer. The pointer is updated by the function.
<code>pBitOffset</code>	Pointer to the bit position in the byte pointed by <code>**ppBitStream</code> . The pointer is updated by the function.
<code>pCoeffs</code>	Pointer to the output coefficients.
<code>pIndxLastNonZero</code>	Pointer to the index of last non-zero coefficient. In case of error during decoding, the index of the previous successfully decoded coefficient is stored in this parameter.
<code>rvlcFlag</code>	Flag, which when set to '0' indicates that VLC tables B.17, B.18, B.20 and B.22 [ISO14496] are used when decoding DCT coefficients, otherwise the reversible variable length tables B.23, B.24 and B.25 [ISO14496] are used.
<code>scan</code>	Type of the inverse scan, takes one of the following values: <code>IPPVC_SCAN_ZIGZAG</code> , indicating the classical zigzag scan, <code>IPPVC_SCAN_VERTICAL</code> , indicating the alternate-vertical scan. See the corresponding <a href="#">enumerator</a> in the introduction to the General Functions section.
<code>pQuantInvInterSpec</code>	Pointer to the <code>IppiQuantInvInterSpec_MPEG4</code> that was initialized by <code>ippiQuantInvInterInit_MPEG4</code> .
<code>QP</code>	Quantization parameter

### Description

The function `ippiReconstructCoeffsInter_MPEG4_lul6s` is declared in the `ippvc.h` header file. This function performs VLC decoding, inverse scan and inverse quantization of the DCT coefficients for one inter coded block. Inter DCT VLC decoding process is specified in [ISO14496], subclause 7.4.1. Inverse scan process is specified

in [ISO14496], subclause 7.4.2. Inverse quantization process is specified in [ISO14496], subclause 7.4.4. This function supports video data precision 4-12 bits per pixel, that is, after inverse quantization the coefficients are saturated to lie in the range:  $[-2^{\text{bitsPerPixel}+3}, 2^{\text{bitsPerPixel}+3} - 1]$ .

This function is used in the MPEG-4 decoder included into Intel IPP Samples. See [introduction](#) to the MPEG-4 section.

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when at least one input pointer (except <i>pQuantMatrix</i> ) is NULL.
<code>ippStsBitOffsetErr</code>	Indicates an error condition if <i>pBitOffset</i> is out of the range [0; 7].
<code>ippStsVLCCodeErr</code>	Indicates an error condition if an illegal code is detected through the DCT decoding.
<code>ippStsQPErr</code>	Indicates an error condition if QP is out of the range $[1; 2^{\text{bitsPerPixel}-3} - 1]$ .

### Postprocessing

## FilterDeblocking8x8HorEdge\_MPEG4, FilterDeblocking8x8VerEdge\_MPEG4

*Perform deblocking filtering on a horizontal or vertical edge of two adjacent blocks.*

---

### Syntax

```
ippStatus ippiFilterDeblocking8x8HorEdge_MPEG4_8u_C1IR(Ipp8u* pSrcDst, int step, int QP,
int THR1, int THR2);
```

```
ippStatus ippiFilterDeblocking8x8VerEdge_MPEG4_8u_C1IR(Ipp8u* pSrcDst, int step, int QP,
int THR1, int THR2);
```

### Parameters

<i>pSrcDst</i>	Pointer to the first pixel of lower (HorEdge) or right (VerEdge) block.
<i>step</i>	Width in bytes of the source plane.
<i>QP</i>	Quantization parameter.
<i>THR1</i> , <i>THR2</i>	Threshold values that specify the filter mode ([ISO14496], Annex F.3.1).

### Description

The functions `ippiFilterDeblocking8x8HorEdge_MPEG4_8u_C1IR` and `ippiFilterDeblocking8x8VerEdge_MPEG4_8u_C1IR` are declared in the `ippvc.h` file. These functions perform deblocking filtering of two adjacent blocks on horizontal and vertical edges, respectively ([ISO14496], Annex F.3.1).

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if the pointer <i>pSrcDst</i> is NULL.



`ippStsMP4QPErr`

Indicates an error condition if  $QP$  is out of range [1, 31]

## FilterDeringingThreshold\_MPEG4

*Computes threshold values for the deringing filtering through a macroblock.*

---

### Syntax

```
IppStatus ippiFilterDeringingThreshold_MPEG4_8u_P3R(Ipp8u* pSrcY, int stepY, const Ipp8u* pSrcCb, int stepCb, const Ipp8u* pSrcCr, int stepCr, int threshold[6]);
```

### Parameters

<code>pSrcY</code>	Pointer to the left upper Y-block in the current macroblock.
<code>stepY</code>	Width in bytes of the luminance plane.
<code>pSrcCb</code>	Pointer to the Cb-block in the current macroblock.
<code>stepCb</code>	Width in bytes of the chrominance (Cb) plane.
<code>pSrcCr</code>	Pointer to the Cr-block in the current macroblock.
<code>stepCr</code>	Width in bytes of the chrominance (Cr) plane.
<code>threshold</code>	Array of the threshold values for all blocks.

### Description

The function `ippiFilterDeringingThreshold_MPEG4_8u` is declared in the `ippvc.h` file. This function performs threshold determination ([ISO14496], Annex F.3.2.1), which is the first subprocess of the deringing filtering. The obtained threshold values are required for the function `FilterDeringingSmooth8x8_MPEG4`.

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointers is <code>NULL</code> .

## FilterDeringingSmooth8x8\_MPEG4

*Performs deringing filtering of a block.*

---

### Syntax

```
IppStatus ippiFilterDeringingSmooth8x8_MPEG4_8u_C1R(Ipp8u* pSrc, int srcStep, Ipp8u* pDst, int dstStep, int QP, int threshold);
```

### Parameters

<code>pSrc</code>	Pointer to the source block.
<code>srcStep</code>	Width in bytes of the source plane.
<code>QP</code>	Quantization parameter ( <code>quantiser_scale</code> ).
<code>threshold</code>	Threshold value for the block.

<i>pDst</i>	Pointer to the destination block.
<i>dstStep</i>	Width in bytes of the destination plane.

### Description

The function `ippiFilterDeringingSmooth8x8_MPEG4_8u` is declared in the `ippvc.h` file. This function performs deringing filtering, specifically, index acquisition and adaptive smoothing ([ISO14496], Annex F.3.2.1, 3.2.2) of the source block *pSrc*, and stores the result in the destination block *pDst*. The threshold value *threshold* is returned by the auxiliary function `FilterDeringingThreshold_MPEG4` that should be called beforehand.

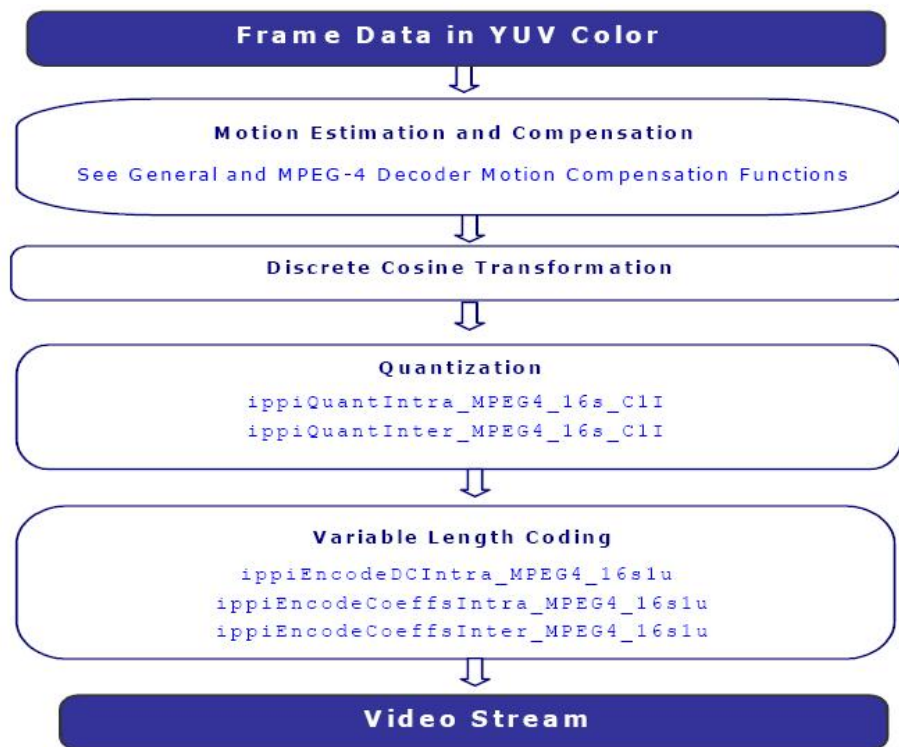
### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointers is <code>NULL</code> .
<code>ippStsMP4QPErr</code>	Indicates an error condition if <i>QP</i> is out of range [1, 31]

## MPEG-4 Video Encoder Functions

This section describes the main steps of MPEG-4 ([ISO14496]) video encoding in accordance with the pipeline shown in "MPEG-4 Encoding Pipeline".

### MPEG-4 Encoding Pipeline



MPEG-4 is a widely used coding method for video signals in various applications such as digital storage media, internet, various forms of wired or wireless communications, etc.

### Data Types and Structures

The basic data types and structures for MPEG-4 encoder are the same as described for the IPP MPEG-4 decoder. See the corresponding subsections in [MPEG-4 Decoder Data Types and Structures](#).

## Quantization

### QuantIntraInit\_MPEG4, QuantInterInit\_MPEG4

*Initialize specification structures.*

---

#### Syntax

```
IppStatus ippiQuantIntraInit_MPEG4(const Ipp8u* pQuantMatrix, IppiQuantIntraSpec_MPEG4* pSpec, int bitsPerPixel);
```

```
IppStatus ippiQuantInterInit_MPEG4(const Ipp8u* pQuantMatrix, IppiQuantInterSpec_MPEG4* pSpec, int bitsPerPixel);
```

#### Parameters

<i>pQuantMatrix</i>	Pointer to quantization matrix size of 64.
<i>pSpec</i>	Pointer to the initialized specification structure IppiQuantIntraSpec_MPEG4 or IppiQuantInterSpec_MPEG4.
<i>bitsPerPixel</i>	Video data precision used for saturation of the result. Valid within the range [4, 12].

#### Description

The functions `ippiQuantIntraInit_MPEG4` and `ippiQuantInterInit_MPEG4` are declared in the `ippvc.h` file. These functions initialize specification structure `IppiQuantIntraSpec_MPEG4` or `IppiQuantInterSpec_MPEG4`. If *pQuantMatrix* is NULL, the second quantization method is used; otherwise, the first method is used.

These functions are used in the MPEG-4 encoder included into Intel IPP Samples. See [introduction](#) to the MPEG-4 section.

#### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error pointer <i>pSpec</i> is NULL.
<code>ippStsOutOfRangeErr</code>	Indicates an error condition when <i>bitsPerPixel</i> is out of the range [4, 12].

### QuantIntraGetSize\_MPEG4, QuantInterGetSize\_MPEG4

*Return size of specification structures.*

---

#### Syntax

```
IppStatus ippiQuantIntraGetSize_MPEG4(int* pSpecSize);
```

```
IppStatus ippiQuantInterGetSize_MPEG4(int* pSpecSize);
```

## Parameters

*pSpecSize* Pointer to the resulting size of the initialized specification structure  
IppiQuantIntraSpec\_MPEG4 or IppiQuantInterSpec\_MPEG4.

## Description

The functions `ippiQuantIntraGetSize_MPEG4` and `ippiQuantInterGetSize_MPEG4` are declared in the `ippvc.h` file. These functions return a size of specification structure `IppiQuantIntraSpec_MPEG4` or `IppiQuantInterSpec_MPEG4`.

These functions are used in the MPEG-4 encoder included into Intel IPP Samples. See [introduction](#) to the MPEG-4 section.

## Return Values

`ippStsNoErr` Indicates no error.  
`ippStsNullPtrErr` Indicates an error pointer *pSpecSize* is NULL.

## QuantIntra\_MPEG4, QuantInter\_MPEG4

Perform inverse quantization on intra/inter coded block.

## Syntax

```
ippStatus ippiQuantIntra_MPEG4_16s_C1I(Ipp16s* pCoeffs, const IppiQuantIntraSpec_MPEG4*
pSpec, int QP, int* pCountNonZero, int blockType);

ippStatus ippiQuantIntra_MPEG4_16s_C1I(Ipp16s* pSrcDst, Ipp8u QP, int blockIndex, const
int* pQPMatrix);
```

## Parameters

*pCoeffs* Pointer to the quantized DCT coefficients.

*pSpec* Pointer to the initialized specification structure  
`IppiQuantIntraSpec_MPEG4` or `IppiQuantInterSpec_MPEG4`  
initialized by `ippiQuantIntraInit_MPEG4` or `ippiQuantInterInit_MPEG4` respectively.

*QP* Quantization parameter.

*pCountNonZero* Pointer to the count of non-zero coefficients.

*blockType* Indicates the block type. Takes one of the following values:  
`IPPVC_BLOCK_LUMA` - for luma and alpha blocks,  
`IPPVC_BLOCK_CHROMA` - for chroma blocks.

*pSrcDst* Pointer to the input intra block coefficients; as an output argument  
points to the quantized intra block coefficients.

*blockIndex* Block index indicating the component type and position as defined in  
subclause 6.1.3.8 of [ISO14496A]. Furthermore, indexes 6 to 9 indicate  
the alpha blocks spatially corresponding to luminance blocks 0 to 3 in  
the same macroblock.

*pQPMatrix*

The pointer, which is NULL, if the second inverse quantization method is used. If the first inverse quantization method is used, it points to the quantization weighting coefficient's buffer (for inter MB) whose first 64 elements are the quantization weighting matrix in Q0. The second 64 elements are their reciprocals in Q21.

## Description

The functions `ippiQuantIntra_MPEG4_16s_C1I` and `ippiQuantInter_MPEG4_16s_C1I` are declared in the `ippvc.h` file. These functions perform quantization on intra/inter coded block.

For `ippiQuantIntra_MPEG4_16s_C1I` and `ippiQuantInter_MPEG4_16s_C1I` output coefficients are saturated to lie in range `[-2047; 2047]`. Also these functions calculate the number of non-zero coefficients after quantization.

These functions are used in the MPEG-4 encoder included into Intel IPP Samples. See [introduction](#) to the MPEG-4 section.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if at least one pointer is NULL.
<code>ippStsQPErr</code>	Indicates an error condition if <i>QP</i> is less than or equal to 0.
<code>ippStsBadArgErr</code>	Indicates bad arguments.

## VLC Encoding

### EncodeDCIntra\_MPEG4

*Encodes one DC coefficient for intra coded block.*

---

## Syntax

```
IppStatus ippiEncodeDCIntra_MPEG4_16s1u(Ipp16s DC, Ipp8u** ppBitStream, int* pBitOffset,
int blockType);
```

## Parameters

<i>DC</i>	DC coefficient to be encoded.
<i>ppBitStream</i>	Pointer to the pointer to the current byte in the bitstream buffer. The pointer is updated by the function.
<i>pBitOffset</i>	Pointer to the bit position in the byte pointed by <i>**ppBitStream</i> . The pointer is updated by the function.
<i>blockType</i>	Indicates the block type, takes one of the following values: <code>IPPVC_BLOCK_LUMA</code> - for luma and alpha blocks, <code>IPPVC_BLOCK_CHROMA</code> - for chroma blocks.

## Description

The function `ippiEncodeDCIntra_MPEG4_16slu` is declared in the `ippvc.h` header file. This function performs VLC encoding of the DC coefficient only for one intra coded block using tables B.13, B.14, B.15 specified in [ISO14496].

This function is used in the MPEG-4 encoder included into Intel IPP Samples. See [introduction](#) to the MPEG-4 section.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when at least one input pointer is NULL.
<code>ippStsBitOffsetErr</code>	Indicates an error condition if <code>*pBitOffset</code> is out of the range [0, 7].

## EncodeCoeffsIntra\_MPEG4

*Encodes DCT coefficients for intra coded block.*

## Syntax

```
ippStatus ippiEncodeCoeffsIntra_MPEG4_16slu(const Ipp16s* pCoeffs, Ipp8u** ppBitStream,
int* pBitOffset, int countNonZero, int rvlcFlag, int noDCFlag, int scan);
```

## Parameters

<code>pCoeffs</code>	Pointer to the input coefficients.
<code>ppBitStream</code>	Pointer to the pointer to the current byte in the bitstream buffer. The pointer is updated by the function.
<code>pBitOffset</code>	Pointer to the bit position in the byte pointed by <code>**ppBitStream</code> . The pointer is updated by the function.
<code>countNonZero</code>	Number of non-zero coefficients.
<code>rvlcFlag</code>	Flag, when set to '0' indicates that VLC tables B.16, B.18, B.19, and B.21 [ISO14496] are used in encoding DCT coefficients, otherwise the reversible variable length tables B.23, B.24, and B.25 [ISO14496] are used.
<code>noDCFlag</code>	Flag, when set to '0' indicates that <code>pCoeffs</code> is set starting with zero element, otherwise - with the first element.
<code>scan</code>	Type of the scan, takes one of the following values: <code>IPPVC_SCAN_NONE</code> , indicating that no scan is performed, <code>IPPVC_SCAN_ZIGZAG</code> , indicating the classical zigzag scan. <code>IPPVC_SCAN_HORIZONTAL</code> , indicating the alternate-horinizational scan. <code>IPPVC_SCAN_VERTICAL</code> , indicating the alternate-vertical scan. See the corresponding <a href="#">enumerator</a> in the introduction to the General Functions section.

## Description

The function `ippiEncodeCoeffsIntra_MPEG4_16slu` is declared in the `ippvc.h` header file. This function performs VLC encoding of the DC coefficient (if `noDCFlag` is 0) and AC coefficients for one intra coded block in scan order defined by `scan`.

This function is used in the MPEG-4 encoder included into Intel IPP Samples. See [introduction](#) to the MPEG-4 section.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when at least one input pointer is <code>NULL</code> .
<code>ippStsBitOffsetErr</code>	Indicates an error condition if <code>*pBitOffset</code> is out of the range <code>[0, 7]</code> .

## EncodeCoeffsInter\_MPEG4

*Encodes DCT coefficients for inter coded block.*

---

## Syntax

```
ippStatus ippiEncodeCoeffsInter_MPEG4_16slu(const Ipp16s* pCoeffs, Ipp8u** ppBitStream,
int* pBitOffset, int* countNonZero, int rvlc, int scan);
```

## Parameters

<code>pCoeffs</code>	Pointer to the input coefficients.
<code>ppBitStream</code>	Pointer to the pointer to the current byte in the bitstream buffer. The pointer is updated by the function.
<code>pBitOffset</code>	Pointer to the bit position in the byte pointed by <code>**ppBitStream</code> . The pointer is updated by the function.
<code>countNonZero</code>	Number of non-zero coefficients.
<code>rvlc</code>	Flag, when set to '0' indicates that VLC tables B.17, B.18, B.20, and B.22 [ISO14496] are used in encoding DCT coefficients, otherwise the reversible variable length tables B.23, B.24, and B.25 [ISO14496] are used.
<code>scan</code>	Type of the scan, takes one of the following values: <div> <div>IPPVC_SCAN_NONE,</div> <div>indicating that no scan is performed,</div> </div> <div> <div>IPPVC_SCAN_ZIGZAG,</div> <div>indicating the classical zigzag scan,</div> </div> <div> <div>IPPVC_SCAN_VERTICAL,</div> <div>indicating the alternate-vertical scan.</div> </div> See the corresponding <a href="#">enumerator</a> in the introduction to the General Functions section.

## Description

The function `ippiEncodeCoefInter_MPEG4_16slu` is declared in the `ippvc.h` header file. This function performs VLC encoding of the DCT coefficients for one inter coded block in scan order defined by `scan`.



This function is used in the MPEG-4 encoder included into Intel IPP Samples. See [introduction](#) to the MPEG-4 section.

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when at least one input pointer is <code>NULL</code> .
<code>ippStsBitOffsetErr</code>	Indicates an error condition if <code>*pBitOffset</code> is out of the range <code>[0, 7]</code> .

## H.261

This section contains functions for encoding and decoding video data according to ITU-T Recommendation H.261 ([ITUH261]).

### H.261 Functions

Function Short Name	Description
<b>Video Decoding Functions</b>	
<a href="#">DecodeCoeffsIntra_H261</a>	Decodes DCT coefficients for intra coded block.
<a href="#">DecodeCoeffsInter_H261</a>	Decodes DCT coefficients for inter coded block.
<a href="#">ReconstructCoeffsIntra_H261</a>	Reconstructs DCT coefficients for intra coded block.
<a href="#">ReconstructCoeffsInter_H261</a>	Reconstructs DCT coefficients for inter coded block.
<b>Video Encoding Functions</b>	
<a href="#">EncodeCoeffsIntra_H261</a>	Encodes and puts quantized DCT coefficients for intra coded block into bitstream.
<a href="#">EncodeCoeffsInter_H261</a>	Encodes and puts quantized DCT coefficients for inter coded block into bitstream.
<a href="#">Filter8x8_H261</a>	Performs two-dimensional filtering on a block.

The use of some functions described in this section is demonstrated in Intel® IPP Samples downloadable from <http://www.intel.com/cd/software/products/asmo-na/eng/220046.htm>.

### H.261 Decoder Functions

This section describes Intel IPP functions that support the decoder and common operations specific to ITU-T Recommendation H.261.

The implemented IPP functions, along with those described in the [General Functions](#) section, cover the following aspects of H.261 Decoder:

- Block-based variable length code (VLC) decoding and inverse zigzag scan
- Inverse quantization, inverse zigzag positioning, and IDCT

- Block layer coefficient reconstruction, including bitstream parsing, VLC decoding, inverse quantization, inverse zigzag positioning, with appropriate clipping on each step
- Motion compensation
- Two-dimensional loop filtering.

The following subsections of this section give a high-level description of the H.261 Decoder functions hierarchy, followed by macro/data structures used in the functions and detailed descriptions of individual functions.

## Decoding INTRA and INTER Macroblocks

### INTRA Macroblock Decoding

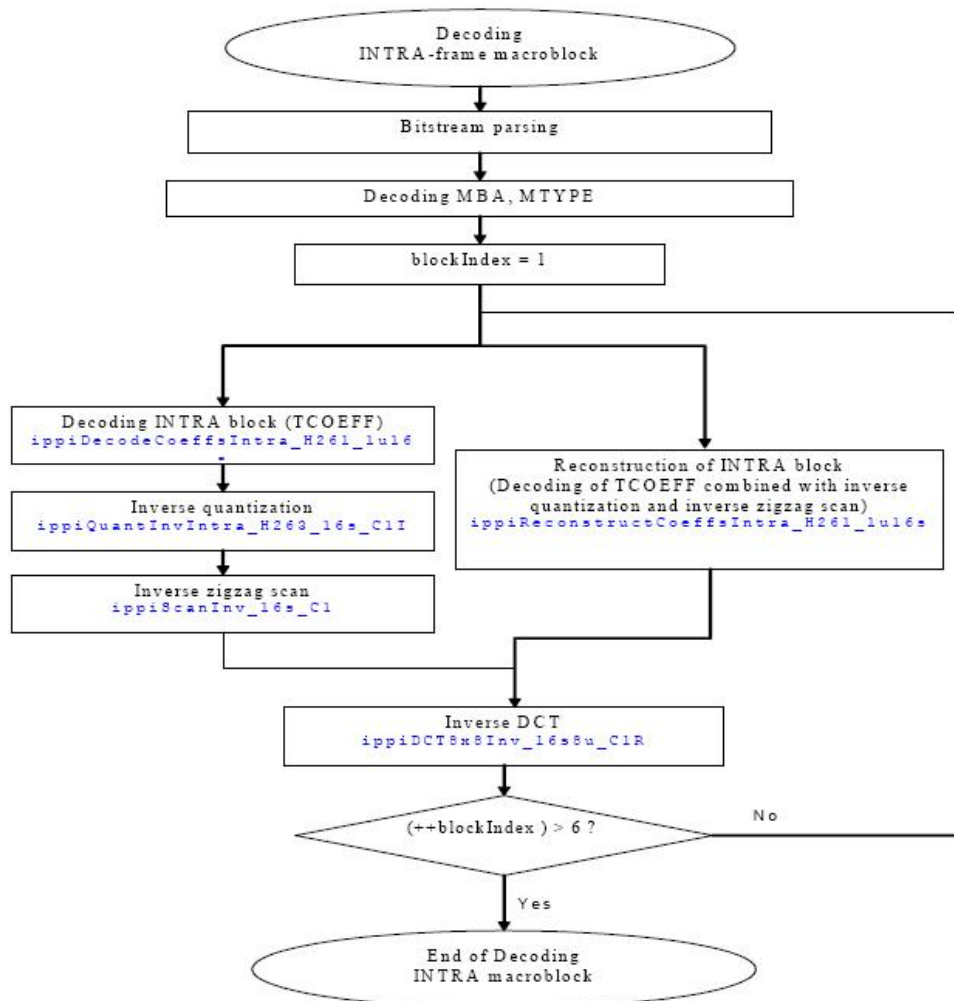
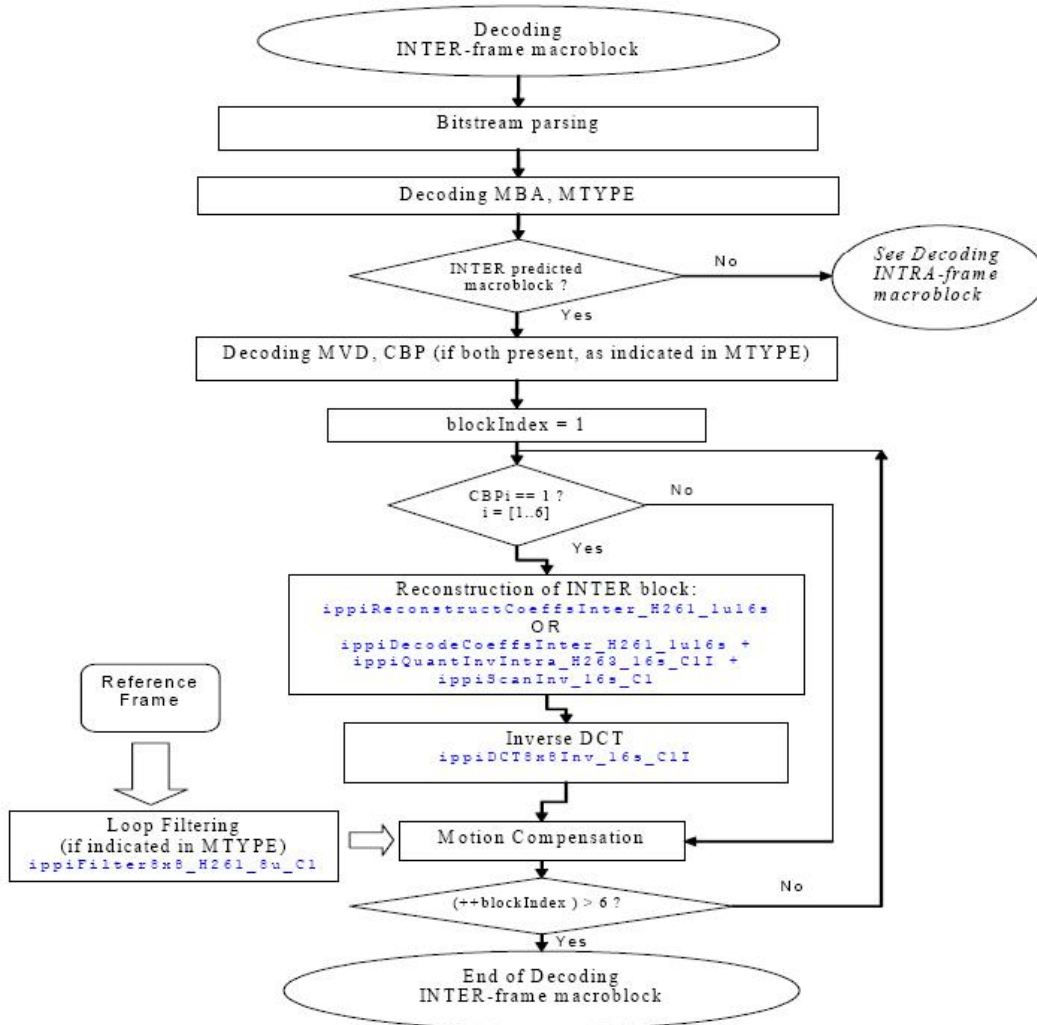


Figure shows the process of INTER-frame macroblock decoding.

### INTER Macroblock Decoding



### DecodeCoeffsIntra\_H261

*Decodes DCT coefficients for intra coded block.*

#### Syntax

```
IppStatus ippiDecodeCoeffsIntra_H261_1u16s(Ipp8u** ppBitStream, int* pBitOffset, Ipp16s* pCoef, int* pIndxLastNonZero, int scan);
```

## Parameters

<i>ppBitStream</i>	Pointer to pointer to the current byte in the bitstream buffer. <i>**ppBitStream</i> is updated by the function.
<i>pBitOffset</i>	Pointer to the bit position in the byte pointed by <i>**ppBitStream</i> . Valid within the range 0 to 7. <i>*pBitOffset</i> is updated by the function.
<i>pCoef</i>	Pointer to the output coefficients. <i>pCoef</i> [0] is the DC coefficient.
<i>pIndxLastNonZero</i>	Pointer to the index of the last non-zero coefficient in the scanning order. If an error is detected while decoding a coefficient, the index of the last decoded coefficient is returned in <i>*pIndxLastNonZero</i> . If the block has no correctly decoded coefficients, <i>*pIndxLastNonZero</i> is set to -1.
<i>scan</i>	Type of the inverse scan, takes one of the following values: IPPVC_SCAN_ZIGZAG, indicating the classical zigzag scan, IPPVC_SCAN_NONE, indicating that no inverse scan is to be performed. See the corresponding <a href="#">enumerator</a> in the introduction to the General Functions section.

## Description

The function `ippiDecodeCoeffsIntra_H261_1u16s` is declared in the `ippvc.h` header file. This function performs decoding and, optionally, inverse scan of the DCT coefficients (DC and AC) for one intra coded block. DC fixed length and AC VLC decoding processes are specified in [ITUH261], subclause 4.2.4.1. If *scan* is not set to `IPPVC_SCAN_NONE`, the DCT coefficients, encoded in the bitstream in the classical zigzag scan order ([ITUH261], Figure12), are reordered in the function into the normal order, that is, the order in which the coefficients are arranged on DCT output.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if at least one of the specified pointers is NULL.
<code>ippStsBitOffsetErr</code>	Indicates an error condition if <i>*pBitOffset</i> is out of the range [0, 7].
<code>ippStsVLCCodeErr</code>	Indicates an error condition if an illegal code is detected through the stream processing.

## DecodeCoeffsInter\_H261

*Decodes DCT coefficients for inter coded block.*

---

## Syntax

```
IppStatus ippiDecodeCoeffsInter_H261_1u16s(Ipp8u** ppBitStream, int* pBitOffset, Ipp16s* pCoef, int* pIndxLastNonZero, int scan);
```

## Parameters

<i>ppBitStream</i>	Pointer to pointer to the current byte in the bitstream buffer. <i>**ppBitStream</i> is updated by the function.
<i>pBitOffset</i>	Pointer to the bit position in the byte pointed by <i>**ppBitStream</i> . Valid within the range 0 to 7. <i>*pBitOffset</i> is updated by the function.
<i>pCoef</i>	Pointer to the output coefficients.
<i>pIndxLastNonZero</i>	Pointer to the index of the last non-zero coefficient in the scanning order. If an error is detected while decoding a coefficient, the index of the last decoded coefficient is returned in <i>*pIndxLastNonZero</i> . If the block has no correctly decoded coefficients, <i>*pIndxLastNonZero</i> is set to -1.
<i>scan</i>	Type of the inverse scan, takes one of the following values: IPPVC_SCAN_ZIGZAG, indicating the classical zigzag scan, IPPVC_SCAN_NONE, indicating that no inverse scan is to be performed. See the corresponding <a href="#">enumerator</a> in the introduction to the General Functions section.

## Description

The function `ippiDecodeCoeffsInter_H261_lu16s` is declared in the `ippvc.h` header file. This function performs VLC decoding and, optionally, inverse scan of the DCT coefficients for one inter coded block as specified in [ITUH261], subclause 4.2.4.1 (Table 4). If *scan* is not set to `IPPVC_SCAN_NONE`, the DCT coefficients, encoded in the bitstream in the classical zigzag scan order ([ITUH261], Figure12), are reordered in the function into the normal order, that is, the order in which the coefficients are arranged on DCT output.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if at least one of the specified pointers is NULL.
<code>ippStsBitOffsetErr</code>	Indicates an error condition if <i>*pBitOffset</i> is out of the range [0, 7].
<code>ippStsVLCCodeErr</code>	Indicates an error condition if an illegal code is detected through the VLC stream processing.

## ReconstructCoeffsIntra\_H261

*Reconstructs DCT coefficients for intra coded block.*

### Syntax

```
ippStatus ippiReconstructCoeffsIntra_H261_lu16s(Ipp8u** ppBitStream, int* pBitOffset,
Ipp16s* pCoef, int* pIndxLastNonZero, int QP);
```

## Parameters

<i>ppBitStream</i>	Pointer to pointer to the current byte in the bitstream buffer. <i>**ppBitStream</i> is updated by the function.
<i>pBitOffset</i>	Pointer to the bit position in the byte pointed by <i>**ppBitStream</i> . Valid within the range 0 to 7. <i>*pBitOffset</i> is updated by the function.
<i>pCoef</i>	Pointer to the output coefficients.
<i>pIndxLastNonZero</i>	Pointer to the index of the last non-zero coefficient in the scanning order. If an error is detected while decoding a coefficient, the index of the last decoded coefficient is returned in <i>*pIndxLastNonZero</i> . If the block has no correctly decoded coefficients, <i>*pIndxLastNonZero</i> is set to -1.
<i>QP</i>	Quantization parameter.

## Description

The function `ippiReconstructCoeffsIntra_H261_lu16s` is declared in the `ippvc.h` header file. This function performs decoding, dequantization, and inverse scanning of the DCT coefficients for one intra coded block.

DCT coefficients decoding and dequantization processes are specified in [ITUH261], subclause 4.2.4.1. The DCT coefficients, encoded in the bitstream in classical zigzag scan order ([ITUH261], Figure 12), are reordered in the function into the normal order, that is, the order, in which the coefficients are arranged on DCT output.

This function is used in the H.261 decoder included into Intel IPP Samples. See [introduction](#) to the H.261 section.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if at least one of the specified pointers is NULL.
<code>ippStsBitOffsetErr</code>	Indicates an error condition if <i>*pBitOffset</i> is out of the range [0, 7].
<code>ippStsVLCCodeErr</code>	Indicates an error condition if an illegal code is detected through the stream processing.
<code>ippStsQPErr</code>	Indicates an error condition if <i>QP</i> is out of the range [1, 31].

## ReconstructCoeffsInter\_H261

Reconstructs DCT coefficients for inter coded block.

## Syntax

```
ippStatus ippiReconstructCoeffsInter_H261_lu16s(Ipp8u** ppBitStream, int* pBitOffset,
Ipp16s* pCoef, int* pIndxLastNonZero, int QP);
```

## Parameters

<i>ppBitStream</i>	Pointer to pointer to the current byte in the bitstream buffer. <i>**ppBitStream</i> is updated by the function.
--------------------	---

<i>pBitOffset</i>	Pointer to the bit position in the byte pointed by <i>**ppBitStream</i> . Valid within the range 0 to 7. <i>*pBitOffset</i> is updated by the function.
<i>pCoef</i>	Pointer to the output coefficients.
<i>pIndxLastNonZero</i>	Pointer to the index of the last non-zero coefficient in the scanning order. If an error is detected while decoding a coefficient, the index of the last decoded coefficient is returned in <i>*pIndxLastNonZero</i> . If the block has no correctly decoded coefficients, <i>*pIndxLastNonZero</i> is set to -1.
<i>QP</i>	Quantization parameter.

## Description

The function `ippiReconstructCoeffsInter_H261_1u16s` is declared in the `ippvc.h` header file. This function performs decoding, dequantization and inverse scanning of the DCT coefficients for one inter coded block.

DCT coefficients decoding and dequantization processes are specified in [ITUH261], subclause 4.2.4.1. The DCT coefficients, encoded in the bitstream in classical zigzag scan order ([ITUH261], Figure 12), are reordered in the function into the normal order, that is, the order, in which the coefficients are arranged on DCT output.

This function is used in the H.261 decoder included into Intel IPP Samples. See [introduction](#) to the H.261 section.

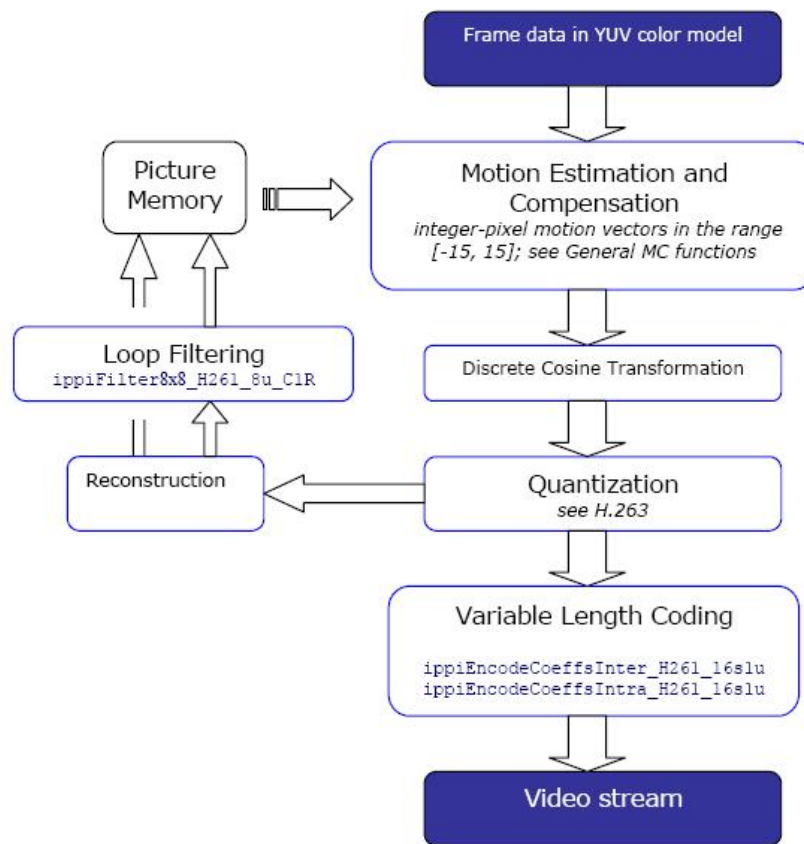
## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if at least one of the specified pointers is NULL.
<code>ippStsBitOffsetErr</code>	Indicates an error condition if <i>*pBitOffset</i> is out of the range [0, 7].
<code>ippStsVLCCodeErr</code>	Indicates an error condition if an illegal code is detected through the stream processing.
<code>ippStsQPErr</code>	Indicates an error condition if <i>QP</i> is out of the range [1, 31].

## H.261 Encoder Functions

This section describes the main steps of H.261 ([ITUH261]) video encoding in accordance with the pipeline shown in the H.261 Encoding Pipeline flowchart:

### H.261 Encoding Pipeline.



## EncodeCoeffsIntra\_H261

*Encodes and puts quantized DCT coefficients for intra coded block into bitstream.*

### Syntax

```
IppStatus ippiEncodeCoeffsIntra_H261_16slu(Ipp16s* pQCoef, Ipp8u** ppBitStream, int*
pBitOffset, int countNonZero, int scan);
```



## Parameters

<i>pQCoef</i>	Pointer to the array of quantized DCT coefficients. <i>pQCoef</i> [0] is the DC coefficient.
<i>ppBitStream</i>	Pointer to pointer to the current byte in the bitstream buffer. <i>**ppBitStream</i> is updated by the function.
<i>pBitOffset</i>	Pointer to the bit position in the byte pointed by <i>**ppBitStream</i> . Valid within the range 0 to 7. <i>*pBitOffset</i> is updated by the function.
<i>countNonZero</i>	Number of non-zero coefficients in the block. Valid within the range 1 to 64.
<i>scan</i>	Type of the scan to be performed on the coefficients before encoding, takes one of the following values:  <div style="display: flex; justify-content: space-between;"> <div>IPPVC_SCAN_ZIGZAG,</div> <div>indicating the classical zigzag scan,</div> </div> <div style="display: flex; justify-content: space-between;"> <div>IPPVC_SCAN_NONE,</div> <div>indicating that no scan is to be performed, that is, the input coefficients are already in the scan order.</div> </div> See the corresponding <a href="#">enumerator</a> in the introduction to the General Functions section.

## Description

The function `ippiEncodeCoeffsIntra_H261_16slu` is declared in the `ippvc.h` header file. This function performs encoding of the quantized DCT coefficients in a scan order for one intra coded block and puts the codes into the bitstream. DC fixed length and AC VLC encoding processes are specified in [ITUH261], subclause 4.2.4.1.

This function is used in the H.261 encoder included into Intel IPP Samples. See [introduction](#) to the H.261 section.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if at least one of the specified pointers is NULL.
<code>ippStsBitOffsetErr</code>	Indicates an error condition if <i>*pBitOffset</i> is out of the range [0, 7].
<code>ippStsOutOfRangeErr</code>	Indicates an error condition if <i>countNonZero</i> is out of the range [1, 64].

## EncodeCoeffsInter\_H261

*Encodes and puts quantized DCT coefficients for inter coded block into bitstream.*

---

## Syntax

```
ippStatus ippiEncodeCoeffsInter_H261_16slu(Ipp16s* pQCoef, Ipp8u** ppBitStream, int* pBitOffset, int countNonZero, int scan);
```

## Parameters

<i>pQCoef</i>	Pointer to the array of quantized DCT coefficients. <i>pQCoef</i> [0] is the DC coefficient.
---------------	--

<i>ppBitStream</i>	Pointer to pointer to the current byte in the bitstream buffer. <i>**ppBitStream</i> is updated by the function.
<i>pBitOffset</i>	Pointer to the bit position in the byte pointed by <i>**ppBitStream</i> . Valid within the range 0 to 7. <i>*pBitOffset</i> is updated by the function.
<i>countNonZero</i>	Number of non-zero coefficients in the block. Valid within the range 1 to 64.
<i>scan</i>	Type of the scan to be performed on the coefficients before encoding, takes one of the following values:  <div style="display: flex; justify-content: space-between;"> <div>IPPVC_SCAN_ZIGZAG,</div> <div>indicating the classical zigzag scan,</div> </div> <div style="display: flex; justify-content: space-between;"> <div>IPPVC_SCAN_NONE,</div> <div>indicating that no scan is to be performed, that is, the input coefficients are already in the scan order.</div> </div> See the corresponding <a href="#">enumerator</a> in the introduction to the General Functions section.

## Description

The function `ippiEncodeCoeffsInter_H261_16s1u` is declared in the `ippvc.h` header file. This function performs VLC encoding of the quantized DCT coefficients in a scan order for one inter coded block and puts the codes into the bitstream. The encoding process is specified in [ITUH261], subclause 4.2.4.1.

This function is used in the H.261 encoder included into Intel IPP Samples. See [introduction](#) to the H.261 section.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if at least one of the specified pointers is NULL.
<code>ippStsBitOffsetErr</code>	Indicates an error condition if <i>*pBitOffset</i> is out of the range [0, 7].
<code>ippStsOutOfRangeErr</code>	Indicates an error condition if <i>countNonZero</i> is out of the range [1, 64].

## Filter8x8\_H261

*Performs two-dimensional filtering on a block.*

---

## Syntax

```
IppStatus ippiFilter8x8_H261_8u_C1R(Ipp8u* pSrc, int srcStep, Ipp8u* pDst, int dstStep);
```

## Parameters

<i>pSrc</i>	Pointer to the origin of the source block.
<i>srcStep</i>	Width in bytes of the source image plane.
<i>pDst</i>	Pointer to the origin of the destination block.
<i>dstStep</i>	Width in bytes of the destination image plane.

## Description

The function `ippiFilter8x8_H261_8u_C1R` is declared in the `ippvc.h` header file. This function performs filtering on an 8x8 block as specified in [ITUH261], subclause 3.2.3. The two-dimensional filter may be used in both encoder and decoder to modify the prediction. The application of the filter to a macroblock (to all six blocks) is signaled in the MTYPE (macroblock type) codeword.

This function is used in the H.261 encoder and decoder included into Intel IPP Samples. See [introduction](#) to the H.261 section.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if at least one of the specified pointers is NULL.

# H.263

This section contains functions for encoding and decoding of video data according to ITU-T Recommendation H.263 ([ITUH263]) and Annexes, which is most often denoted by the "H263 + decoder" acronym.

The use of some functions described in this section is demonstrated in Intel® IPP Samples downloadable from <http://www.intel.com/cd/software/products/asmo-na/eng/220046.htm>.

## H.263 Video Decoder Functions

Function Short Name	Description
<b>VLC Decoding</b>	
<a href="#">DecodeDCIntra_H263</a>	Decodes DC coefficient for intra coded block.
<a href="#">DecodeCoeffsIntra_H263</a>	Decodes AC coefficients for intra coded block.
<a href="#">DecodeCoeffsInter_H263</a>	Decodes DCT coefficients for inter coded block.
<b>Quantization</b>	
<a href="#">QuantInvIntra_H263</a>	Performs inverse quantization on an intra coded block stored in a one-dimensional buffer.
<a href="#">QuantInvInter_H263</a>	Performs inverse quantization on an inter coded block stored in a one-dimensional buffer.
<b>Prediction</b>	
<a href="#">AddBackPredPB_H263</a>	Performs bidirectional prediction for a B-block in a PB-frame.
<b>Resampling</b>	
<a href="#">Resample_H263</a>	Performs picture resampling defined in terms of picture area corner displacements.

Function Short Name	Description
<a href="#">UpsampleFour_H263</a>	Performs factor-of-4 picture upsampling.
<a href="#">DownsampleFour_H263</a>	Performs factor-of-4 picture downsampling.
<a href="#">UpsampleFour8x8_H263</a>	Performs factor-of-4 upsampling on an 8x8 block.
<a href="#">SpatialInterpolation_H263</a>	Interpolates a picture by a factor of two horizontally, vertically, or both horizontally and vertically.
<b>Boundary Filtering</b>	
<a href="#">FilterBlockBoundaryHorEdge_H263</a> , <a href="#">FilterBlockBoundaryVerEdge_H263</a>	Perform block boundary filtering on a horizontal or vertical boundary of two adjacent 16x16 blocks.
<a href="#">FilterDeblocking8x8HorEdge_H263</a> , <a href="#">FilterDeblocking8x8VerEdge_H263</a>	Perform deblocking filtering of one block edge on the reconstructed frames.
<a href="#">FilterDeblocking16x16HorEdge_H263</a> , <a href="#">FilterDeblocking16x16VerEdge_H263</a>	Perform deblocking filtering on a horizontal or vertical boundary of two adjacent 16x16 blocks.
<b>Middle Level Functions</b>	
<a href="#">ReconstructCoeffsIntra_H263</a>	Reconstructs DCT coefficients for an intra coded block.
<a href="#">ReconstructCoeffsInter_H263</a>	Reconstructs DCT coefficients for an inter coded block.
<b>H.263 Video Encoder Functions</b>	
Function Short Name	Description
<b>VLC Encoding</b>	
<a href="#">EncodeDCIntra_H263</a>	Encodes and puts a quantized DC coefficient for an intra coded block into bitstream.
<a href="#">EncodeCoeffsIntra_H263</a>	Encodes and puts quantized DCT coefficients for an intra coded block into bitstream.
<a href="#">EncodeCoeffsInter_H263</a>	Encodes and puts quantized DCT coefficients for inter coded block into bitstream.
<b>Quantization</b>	
<a href="#">QuantIntra_H263</a>	Performs quantization on an intra coded block.
<a href="#">QuantInter_H263</a>	Performs quantization on an inter coded block.
<b>Resampling</b>	
<a href="#">DownsampleFour16x16_H263</a>	Performs factor-of-4 downsampling on a 16x16 block.

## H.263 Decoder Functions

This section describes Intel IPP functions that support general video processing and the decoder part of ITU-T Recommendation H.263 (see ([[ITUH263](#)])) and Annexes, which is often denoted by the “H263+ decoder” acronym.

The implemented Intel IPP functions, along with those described in the [General Functions](#) and [MPEG-4](#) sections, cover the following aspects of H.263+ Decoder:

- Block-based variable length code (VLC) decoding and inverse zigzag scan
- Inverse quantization, inverse zigzag positioning, and IDCT
- Block layer coefficient reconstruction, including bitstream parsing, VLC decoding, inverse quantization, inverse zigzag positioning, with appropriate clipping on each step
- Motion Compensation
- Unrestricted Motion Vectors mode (Annex D/H.263+)
- Advanced Prediction mode (Annex F/H.263+)
- PB-frames mode (Annex G/H.263+)
- Advanced Intra-Coding mode (Annex I/H.263+)
- Deblocking Filter mode (Annex J/H.263+)
- Temporal, SNR, and Spatial Scalability mode (Annex O/H.263+)
- Reference Picture Resampling (Annex P/H.263+)
- Reduced-Resolution Update mode (Annex Q/H.263+)
- Alternative INTER VLC mode (Annex S/H.263+)
- Modified Quantization mode (Annex T/H.263+).

The following subsections of this section give a high-level description of the H.263+ Decoder functions hierarchy, followed by macro/data structures used in the functions and the detailed descriptions of individual functions.

## INTRA and INTER Macroblocks Decoding

### INTRA Macroblock Decoding

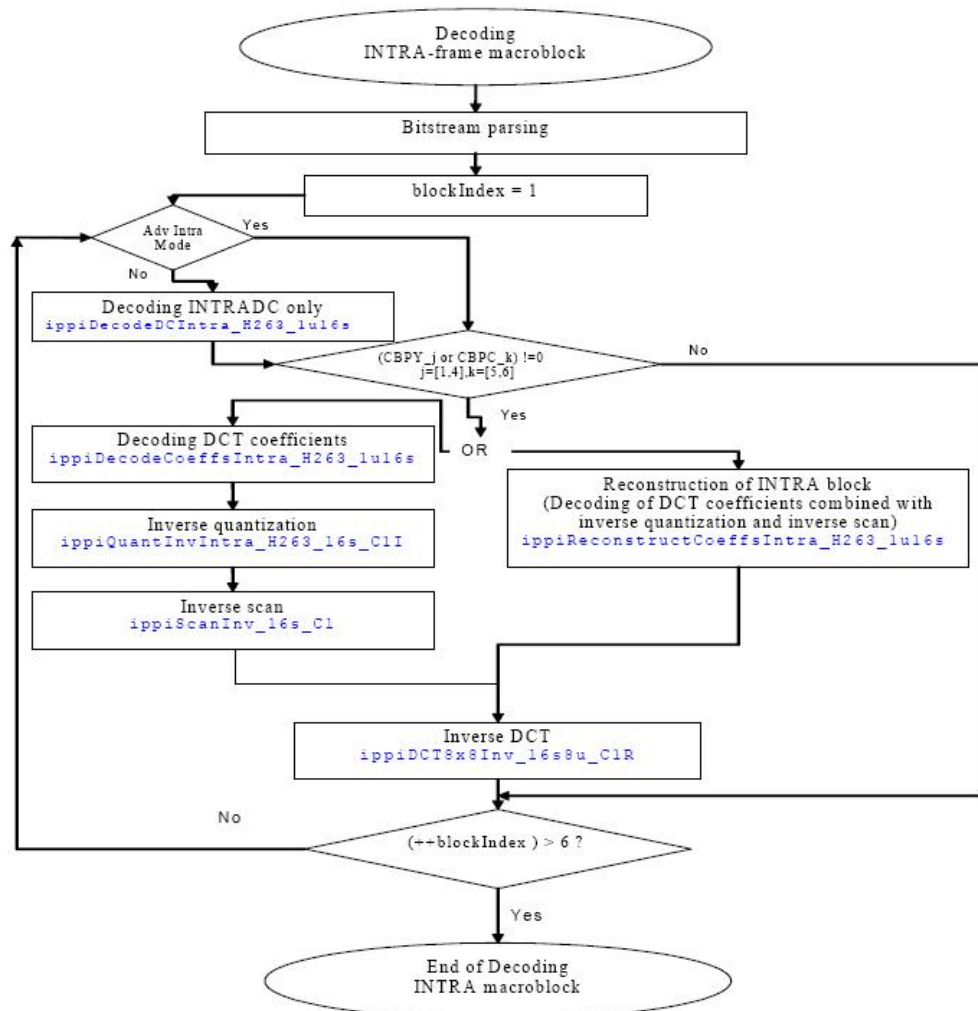
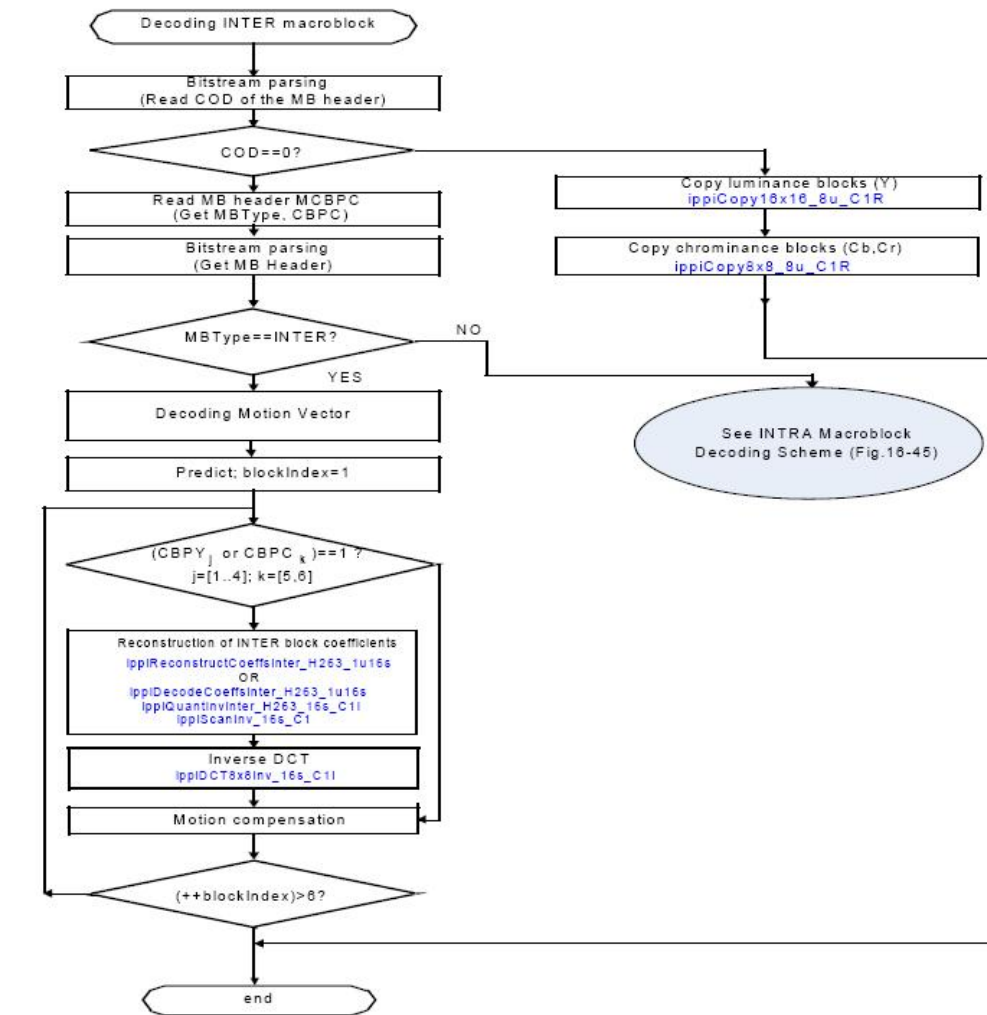


Figure "INTER Macroblock Decoding" shows the process of inter macroblock decoding.

### INTER Macroblock Decoding



### VLC Decoding

### DecodeDCIntra\_H263

Decodes DC coefficient for intra coded block.

### Syntax

```

IppStatus ippiDecodeDCIntra_H263_1u16s(Ipp8u** ppBitStream, int* pBitOffset, Ipp16s* pDC);

```

## Parameters

<i>ppBitStream</i>	Pointer to pointer to the current byte in the bitstream buffer. <i>**ppBitStream</i> is updated by the function.
<i>pBitOffset</i>	Pointer to the bit position in the byte pointed by <i>**ppBitStream</i> . Valid within the range 0 to 7. <i>*pBitOffset</i> is updated by the function.
<i>pDC</i>	Pointer to the output coefficient.

## Description

The function `ippiDecodeDCIntra_H263_1u16s` is declared in the `ippvc.h` header file. This function performs fixed length decoding of the DC coefficient for one intra coded block. Intra DC decoding process is specified in [ITUH263], subclause 5.4.1.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if at least one of the specified pointers is NULL.
<code>ippStsBitOffsetErr</code>	Indicates an error condition if <i>*pBitOffset</i> is out of the range [0, 7].
<code>ippStsVLCErr</code>	Indicates an error condition if an illegal code is detected through the DC stream processing.

## DecodeCoeffsIntra\_H263

*Decodes AC coefficients for intra coded block.*

---

## Syntax

```
IppStatus ippiDecodeCoeffsIntra_H263_1u16s(Ipp8u** ppBitStream , int* pBitOffset , Ipp16s* pCoef, int* pIndxLastNonZero, int advIntraFlag, int modQuantFlag, int scan);
```

## Parameters

<i>ppBitStream</i>	Pointer to pointer to the current byte in the bitstream buffer. <i>**ppBitStream</i> is updated by the function.
<i>pBitOffset</i>	Pointer to the bit position in the byte pointed by <i>**ppBitStream</i> . Valid within the range 0 to 7. <i>*pBitOffset</i> is updated by the function.
<i>pCoef</i>	Pointer to the output coefficients. <i>pCoef</i> [0] is the DC coefficient.
<i>pIndxLastNonZero</i>	Pointer to the index of the last non-zero coefficient in the scanning order. If an error is detected while decoding a coefficient, the index of the last decoded coefficient is returned in <i>*pIndxLastNonZero</i> . If the block has no correctly decoded coefficients, <i>*pIndxLastNonZero</i> is set to -1 when in Advanced Intra Coding mode, and to 0 otherwise.
<i>advIntraFlag</i>	Flag equal to a non-zero value when Advanced Intra Coding mode is in use, equal to 0 otherwise.
<i>modQuantFlag</i>	Flag equal to a non-zero value when Modified Quantization mode is in use, equal to 0 otherwise.



*scan* Type of the inverse scan, takes one of the following values:

IPPVC\_SCAN\_ZIGZAG, indicating the classical zigzag scan,  
 IPPVC\_SCAN\_HORIZONTAL, indicating the alternate-horizontal scan,  
 IPPVC\_SCAN\_VERTICAL indicating the alternate-vertical scan,  
 ,  
 IPPVC\_SCAN\_NONE , indicating that no inverse scan is to be performed.

See the corresponding [enumerator](#) in the introduction to the General Functions section .

## Description

The function `ippiDecodeCoeffsIntra_H263_1u16s` is declared in the `ippvc.h` header file. This function performs VLC decoding and, optionally, inverse scan of the AC coefficients for one intra coded block. Intra AC VLC decoding process is specified in [ITUH263], subclause 5.4.2, and is modified as specified in [ITUH263] Annex T, clause T.4, when Modified Quantization mode is in use. When in Advanced Intra Coding mode, VLC Table I.2 from [ITUH263] Annex I is used for all intra DC and intra AC coefficients, otherwise Table 16 [ITUH263] is used to decode AC coefficients (starting from `pCoef [1]`) only. If *scan* is not set to `IPPVC_SCAN_NONE`, the DCT coefficients, encoded in the bitstream in the classical zigzag, alternate-horizontal, or alternate-vertical scan order, are reordered in the function into the normal order, that is, the order in which the coefficients are arranged on DCT output. The three scan patterns are shown in [ITUH263], Figure14 and [ITUH263], Annex I, Figure I.2.

This function is used in the H.263 decoder included into Intel IPP Samples. See [introduction](#) to the H.263 section.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if at least one of the specified pointers is NULL.
<code>ippStsBitOffsetErr</code>	Indicates an error condition if <i>*pBitOffset</i> is out of the range [0, 7].
<code>ippStsVLCerr</code>	Indicates an error condition if an illegal code is detected through the VLC stream processing.

## DecodeCoeffsInter\_H263

*Decodes DCT coefficients for inter coded block.*

### Syntax

```
IppStatus ippiDecodeCoeffsInter_H263_1u16s(Ipp8u** ppBitStream, int* pBitOffset, Ipp16s* pCoef, int* pIndxLastNonZero, int modQuantFlag, int scan);
```

### Parameters

<i>ppBitStream</i>	Pointer to pointer to the current byte in the bitstream buffer. ** <i>ppBitStream</i> is updated by the function.
<i>pBitOffset</i>	Pointer to the bit position in the byte pointed by ** <i>ppBitStream</i> . Valid within the range 0 to 7. <i>*pBitOffset</i> is updated by the function.

<i>pCoef</i>	Pointer to the output coefficients.
<i>pIndxLastNonZero</i>	Pointer to the index of the last non-zero coefficient in the scanning order. If an error is detected while decoding a coefficient, the index of the last decoded coefficient is returned in <i>*pIndxLastNonZero</i> . If the block has no correctly decoded coefficients, <i>*pIndxLastNonZero</i> is set to -1.
<i>modQuantFlag</i>	Flag equal to a non-zero value when Modified Quantization mode is in use, equal to 0 otherwise.
<i>scan</i>	Type of the inverse scan, takes one of the following values: IPPVC_SCAN_ZIGZAG, indicating the classical zigzag scan, IPPVC_SCAN_NONE, indicating that no inverse scan is to be performed. See the corresponding <a href="#">enumerator</a> in the introduction to the General Functions section.

## Description

The function `ippiDecodeCoeffsInter_H263_1u16s` is declared in the `ippvc.h` header file. This function performs VLC decoding and, optionally, inverse scan of the DCT coefficients for one inter coded block. Inter DCT VLC decoding process is specified in [ITUH263], subclause 5.4.2 (Table 16), and is modified as specified in [ITUH263] Annex T, clause T.4, when Modified Quantization mode is in use. If *scan* is not set to `IPPVC_SCAN_NONE`, the DCT coefficients, encoded in the bitstream in the classical zigzag scan order, are reordered in the function into the normal order, that is, the order in which the coefficients are arranged on DCT output. The zigzag scan pattern is shown in [ITUH263], Figure14.

This function is used in the H.263 decoder included into Intel IPP Samples. See [introduction](#) to the H.263 section.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if at least one of the specified pointers is NULL.
<code>ippStsBitOffsetErr</code>	Indicates an error condition if <i>*pBitOffset</i> is out of the range [0, 7].
<code>ippStsVLCerr</code>	Indicates an error condition if an illegal code is detected through the VLC stream processing.

## Inverse Quantization

### QuantInvIntra\_H263

*Performs inverse quantization on an intra coded block stored in a one-dimensional buffer.*

---

## Syntax

```
ippStatus ippiQuantInvIntra_H263_16s_C1I(Ipp16s* pSrcDst, int indxLastNonZero, int QP,
int advIntraFlag, int modQuantFlag);
```

## Parameters

<i>pSrcDst</i>	Pointer to the coefficient buffer of the block. The parameter contains quantized coefficients on input and dequantized coefficients on output.
<i>indxLastNonZero</i>	Index of the last non-zero coefficient. This parameter provides faster operation. If the value is unknown, set to 63.
<i>QP</i>	Quantization parameter.
<i>advIntraFlag</i>	Flag equal to a non-zero value when Advanced Intra Coding mode is in use, equal to 0 otherwise.
<i>modQuantFlag</i>	Flag equal to a non-zero value when Modified Quantization mode is in use, equal to 0 otherwise.

## Description

The function `ippiQuantInvIntra_H263_16s_C1I` is declared in the `ippvc.h` header file. This function performs inverse quantization on intra coded block. When not in Advanced Intra Coding mode, the dequantization processes for the intra DC and for all other non-zero coefficients are specified in [ITUH263], subclause 6.2.1, otherwise all the coefficients are dequantized as specified in [ITUH263] Annex I, clause I.3. When not in Advanced Intra Coding mode and not in Modified Quantization mode, the output coefficients other than the intra DC are clipped to the range  $[-2048, 2047]$  ([ITUH263], subclause 6.2.2). The overall procedure is defined in [ITUH263] as:

if (*advIntraFlag* == 0):

$$pDst[i] = \begin{cases} 0, & pSrc[i] = 0 \\ \text{sign}(pSrc[i]) * QP * (2 * |pSrc[i]| + 1), & pSrc[i] \neq 0, \quad QP \text{ is odd} \\ \text{sign}(pSrc[i]) * (QP * (2 * |pSrc[i]| + 1) - 1), & pSrc[i] \neq 0, \quad QP \text{ is even} \end{cases}$$

where  $i = 1, 2, \dots, \text{indxLastNonZero}$

$pDst[0] = 8 * pSrc[0]$

if (*advIntraFlag* != 0):

$pDst[i] = 2 * QP * pSrc[i]$

where  $i = 0, 1, 2, \dots, \text{indxLastNonZero}$

if (*advIntraFlag* == 0 && *modQuantFlag* == 0):

$pDst[i] = \text{MIN}(\text{MAX}(pDst[i], -2048), 2047).$



**NOTE.** The function `ippiQuantInvIntra_H263_16s_C1I` can be applied to a buffer of arbitrary size (*indxLastNonZero* can be any positive number), and can thus be used, for example, to process multiple blocks in one call. In this case, for any intra block following the first one, the intra DC should be processed separately, if not in the Advanced Intra mode.

This function is used in the H.261, H.263, and MPEG-4 encoders and decoders included into Intel IPP Samples. See [introduction](#) to the H.263 section.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <code>pSrcDst</code> is <code>NULL</code> .
<code>ippStsQPErr</code>	Indicates an error condition if <code>QP</code> is out of the range <code>[1, 31]</code> .
<code>ippStsOutOfRangeErr</code>	Indicates an error condition if <code>indxLastNonZero</code> is negative.

## QuantInvInter\_H263

*Performs inverse quantization on an inter coded block stored in a one-dimensional buffer.*

---

## Syntax

```
ippStatus ippiQuantInvInter_H263_16s_C1I(Ipp16s* pSrcDst, int indxLastNonZero, int QP,
int modQuantFlag);
```

## Parameters

<code>pSrcDst</code>	Pointer to the coefficient buffer of the block. The parameter contains quantized coefficients on input and dequantized coefficients on output.
<code>indxLastNonZero</code>	Index of the last non-zero coefficient. This parameter provides faster operation. If the value is unknown, set to 63.
<code>QP</code>	Quantization parameter.
<code>modQuantFlag</code>	Flag equal to a non-zero value when Modified Quantization mode is in use, equal to 0 otherwise.

## Description

The function `ippiQuantInvInter_H263_16s_C1I` is declared in the `ippvc.h` header file. This function performs inverse quantization on inter coded block. The dequantization process is specified in [ITUH263], subclause 6.2.1. When not in Modified Quantization mode, the output coefficients are clipped to the range `[-2048, 2047]` ([ITUH263], subclause 6.2.2). The overall procedure is defined in [ITUH263] as:

$$pDst[i] = \begin{cases} 0, & pSrc[i] = 0 \\ \text{Sign}(pSrc[i]) * QP * (2 * |pSrc[i]| + 1), & pSrc[i] \neq 0, \text{ } QP \text{ is odd} \\ \text{Sign}(pSrc[i]) * (QP * (2 * |pSrc[i]| + 1) - 1), & pSrc[i] \neq 0, \text{ } QP \text{ is even} \end{cases}$$

where  $i = 0, 1, 2, \dots, \text{indxLastNonZero}$

if  $(\text{modQuantFlag} == 0)$ :

$pDst[i] = \text{MIN}(\text{MAX}(pDst[i], -2048), 2047)$ .



**NOTE.** The function `ippiQuantInvInter_H263_16s_C1I` can be applied to a buffer of arbitrary size (`indxLastNonZero` can be any positive number), and can thus be used, for example, to process multiple blocks in one call. In this case, for any intra block following the first one, the intra DC should be processed separately, if not in Advanced Intra mode.

This function is used in the H.261, H.263, and MPEG-4 encoders and decoders included into Intel IPP Samples. See [introduction](#) to the H.263 section.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <code>pSrcDst</code> is NULL.
<code>ippStsQPErr</code>	Indicates an error condition if <code>QP</code> is out of the range [1, 31].
<code>ippStsOutOfRangeErr</code>	Indicates an error condition if <code>indxLastNonZero</code> is negative.

## Prediction

### AddBackPredPB\_H263

*Performs bidirectional prediction for a B-block in a PB-frame.*

## Syntax

```
IppStatus ippiAddBackPredPB_H263_8u_C1R(const Ipp8u* pSrc, int srcStep, IppiSize srcRoiSize,
Ipp8u* pSrcDst, int srcDstStep, int acc);
```

## Parameters

<code>pSrc</code>	Pointer to the origin of the source image (P-macroblock) region of interest (ROI).
<code>srcStep</code>	Width in bytes of the source image plane, that is, distance in bytes between the starting ends of consecutive lines of the source image.
<code>srcRoiSize</code>	Size of the source ROI.
<code>pSrcDst</code>	Pointer to the origin of the source-destination image ROI, that is, bidirectionally-predicted part of the block.
<code>srcDstStep</code>	Width in bytes of the source-destination image plane.
<code>acc</code>	Parameter that defines pixel accuracy for backward prediction: bit 0 (the least significant bit) contains the half-pixel offset in horizontal direction, bit 1 - the offset in vertical direction.

## Description

The function `ippiAddBackPredPB_H263_8u_C1R` is declared in the `ippvc.h` header file. This function calculates backward prediction for a B-block of a PB-frame and adds it to the block, previously reconstructed with forward prediction. All the operations are restricted to the bidirectionally-predicted part of the B-block. The parameter `srcRoiSize` defines the area size. The backward prediction is performed with pixel accuracy defined by `acc`,

and the sum of the forward and backward predictions for every pixel within *srcRoiSize* is divided by 2 (division by truncation). The bidirectional prediction procedure is specified in [ITUH263], Annex G, clause G.5, bidirectionally- and forward-predicted areas for a B-block are demonstrated in [ITUH263], Annex G, Figure G.2.

This function is used in the H.263 decoder included into Intel IPP Samples. See [introduction](#) to the H.263 section.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if at least one of the specified pointers is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>srcRoiSize</i> has a field with zero or negative value.

## Resampling

### Resample\_H263

*Performs picture resampling defined in terms of picture area corner displacements.*

---

## Syntax

```
ippStatus ippiResample_H263_8u_P3R(const Ipp8u* pSrcY, int srcYStep, IppiSize ySrcRoiSize,
const Ipp8u* pSrcCb, int srcCbStep, const Ipp8u* pSrcCr, int srcCrStep, Ipp8u* pDstY, int
dstYStep, IppiSize dstYRoiSize, Ipp8u* pDstCb, int dstCbStep, Ipp8u* pDstCr, int dstCrStep,
IppMotionVector warpParams[4], int wda, int rounding, int fillMode, int fillColor[3]);
```

## Parameters

<i>pSrcY</i>	Pointer to the origin of the source image region of interest (ROI) in the luminance plane.
<i>srcYStep</i>	Width in bytes of the source image luminance (Y) plane, that is, distance in bytes between the starting ends of consecutive lines of the source image in the luminance plane.
<i>ySrcRoiSize</i>	Size of the source ROI in the luminance plane.
<i>pSrcCb</i>	Pointer to the origin of the source ROI in Cb chrominance plane.
<i>srcCbStep</i>	Width in bytes of the source image Cb chrominance plane.
<i>pSrcCr</i>	Pointer to the origin of the source image ROI in Cr chrominance plane.
<i>srcCrStep</i>	Width in bytes of the source image Cr chrominance plane.
<i>pDstY</i>	Pointer to the origin of the destination image ROI in the luminance plane.
<i>dstYStep</i>	Width in bytes of the destination image luminance plane.
<i>yDstRoiSize</i>	Size of the destination ROI in the luminance plane.
<i>pDstCb</i>	Pointer to the origin of the destination image ROI in Cb chrominance plane.
<i>dstCbStep</i>	Width in bytes of the destination image Cb chrominance plane.

<i>pDstCr</i>	Pointer to the origin of the destination image ROI in Cr chrominance plane.
<i>dstCrStep</i>	Width in bytes of the destination image Cr chrominance plane.
<i>warpParams</i>	Array of warping parameters - 4 pairs of motion vectors, describing, in the order they are stored in the array, how the upper left, upper right, lower left, and lower right corners of the destination ROI are mapped onto the source image.
<i>wda</i>	Warping displacement accuracy flag, if set to 0, pixel displacements are quantized to half-pixel accuracy, otherwise - to 1/16-pixel accuracy.
<i>rounding</i>	Rounding value that is used in pixel interpolation, can be 0 or 1.
<i>fillMode</i>	Flag that defines the fill-mode action for the values of the source pixels for which the calculated location in the source image lies outside of the source image ROI. This parameter takes one of the following values: <ul style="list-style-type: none"> <li>0, indicating color fill mode, the “outside” Y, Cb, and Cr pixel values are set to <i>fillColor</i>[0], <i>fillColor</i>[1], and <i>fillColor</i>[2], respectively.</li> <li>1 - <i>black</i> fill mode, the “outside” pixel values are set as follows: Y = 16, Cb = Cr = 128.</li> <li>2 - <i>gray</i> fill mode, the “outside” pixel values are all set to 128.</li> <li>3 - <i>blackclip</i> fill mode, the “outside” pixel values are extrapolated from the values of pixels at the ROI border, as specified in [ITUH263], Annex D.</li> </ul>
<i>fillColor</i>	Array of fill color values used in color fill mode.

## Description

The function `ippiResample_H263_8u_P3R` is declared in the `ippvc.h` header file. This function resamples a YCbCr picture as specified in [ITUH263], Annex P. The destination picture ROI is mapped onto the source picture ROI as defined by *warpParams*. The pixels falling outside the source image ROI are estimated according to *fillMode*.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if at least one of the specified pointers is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>yRoiSize</i> or <i>yDstRoiSize</i> have a field that is odd or less than 4.

## UpsampleFour\_H263

*Performs factor-of-4 picture upsampling.*

---

### Syntax

```
IppStatus ippiUpsampleFour_H263_8u_C1R(const Ipp8u* pSrc, int srcStep, IppiSize srcRoiSize,
Ipp8u* pDst, int dstStep, int rounding, int fillColor);
```

### Parameters

<i>pSrc</i>	Pointer to the origin of the source image region of interest (ROI).
<i>srcStep</i>	Width in bytes of the source image plane, that is, distance in bytes between the starting ends of consecutive lines of the source image.
<i>srcRoiSize</i>	Size of the source ROI.
<i>pDst</i>	Pointer to the origin of the destination image ROI.
<i>dstStep</i>	Width in bytes of the destination image plane.
<i>rounding</i>	Rounding value used in pixel interpolation, can be 0 or 1.
<i>fillColor</i>	Fill color value used for the source pixels for which the calculated location in the source image lies outside of the source image ROI. When <i>fillColor</i> is negative, <i>clip</i> fill-mode action is employed - the "outside" pixel values are extrapolated from the values of pixels at the ROI border, as specified in [ITUH263], Annex D.

### Description

The function `ippiUpsampleFour_H263_8u_C1R` is declared in the `ippvc.h` header file. This function performs factor-of-4 picture upsampling as specified in [ITUH263], Annex P, subclause P.5.1.

This function is used in the H.263 decoder included into Intel IPP Samples. See [introduction](#) to the H.263 section.

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if at least one of the specified pointers is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>srcRoiSize</i> has a field that is odd or less than 4.

## DownsampleFour\_H263

*Performs factor-of-4 picture downsampling.*

---

### Syntax

```
IppStatus ippiDownsampleFour_H263_8u_C1R(const Ipp8u* pSrc, int srcStep, IppiSize srcRoiSize,
Ipp8u* pDst, int dstStep, int rounding);
```



## Parameters

<i>pSrc</i>	Pointer to the origin of the source image region of interest (ROI).
<i>srcStep</i>	Width in bytes of the source image plane, that is, distance in bytes between the starting ends of consecutive lines of the source image.
<i>srcRoiSize</i>	Size of the source ROI.
<i>pDst</i>	Pointer to the origin of the destination image ROI.
<i>dstStep</i>	Width in bytes of the destination image plane.
<i>rounding</i>	Rounding value used in pixel interpolation, can be 0 or 1.

## Description

The function `ippiDownsampleFour_H263_8u_C1R` is declared in the `ippvc.h` header file. This function performs factor-of-4 picture downsampling as specified in [ITUH263], Annex P, subclause P.5.2.

This function is used in the H.263 decoder included into Intel IPP Samples. See [introduction](#) to the H.263 section.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if at least one of the specified pointers is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>srcRoiSize</i> has a field with zero or negative value.

## UpsampleFour8x8\_H263

*Performs factor-of-4 upsampling on an 8x8 block.*

## Syntax

```
ippStatus ippiUpsampleFour8x8_H263_16s_C1R(const Ipp16s* pSrc, int srcStep, Ipp16s* pDst,
int dstStep);
```

## Parameters

<i>pSrc</i>	Pointer to the origin of the source 8x8 block.
<i>srcStep</i>	Width in bytes of the source image plane, that is, distance in bytes between the starting ends of consecutive lines of the source block.
<i>pDst</i>	Pointer to the origin of the destination 16x16 block.
<i>dstStep</i>	Width in bytes of the destination image plane.

## Description

The function `ippiUpsampleFour8x8_H263_16s_C1R` is declared in the `ippvc.h` header file. This function performs factor-of-4 upsampling of an 8x8 source block to a 16x16 destination block, as specified in [ITUH263], Annex Q, clause Q.6.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if at least one of the specified pointers is NULL.

## SpatialInterpolation\_H263

*Interpolates a picture by a factor of two horizontally, vertically, or both horizontally and vertically.*

---

### Syntax

```
IppStatus ippiSpatialInterpolation_H263_8u_C1R(const Ipp8u* pSrc, int srcStep, IppiSize srcRoiSize, Ipp8u* pDst, int dstStep, int interpType);
```

### Parameters

<code>pSrc</code>	Pointer to the origin of the source image region of interest (ROI).
<code>srcStep</code>	Width in bytes of the source image plane, that is, distance in bytes between the starting ends of consecutive lines of the source image.
<code>srcRoiSize</code>	Size of the source ROI.
<code>pDst</code>	Pointer to the origin of the destination image ROI.
<code>dstStep</code>	Width in bytes of the destination image plane.
<code>interpType</code>	Interpolation type, takes one of the following values: IPFVC_INTERP_HORIZONTAL, indicating one-dimensional (1-D) horizontal interpolation, IPFVC_INTERP_VERTICAL, indicating 1-D vertical interpolation, IPFVC_INTERP_2D, indicating 2-D interpolation. See the corresponding <a href="#">enumerator</a> in the introduction to the General Functions section.

### Description

The function `ippiSpatialInterpolation_H263_8u_C1R` is declared in the `ippvc.h` header file. This function performs picture interpolation for 1-D or 2-D spatial scalability, as specified in [ITUH263], Annex O, clause O.6. Depending on `interpType`, the source image ROI is interpolated by a factor of 2 either horizontally, vertically, or both horizontally and vertically, the size of the destination image ROI being equal to the size of the source image ROI multiplied by 2 in the direction(s) for which the interpolation is performed.

This function is used in the H.263 decoder included into Intel IPP Samples. See [introduction](#) to the H.263 section.

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if at least one of the specified pointers is NULL.

`ippStsSizeErr` Indicates an error condition if `srcRoiSize` has a field that is odd or less than 4.

## Boundary Filtering

### FilterBlockBoundaryHorEdge\_H263, FilterBlockBoundaryVerEdge\_H263

*Perform block boundary filtering on a horizontal or vertical boundary of two adjacent 16x16 blocks.*

#### Syntax

```
ippStatus ippiFilterBlockBoundaryHorEdge_H263_8u_C1IR(Ipp8u* pSrcDst, int srcDstStep);
ippStatus ippiFilterBlockBoundaryVerEdge_H263_8u_C1IR(Ipp8u* pSrcDst, int srcDstStep);
```

#### Parameters

`pSrcDst` Pointer to the origin of the lower (HorEdge) or the right (VerEdge) 16x16 block.

`srcDstStep` Width in bytes of the image plane.

#### Description

The functions `ippiFilterBlockBoundaryHorEdge_H263_8u_C1IR` and `ippiFilterBlockBoundaryVerEdge_H263_8u_C1IR` are declared in the `ippvc.h` header file. These functions perform block boundary filtering on bordering edges, horizontal and vertical respectively, of two adjacent 16x16 blocks, as specified in [ITUH263], Annex Q, subclause Q.7.1.

#### Return Values

`ippStsNoErr` Indicates no error.

`ippStsNullPtrErr` Indicates an error condition if `pSrcDst` is NULL.

### FilterDeblocking8x8HorEdge\_H263, FilterDeblocking8x8VerEdge\_H263

*Perform deblocking filtering of one block edge on the reconstructed frames.*

#### Syntax

```
ippStatus ippiFilterDeblocking8x8HorEdge_H263_8u_C1IR(Ipp8u* pSrcDst, int srcDstStep, int QP);
ippStatus ippiFilterDeblocking8x8VerEdge_H263_8u_C1IR(Ipp8u* pSrcDst, int srcDstStep, int QP);
```

#### Parameters

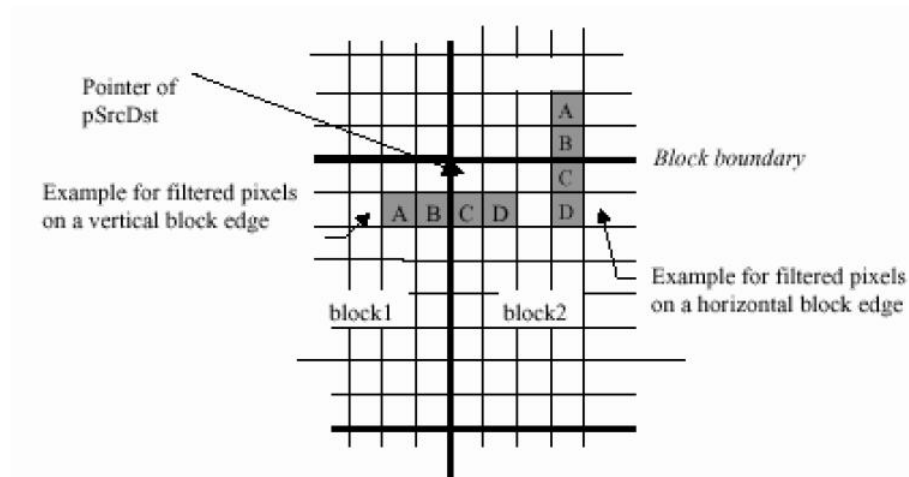
`pSrcDst` Pointer to the first pixel of the second block (block 2) of the two applied blocks.

<i>srcDstStep</i>	Width of the source and destination plane.
<i>QP</i>	Quantization parameter. The value of <i>QP</i> is found as described in Section J.3 of Annex J/H.263+.

## Description

The functions `ippiFilterDeblocking8x8HorEdge_H263_8u_C1IR` and `ippiFilterDeblocking_VerEdge8x8H263_8u_C1IR` are declared in the `ippvc.h` file. These functions perform deblocking filtering of one block edge (horizontal or vertical, respectively) on the reconstructed frames. The pointer *pSrcDst* points to the first pixel of the block 2 as shown in [Figure "Deblocking Filtering Layout"](#).

## Deblocking Filtering Layout



These functions are used in the H.263 encoder and decoder included into Intel IPP Samples. See [introduction](#) to the H.263 section.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pSrcDst</i> pointer is NULL.
<code>ippStsQPErr</code>	Indicates an error condition if the quantizer value has a zero or negative value, or if it is greater than 31.

## FilterDeblocking16x16HorEdge\_H263, FilterDeblocking16x16VerEdge\_H263

*Perform deblocking filtering on a horizontal or vertical boundary of two adjacent 16x16 blocks.*

## Syntax

```
ippStatus ippiFilterDeblocking16x16HorEdge_H263_8u_C1IR(Ipp8u* pSrcDst, int srcDstStep);
```

```
IppStatus ippiFilterDeblocking16x16VerEdge_H263_8u_C1IR(Ipp8u* pSrcDst, int srcDstStep);
```

## Parameters

<i>pSrcDst</i>	Pointer to the origin of the lower (HorEdge) or the right (VerEdge) 16x16 block.
<i>step</i>	Width in bytes of the image plane.

## Description

The functions `ippiFilterDeblocking16x16HorEdge_H263_8u_C1IR` and `ippiFilterDeblocking16x16VerEdge_H263_8u_C1IR` are declared in the `ippvc.h` header file. These functions perform deblocking filtering on bordering edges, horizontal and vertical respectively, of two adjacent 16x16 blocks, as specified in [ITUH263], Annex Q, subclause Q.7.2 and Annex J, clause J.3.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pSrcDst</i> is NULL.

## Middle Level Functions

## ReconstructCoeffsIntra\_H263

Reconstructs DCT coefficients for an intra coded block.

## Syntax

```
IppStatus ippiReconstructCoeffsIntra_H263_1u16s(Ipp8u** ppBitStream, int* pBitOffset,
Ipp16s* pCoef, int* pIndxLastNonZero, int cbp, int QP, int advIntraFlag, int scan, int
modQuantFlag);
```

## Parameters

<i>ppBitStream</i>	Pointer to pointer to the current byte in the bitstream buffer. <i>**ppBitStream</i> is updated by the function.
<i>pBitOffset</i>	Pointer to the bit position in the byte pointed by <i>**ppBitStream</i> . Valid within the range 0 to 7. <i>*pBitOffset</i> is updated by the function.
<i>pCoef</i>	Pointer to the output coefficients.
<i>pIndxLastNonZero</i>	Pointer to the index of the last non-zero coefficient in the scanning order. If an error is detected while decoding a coefficient, the index of the last decoded coefficient is returned in <i>*pIndxLastNonZero</i> . If the block has no correctly decoded coefficients, <i>*pIndxLastNonZero</i> is set to -1.
<i>cbp</i>	Coded block pattern, when set to 0 indicates that the block contains only intra DC coefficient.
<i>QP</i>	Quantization parameter.
<i>advIntraFlag</i>	Flag equal to a non-zero value when Advanced Intra Coding mode is in use, equal to 0 otherwise.

<i>scan</i>	Type of the inverse scan, takes one of the following values: IPPVC_SCAN_ZIGZAG, indicating the classical zigzag scan, IPPVC_SCAN_HORIZONTAL, indicating the alternate-horizontal scan, IPPVC_SCAN_VERTICAL, indicating alternate-vertical scan.  See the corresponding <a href="#">enumerator</a> in the introduction to the General Functions section.
<i>modQuantFlag</i>	Flag equal to a non-zero value when Modified Quantization mode is in use, equal to 0 otherwise.

## Description

The function `ippiReconstructCoeffsIntra_H263_1u16s` is declared in the `ippvc.h` header file. This function performs decoding, dequantization, and inverse scan of the DCT coefficients for one intra coded block.

Intra DC decoding process is specified in [ITUH263], subclause 5.4.1. Intra AC VLC decoding process is specified in [ITUH263], subclause 5.4.2, and is modified as specified in [ITUH263] Annex T, clause T.4, when Modified Quantization mode is in use. When in Advanced Intra Coding mode, VLC Table I.2 from [ITUH263] Annex I is used for all intra DC and intra AC coefficients, otherwise Table 16 [ITUH263] is used to decode AC coefficients only (for blocks with non-zero *cbp*). When not in Advanced Intra Coding mode, the dequantization processes for the intra DC and for all other non-zero coefficients are specified in [ITUH263], subclause 6.2.1, otherwise all the coefficients are dequantized as specified in [ITUH263] Annex I, clause I.3. When not in Advanced Intra Coding mode and not in Modified Quantization mode, the output coefficients other than the intra DC one are clipped to the range [-2048, 2047] ([ITUH263], subclause 6.2.2).

The DCT coefficients, encoded in the bitstream in the classical zigzag, alternate-horizontal, or alternate-vertical scan order, are reordered in the function into the normal order, that is, the order in which the coefficients are arranged on DCT output. The three scan patterns are shown in [ITUH263], Figure14 and [ITUH263], Annex I, Figure I.2.

This function is used in the H.263 and MPEG-4 decoders included into Intel IPP Samples. See [introduction](#) to the H.263 section.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if at least one of the specified pointers is NULL.
<code>ippStsBitOffsetErr</code>	Indicates an error condition if <i>*pBitOffset</i> is out of the range [0, 7].
<code>ippStsVLCerr</code>	Indicates an error condition if an illegal code is detected through the stream processing.
<code>ippStsQPerr</code>	Indicates an error condition if <i>QP</i> is out of the range [1, 31].

## ReconstructCoeffsInter\_H263

Reconstructs DCT coefficients for an inter coded block.

### Syntax

```
ippStatus ippiReconstructCoeffsInter_H263_1u16s(Ipp8u** ppBitStream, int* pBitOffset,
Ipp16s* pCoef, int* pIdxLastNonZero, int QP, int modQuantFlag);
```

## Parameters

<i>ppBitStream</i>	Pointer to pointer to the current byte in the bitstream buffer. <i>**ppBitStream</i> is updated by the function.
<i>pBitOffset</i>	Pointer to the bit position in the byte pointed by <i>**ppBitStream</i> . Valid within the range 0 to 7. <i>*pBitOffset</i> is updated by the function.
<i>pCoef</i>	Pointer to the output coefficients.
<i>pIndxLastNonZero</i>	Pointer to the index of the last non-zero coefficient in the scanning order. If an error is detected while decoding a coefficient, the index of the last decoded coefficient is returned in <i>*pIndxLastNonZero</i> . If the block has no correctly decoded coefficients, <i>*pIndxLastNonZero</i> is set to -1.
<i>QP</i>	Quantization parameter.
<i>modQuantFlag</i>	Flag equal to a non-zero value when Modified Quantization mode is in use, equal to 0 otherwise.

## Description

The function `ippiReconstructCoeffsInter_H263_1u16s` is declared in the `ippvc.h` header file. This function performs decoding, dequantization, and inverse scan of the DCT coefficients for one inter coded block.

Inter DCT VLC decoding process is specified in [ITUH263], subclause 5.4.2 (Table 16), and is modified as specified in [ITUH263] Annex T, clause T.4, when Modified Quantization mode is in use. The dequantization process is specified in [ITUH263], subclause 6.2.1. When not in Modified Quantization mode, the output coefficients are clipped to the range [-2048, 2047] ([ITUH263], subclause 6.2.2). The DCT coefficients, encoded in the bitstream in the classical zigzag scan order ([ITUH263], Figure14), are reordered in the function into the normal order, that is, the order in which the coefficients are arranged on DCT output.

This function is used in the H.263 and MPEG-4 decoders included into Intel IPP Samples. See [introduction](#) to the H.263 section.

## Return Values

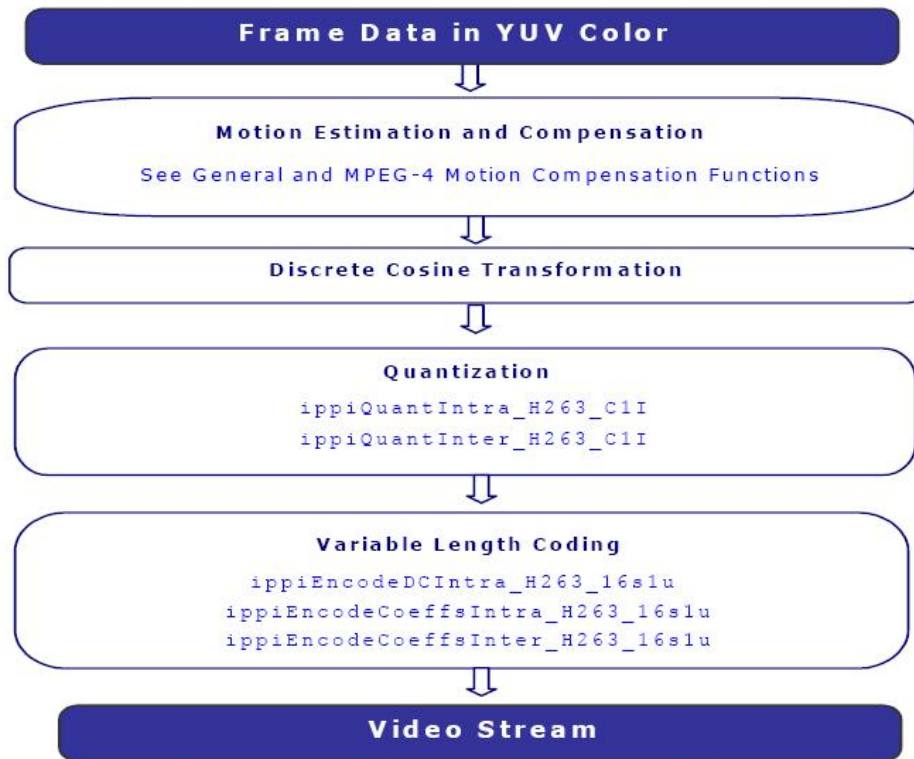
<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if at least one of the specified pointers is NULL.
<code>ippStsBitOffsetErr</code>	Indicates an error condition if <i>*pBitOffset</i> is out of the range [0, 7].
<code>ippStsVLCerr</code>	Indicates an error condition if an illegal code is detected through the stream processing.
<code>ippStsQPErr</code>	Indicates an error condition if <i>QP</i> is out of the range [1, 31].

## H.263 Encoder Functions

This section describes the main steps of H.263 ([ITUH263]) video encoding in accordance with the pipeline shown in Figure "H.263 Encoding Pipeline" .

### H.263 Encoding Pipeline

---



## VLC Encoding

### EncodeDCIntra\_H263

*Encodes and puts a quantized DC coefficient for an intra coded block into bitstream.*

---

#### Syntax

```
IppStatus ippiEncodeDCIntra_H263_16s1u(Ipp16s qDC, Ipp8u** ppBitStream, int* pBitOffset);
```



## Parameters

<i>qDC</i>	Quantized DC coefficient.
<i>ppBitStream</i>	Pointer to pointer to the current byte in the bitstream buffer. <i>**ppBitStream</i> is updated by the function.
<i>pBitOffset</i>	Pointer to the bit position in the byte pointed by <i>**ppBitStream</i> . Valid within the range 0 to 7. <i>*pBitOffset</i> is updated by the function.

## Description

The function `ippiEncodeDCIntra_H263_16slu` is declared in the `ippvc.h` header file. This function performs fixed length encoding of the DC coefficient for one intra coded block and puts the code into the bitstream. Intra DC encoding process is specified in [ITUH263], subclause 5.4.1.

This function is used in the H.263 and MPEG-4 encoders included into Intel IPP Samples. See [introduction](#) to the H.263 section.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if at least one of the specified pointers is NULL.
<code>ippStsBitOffsetErr</code>	Indicates an error condition if <i>*pBitOffset</i> is out of the range [0, 7].

## EncodeCoeffsIntra\_H263

*Encodes and puts quantized DCT coefficients for an intra coded block into bitstream.*

## Syntax

```
ippStatus ippiEncodeCoeffsIntra_H263_16slu(Ipp16s* pQCoef, Ipp8u** ppBitStream, int* pBitOffset, int countNonZero, int advIntraFlag, int modQuantFlag, int scan);
```

## Parameters

<i>pQCoef</i>	Pointer to the array of quantized DCT coefficients. <i>pQCoef</i> [0] is the DC coefficient.
<i>ppBitStream</i>	Pointer to pointer to the current byte in the bitstream buffer. <i>**ppBitStream</i> is updated by the function.
<i>pBitOffset</i>	Pointer to the bit position in the byte pointed by <i>**ppBitStream</i> . Valid within the range 0 to 7. <i>*pBitOffset</i> is updated by the function.
<i>countNonZero</i>	Number of non-zero coefficients in the block. Valid within the range 1 to 64.
<i>advIntraFlag</i>	Flag equal to a non-zero value when Advanced Intra Coding mode is in use, equal to 0 otherwise.
<i>modQuantFlag</i>	Flag equal to a non-zero value when Modified Quantization mode is in use, equal to 0 otherwise.

*scan*

Type of the scan to be performed on the coefficients before encoding, takes one of the following values:

IPPVC\_SCAN\_ZIGZAG, indicating the classical zigzag scan,  
 IPPVC\_SCAN\_HORIZONTAL, indicating the alternate-horizontal scan,  
 IPPVC\_SCAN\_VERTICAL, indicating the alternate-vertical scan,  
 IPPVC\_SCAN\_NONE, indicating that no scan is to be performed,  
 that is, the input coefficients are already in  
 the scan order.

See the corresponding [enumerator](#) in the introduction to the General Functions section.

## Description

The function `ippiEncodeCoeffsIntra_H263_16s1u` is declared in the `ippvc.h` header file. This function performs VLC encoding of the quantized AC coefficients in a scan order for one intra coded block and puts the codes into the bitstream. Intra AC VLC encoding process is specified in [ITUH263], subclause 5.4.2, and is modified as specified in [ITUH263] Annex T, clause T.4, when Modified Quantization mode is in use. When in Advanced Intra Coding mode, VLC Table I.2 from [ITUH263] Annex I is used for all intra DC and intra AC coefficients, otherwise Table 16 [ITUH263] is used to encode AC coefficients (starting from `pQCoef[1]`) only. In this case `countNonZero` is the number of non-zero AC coefficients.

This function is used in the H.263 and MPEG-4 encoders included into Intel IPP Samples. See [introduction](#) to the H.263 section.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if at least one of the specified pointers is NULL.
<code>ippStsBitOffsetErr</code>	Indicates an error condition if <code>*pBitOffset</code> is out of the range [0, 7].
<code>ippStsOutOfRangeErr</code>	Indicates an error condition if <code>countNonZero</code> is out of the range [1, 64].

## EncodeCoeffsInter\_H263

*Encodes and puts quantized DCT coefficients for inter coded block into bitstream.*

---

## Syntax

```
IppStatus ippiEncodeCoeffsInter_H263_16s1u(Ipp16s* pQCoef, Ipp8u** ppBitStream, int* pBitOffset, int countNonZero, int modQuantFlag, int scan);
```

## Parameters

<i>pQCoef</i>	Pointer to the array of quantized DCT coefficients. <code>pQCoef[0]</code> is the DC coefficient.
<i>ppBitStream</i>	Pointer to pointer to the current byte in the bitstream buffer. <code>**ppBitStream</code> is updated by the function.

<code>pBitOffset</code>	Pointer to the bit position in the byte pointed by <code>**ppBitStream</code> . Valid within the range 0 to 7. <code>*pBitOffset</code> is updated by the function.
<code>countNonZero</code>	Number of non-zero coefficients in the block. Valid within the range 1 to 64.
<code>modQuantFlag</code>	Flag equal to a non-zero value when Modified Quantization mode is in use, equal to 0 otherwise.
<code>scan</code>	Type of the scan to be performed on the coefficients before encoding, takes one of the following values:  <div style="display: flex; justify-content: space-between;"> <div> <code>IPPVC_SCAN_ZIGZAG,</code>  <code>IPPVC_SCAN_NONE,</code> </div> <div> indicating the classical zigzag scan,  indicating that no scan is to be performed,  that is, the input coefficients are already in  the scan order. </div> </div> See the corresponding <a href="#">enumerator</a> in the introduction to the General Functions section.

## Description

The function `ippiEncodeCoeffsInter_H263_16s1u` is declared in the `ippvc.h` header file. This function performs VLC encoding of the quantized DCT coefficients in a scan order for one inter coded block and puts the codes into the bitstream. Inter DCT VLC encoding process is specified in [ITUH263], subclause 5.4.2 (Table 16), and is modified as specified in [ITUH263] Annex T, clause T.4, when Modified Quantization mode is in use.

This function is used in the H.263 and MPEG-4 encoders included into Intel IPP Samples. See [introduction](#) to the H.263 section.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if at least one of the specified pointers is NULL.
<code>ippStsBitOffsetErr</code>	Indicates an error condition if <code>*pBitOffset</code> is out of the range [0, 7].
<code>ippStsOutOfRangeErr</code>	Indicates an error condition if <code>countNonZero</code> is out of the range [1, 64].

## Quantization

### QuantIntra\_H263

*Performs quantization on an intra coded block.*

## Syntax

```
ippStatus ippiQuantIntra_H263_16s_C1I(Ipp16s* pSrcDst, int QP, int* pCountNonZero, int
advIntraFlag, int modQuantFlag);
```

## Parameters

<code>pSrcDst</code>	Pointer to the coefficient buffer of the block.
<code>QP</code>	Quantization parameter.

<i>pCountNonZero</i>	Pointer to the number of non-zero coefficients after quantization.
<i>advIntraFlag</i>	Flag equal to a non-zero value when Advanced Intra Coding mode is in use, equal to 0 otherwise.
<i>modQuantFlag</i>	Flag equal to a non-zero value when Modified Quantization mode is in use, equal to 0 otherwise.

## Description

The function `ippiQuantIntra_H263_16s_C1I` is declared in the `ippvc.h` header file. This function performs quantization on an intra coded block according to H.263 standard. The standard specifies dequantization process, while quantization decision levels are not defined. When not in Advanced Intra Coding mode, the intra DC coefficient is dequantized using uniformly placed reconstruction levels with a step size of 8, and the other DCT coefficients are reconstructed using equally spaced levels with a central dead-zone around zero and with a step size of  $2*QP$  ([ITUH263], subclauses 4.2.4, 6.2). When in Advanced Intra Coding mode, all the block coefficients are dequantized using a reconstruction spacing without a dead-zone and with a step size of  $2*QP$  ([ITUH263] Annex I, clause I.3). When not in Modified Quantization mode, the quantized intra DC coefficient (when not in Advanced Intra Coding mode) is clipped to the range [1, 254], and the other quantized coefficients (all coefficients, if in Advanced Intra Coding mode) are clipped to the range [-127, 127].

This function is used in the H.261, H.263, and MPEG-4 encoders included into Intel IPP Samples. See [introduction](#) to the H.263 section.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if at least one of the specified pointers is NULL.
<code>ippStsQPErr</code>	Indicates an error condition if $QP$ is out of the range [1, 31].

## QuantInter\_H263

*Performs quantization on an inter coded block.*

---

## Syntax

```
ippStatus ippiQuantInter_H263_16s_C1I(Ipp16s* pSrcDst, int QP, int* pCountNonZero, int modQuantFlag);
```

## Parameters

<i>pSrcDst</i>	Pointer to the coefficient buffer of the block.
<i>QP</i>	Quantization parameter.
<i>pCountNonZero</i>	Pointer to the number of non-zero coefficients after quantization.
<i>modQuantFlag</i>	Flag equal to a non-zero value when Modified Quantization mode is in use, equal to 0 otherwise.

## Description

The function `ippiQuantInter_H263_16s_C1I` is declared in the `ippvc.h` header file. This function performs quantization on an inter coded block according to H.263 standard. The standard specifies dequantization process, while quantization decision levels are not defined. The DCT coefficients are reconstructed using equally spaced levels with a central dead-zone around zero and with a step size of  $2 \cdot QP$  ([ITUH263], subclauses 4.2.4, 6.2). When not in Modified Quantization mode, the quantized coefficients are clipped to the range  $[-127, 127]$ .

This function is used in the H.261, H.263, and MPEG-4 encoders included into Intel IPP Samples. See [introduction](#) to the H.263 section.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if at least one of the specified pointers is NULL.
<code>ippStsQPErr</code>	Indicates an error condition if $QP$ is out of the range $[1, 31]$ .

## Resampling

## DownsampleFour16x16\_H263

*Performs factor-of-4 downsampling on a 16x16 block.*

## Syntax

```
IPPStatus ippiDownsampleFour16x16_H263_16s_C1R(const Ipp16s* pSrc, int srcStep, Ipp16s* pDst, int dstStep);
```

## Parameters

<code>pSrc</code>	Pointer to the origin of the source 16x16 block.
<code>srcStep</code>	Width in bytes of the source image plane, that is, distance in bytes between the starts of consecutive lines of the source block.
<code>pDst</code>	Pointer to the origin of the destination 8x8 block.
<code>dstStep</code>	Width in bytes of the destination image plane.

## Description

The function `ippiDownsampleFour16x16_H263_16s_C1R` is declared in the `ippvc.h` header file. This function performs factor-of-4 downsampling of a 16x16 source block to an 8x8 destination block, which is used for block encoding in Reduced-Resolution Update mode specified in [ITUH263], Annex Q. The inverse factor-of-4 upsampling procedure, specified in [ITUH263], Annex Q, clause Q.6, is performed by the function [UpsampleFour8x8\\_H263](#).

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if at least one of the specified pointers is NULL.

## H.264

This section describes functions for decoding of video data in accordance with JVT-G050 ([JVTG050]) and ITUH264 ([ITUH264]) standards.

The use of some functions described in this section is demonstrated in Intel® IPP Samples downloadable from <http://www.intel.com/cd/software/products/asmo-na/eng/220046.htm>.

The following enumeration is used to indicate prediction modes of the Intra\_4x4 prediction process for luma samples (8.3.1 of JVT-G050):

```
typedef enum {
    IPP_4x4_VERT      = 0,
    IPP_4x4_HOR       = 1,
    IPP_4x4_DC        = 2,
    IPP_4x4_DIAG_DL   = 3,
    IPP_4x4_DIAG_DR   = 4,
    IPP_4x4_VR        = 5,
    IPP_4x4_HD        = 6,
    IPP_4x4_VL        = 7,
    IPP_4x4_HU        = 8,
} IppIntra4x4PredMode_H264;
```

The table below shows the correspondence between prediction modes and constants of enum IppIntra4x4PredMode\_H264.

Name of Constant	Prediction Mode	Chapter in JVT-G050
IPP_4x4_VERT	Intra_4x4_Vertical	8.3.1.2.1
IPP_4x4_HOR	Intra_4x4_Horizontal	8.3.1.2.2
IPP_4x4_DC	Intra_4x4_DC	8.3.1.2.3
IPP_4x4_DIAG_DL	Intra_4x4_Diagonal_Down_Left	8.3.1.2.4
IPP_4x4_DIAG_DR	Intra_4x4_Diagonal_Down_Right	8.3.1.2.5
IPP_4x4_VR	Intra_4x4_Vertical_Right	8.3.1.2.6
IPP_4x4_HD	Intra_4x4_Horizontal_Down	8.3.1.2.7
IPP_4x4_VL	Intra_4x4_Vertical_Left	8.3.1.2.8
IPP_4x4_HU	Intra_4x4_Horizontal_Up	8.3.1.2.9

The following enumeration is used to indicate prediction modes of the Intra\_16x16 prediction process for luma samples (8.3.2 of JVT-G050):

```
typedef enum {
    IPP_16X16_VERT      = 0,
    IPP_16X16_HOR       = 1,
    IPP_16X16_DC        = 2,
    IPP_16X16_PLANE     = 3,
} IppIntra16x16PredMode_H264;
```

The table below shows the correspondence between prediction modes and constants of enum IppIntra16x16PredMode\_H264.

Name of Constant	Prediction Mode	Chapter in JVT-G050
IPP_16X16_VERT	Intra_16x16_Vertical	8.3.2.1
IPP_16X16_HOR	Intra_16x16_Horizontal	8.3.2.2
IPP_16X16_DC	Intra_16x16_DC	8.3.2.3
IPP_16X16_PLANE	Intra_16x16_Plane	8.3.2.4

The following enumeration is used to indicate prediction modes of the intra prediction process for chroma samples (8.3.3 of JVT-G050):

```
typedef enum {
    IPP_CHROMA_DC      = 0,
    IPP_CHROMA_HOR     = 1,
    IPP_CHROMA_VERT    = 2,
    IPP_CHROMA_PLANE   = 3,
} IppIntraChromaPredMode_H264;
```

The table below shows the correspondence between prediction modes and constants of enum IppIntraChromaPredMode\_H264.

Name of Constant	Prediction Mode	Chapter in JVT-G050
IPP_CHROMA_DC	Intra_Chroma_DC	8.3.3.1
IPP_CHROMA_HOR	Intra_Chroma_Horizontal	8.3.3.2
IPP_CHROMA_VERT	Intra_Chroma_Vertical	8.3.3.3
IPP_CHROMA_PLANE	Intra_Chroma_Plane	8.3.3.4

The following enumeration is used to indicate availability of the corresponding positions for prediction in the picture:

```
typedef enum {
    IPP_UPPER      = 1,
    IPP_LEFT       = 2,
    IPP_CENTER     = 4,
    IPP_RIGHT      = 8,
    IPP_LOWER      = 16,
    IPP_UPPER_LEFT = 32,
    IPP_UPPER_RIGHT = 64,
    IPP_LOWER_LEFT = 128,
    IPP_LOWER_RIGHT = 256,
} IppLayoutFlag;
```

The following enumeration is used to indicate image type of the picture:

```
typedef enum _IPPVC_FRAME_FIELD_FLAG
{
    IPPVC_FRAME      = 0x0,
    IPPVC_TOP_FIELD  = 0x1,
    IPPVC_BOTTOM_FIELD = 0x2
} IPPVC_FRAME_FIELD_FLAG;
```

## H.264 Video Decoder Functions

Function Short Name	Description
<b>CAVLC Parsing</b>	
<a href="#">DecodeCAVLCCoeffs_H264</a>	Decodes any non-chroma DC coefficients CAVLC coded.
<a href="#">DecodeCAVLCChromaDcCoeffs_H264</a> , <a href="#">DecodeCAVLCChroma422DcCoeffs_H264</a>	Decode chroma DC coefficients CAVLC coded.
<a href="#">DecodeExpGolombOne_H264</a>	Decodes one Exp-Golomb code accordance to 9.1 H.264 standard.
<b>Inverse Quantization and Inverse Transform</b>	

Function Short Name	Description
<a href="#">TransformDequantLumaDC_H264</a>	Performs integer inverse transformation and dequantization for 4x4 luma DC coefficients.
<a href="#">TransformDequantChromaDC_H264</a>	Performs integer inverse transformation and dequantization for 2x2 chroma DC coefficients.
<a href="#">TransformResidual4x4Inv_H264</a>	Performs integer inverse transformation for a 4x4 block of residuals.
<a href="#">DequantTransformResidual_H264</a>	Performs dequantization, integer inverse transformation, and shift for a 4x4 block of residuals.
<a href="#">DequantTransformResidualAndAdd_H264</a>	Performs dequantization, integer inverse transformation, shift for a 4x4 block of residuals with subsequent intra prediction or motion compensation.
<a href="#">TransformPrediction_H264</a>	Performs inverse transform of inter prediction samples for the current macroblock in decoding process for P macroblocks in SP slices or SI macroblocks.
<a href="#">DequantTransformResidual_SISP_H264</a>	Performs integer inverse transformation and dequantization of one block in P macroblocks in SP slices or SI macroblocks.
<a href="#">TransformDequantChromaDC_SISP_H264</a>	Performs integer inverse transformation and dequantization for 2x2 chroma DC coefficients in P macroblocks in SP slices or SI macroblocks.
<b>Intra Prediction</b>	
<a href="#">PredictIntra_4x4_H264</a>	Performs luma component prediction for intra 4x4 macroblock type.
<a href="#">PredictIntra_16x16_H264</a>	Performs luma component prediction for intra 16x16 macroblock type.
<a href="#">PredictIntraChroma8x8_H264</a>	Performs chroma component prediction for intra macroblock type.
<b>Inter Prediction</b>	
<a href="#">ExpandPlane_H264</a>	Expands the plane.
<a href="#">InterpolateLuma_H264</a>	Performs interpolation for motion estimation of the luma component using fractional part of motion vector.



Function Short Name	Description
<code>InterpolateLumaTop_H264</code>	Performs interpolation for motion estimation of the luma component at the frame top boundary.
<code>InterpolateLumaBottom_H264</code>	Performs interpolation for motion estimation of the luma component at the frame bottom boundary.
<code>InterpolateLumaBlock_H264</code>	Performs interpolation for motion estimation of the luma component using entire motion vector.
<code>InterpolateChroma_H264</code>	Performs interpolation for motion estimation of the chroma component using fractional part of motion vector.
<code>InterpolateChromaTop_H264</code>	Performs interpolation for motion estimation of the chroma component at the frame top boundary.
<code>InterpolateChromaBottom_H264</code>	Performs interpolation for motion estimation of the chroma component at the frame bottom boundary.
<code>InterpolateChromaBlock_H264</code>	Performs interpolation for motion estimation of the chroma component using entire motion vector.
<code>InterpolateBlock_H264, Bidir_H264</code>	Calculate the average value for each source pair values in block.
<code>WeightedAverage_H264</code>	Averages two blocks with weights.
<code>UnidirWeightBlock_H264, UnidirWeight_H264</code>	Weigh source block.
<code>BiDirWeightBlock_H264, BidirWeight_H264</code>	Average two blocks with two weights and two offsets.
<code>BiDirWeightBlockImplicit_H264, BidirWeightImplicit_H264</code>	Average two blocks with weights if <i>ulog2wd</i> is equal to 5.
<b>Macroblock Reconstruction</b>	
<code>ReconstructChromaInterMB_H264</code>	Reconstructs inter chroma macroblock.
<code>ReconstructChromaIntraHalvesMB_H264</code>	Reconstructs two halves of intra chroma macroblock.
<code>ReconstructChromaIntraMB_H264</code>	Reconstructs intra chroma macroblock.
<code>ReconstructChromaInter4x4MB_H264, ReconstructChromaInter4x4_H264High</code>	Reconstruct 4X4 inter chroma macroblock for high profile.

Function Short Name	Description
<code>ReconstructChroma422Inter4x4_H264High</code>	Reconstructs 4X4 inter chroma macroblock for 4:2:2 chroma mode.
<code>ReconstructChromaIntraHalves4x4MB_H264</code> , <code>ReconstructChromaIntraHalf4x4_H264High</code>	Reconstruct two independent halves of 4X4 intra chroma macroblock for high profile.
<code>ReconstructChromaIntra4x4MB_H264</code> , <code>ReconstructChromaIntra4x4_H264High</code>	Reconstruct 4X4 intra chroma macroblock for high profile.
<code>ReconstructChroma422IntraHalf4x4_H264High</code>	Reconstructs two independent halves of 4X4 intra chroma macroblock for 4:2:2 chroma mode.
<code>ReconstructChroma422Intra4x4_H264High</code>	Reconstructs 4X4 intra chroma macroblock for 4:2:2 chroma mode.
<code>ReconstructLumaInterMB_H264</code>	Reconstructs inter luma macroblock.
<code>ReconstructLumaIntraHalfMB_H264</code>	Reconstructs half of intra luma macroblock.
<code>ReconstructLumaIntraMB_H264</code>	Reconstructs intra luma macroblock.
<code>ReconstructLumaInter4x4MB_H264</code> , <code>ReconstructLumaInter4x4_H264High</code>	Reconstruct 4X4 inter luma macroblock for high profile.
<code>ReconstructLumaIntraHalf4x4MB_H264</code> , <code>ReconstructLumaIntraHalf4x4_H264High</code>	Reconstruct half of 4X4 intra luma macroblock for high profile.
<code>ReconstructLumaIntra4x4MB_H264</code> , <code>ReconstructLumaIntra4x4_H264High</code>	Reconstruct 4X4 intra luma macroblock for high profile.
<code>ReconstructLumaInter8x8MB_H264</code> , <code>ReconstructLumaInter8x8_H264High</code>	Reconstruct 8X8 inter luma macroblock for high profile.
<code>ReconstructLumaIntraHalf8x8MB_H264</code> , <code>ReconstructLumaIntraHalf8x8_H264High</code>	Reconstruct half of 8X8 intra luma macroblock for high profile.
<code>ReconstructLumaIntra8x8MB_H264</code> , <code>ReconstructLumaIntra8x8_H264High</code>	Reconstruct 8X8 intra luma macroblock for high profile.
<code>ReconstructLumaIntra16x16MB_H264</code>	Reconstructs intra 16X16 luma macroblock.
<code>ReconstructLumaIntra_16x16MB_H264</code> , <code>ReconstructLumaIntra16x16_H264High</code>	Reconstruct intra 16X16 luma macroblock for high profile.
<b>Deblocking Filtering</b>	
<code>FilterDeblockingLuma_VerEdge_H264</code>	Performs deblocking filtering on the vertical edges of a luma 16x16 macroblock.

Function Short Name	Description
<code>FilterDeblockingLuma_VerEdge_MBAFF_H264</code>	Performs deblocking filtering on the external vertical edges of half of a 16x16 luma macroblock.
<code>FilterDeblockingLuma_HorEdge_H264</code>	Performs deblocking filtering on the horizontal edges of a luma 16x16 macroblock.
<code>FilterDeblockingChroma_VerEdge_H264, FilterDeblockingChromaVerEdge_H264, FilterDeblockingChroma422VerEdge_H264</code>	Perform deblocking filtering on the vertical edges of a chroma macroblock.
<code>FilterDeblockingChroma_VerEdge_MBAFF_H264, FilterDeblockingChroma422VerEdgeMBAFF_H264</code>	Perform deblocking filtering on the external vertical edges of half of an 8x8 chroma macroblock.
<code>FilterDeblockingChroma_HorEdge_H264, FilterDeblockingChromaHorEdge_H264, FilterDeblockingChroma422HorEdge_H264</code>	Perform deblocking filtering on the horizontal edges of a chroma macroblock.

## H.264 Video Encoder Functions

Function Short Name	Description
<b>Edges Detection</b>	
<code>EdgesDetect16x16</code>	Detects edges inside a 16X16 block.
<b>Calculation of Inter Predicted Blocks</b>	
<code>InterpolateLuma_H264</code>	Performs interpolation for motion estimation of the luma component.
<code>InterpolateLumaTop_H264</code>	Performs interpolation for motion estimation of the luma component at the frame top boundary.
<code>InterpolateLumaBottom_H264</code>	Performs interpolation for motion estimation of the luma component at the frame bottom boundary.
<code>InterpolateChroma_H264</code>	Performs interpolation for motion estimation of the chroma component.
<code>InterpolateChromaTop_H264</code>	Performs interpolation for motion estimation of the chroma component at the frame top boundary.
<code>InterpolateChromaBottom_H264</code>	Performs interpolation for motion estimation of the chroma component at the frame bottom boundary.

Function Short Name	Description
<b>Calculation of Intra Predicted Blocks</b>	
<code>PredictIntra_4x4_H264</code>	Performs intra prediction for a 4x4 luma component.
<code>PredictIntra_16x16_H264</code>	Performs intra prediction for a 16x16 luma component.
<code>PredictIntraChroma8x8_H264</code>	Performs intra prediction for a 8x8 chroma component.
<b>Estimation of Inter and Intra Predicted Blocks</b>	
<code>SAD16x16</code>	Evaluates sum of absolute difference between current and reference 16X16 blocks.
<code>SAD16x8</code>	Evaluates sum of absolute difference between current and reference 16X8 blocks.
<code>SAD8x16</code>	Evaluates sum of absolute difference between current and reference 8X16 blocks.
<code>SAD8x8</code>	Evaluates sum of absolute difference between current and reference 8X8 blocks.
<code>SAD8x4</code>	Evaluates sum of absolute difference between current and reference 8X4 blocks.
<code>SAD4x8</code>	Evaluates sum of absolute difference between current and reference 4X8 blocks.
<code>SAD4x4</code>	Evaluates sum of absolute difference between current and reference 4X4 blocks.
<code>SAD16x16Blocks8x8</code>	Evaluates four partial sums of absolute differences between current and reference 16X16 blocks.
<code>SAD16x16Blocks4x4</code>	Evaluates 16 partial sums of absolute differences between current and reference 16X16 blocks.
<code>SATD16x16</code>	Evaluates sum of absolute transformed differences between current and reference 16X16 blocks using 4x4 transform.
<code>SATD16x8</code>	Evaluates sum of absolute transformed differences between current and reference 16X8 blocks using 4x4 transform.

Function Short Name	Description
<a href="#">SATD8x16</a>	Evaluates sum of absolute transformed differences between current and reference 8X16 blocks using 4x4 transform.
<a href="#">SATD8x8</a>	Evaluates sum of absolute transformed differences between current and reference 8X8 blocks using 4x4 transform.
<a href="#">SATD8x4</a>	Evaluates sum of absolute transformed differences between current and reference 8X4 blocks using 4x4 transform.
<a href="#">SATD4x8</a>	Evaluates sum of absolute transformed differences between current and reference 4X8 blocks using 4x4 transform.
<a href="#">SATD4x4</a>	Evaluates sum of absolute transformed differences between current and reference 4X4 blocks using 4x4 transform.
<a href="#">SAT8x8D</a>	Evaluates sum of absolute transformed differences between current and reference 8X8 blocks using 8x8 transform.
<b>Obtaining Residual and DC Blocks</b>	
<a href="#">GetDiff4x4</a>	Calculates a residual 4x4 block in the cases of intra 4x4 luma block or inter luma block.
<a href="#">SumsDiff16x16Blocks4x4</a>	Calculates a 4x4 DC luma block and 4x4 residual blocks in the case of an intra 16x16 luma block.
<a href="#">SumsDiff8x8Blocks4x4</a>	Calculates a 2x2 DC chroma block and residual 4x4 chroma blocks.
<b>Forward Transform and Quantization</b>	
<a href="#">TransformQuantChromaDC_H264</a>	Performs forward transform and quantization for 2x2 DC chroma blocks.
<a href="#">TransformQuantLumaDC_H264</a>	Performs forward transform and quantization for 4x4 DC luma blocks.
<a href="#">TransformResidual4x4Fwd_H264</a>	Performs forward transform for 4X4 residual blocks.
<a href="#">QuantizeResidual4x4Fwd_H264</a>	Performs forward quantization for 4X4 residual blocks.
<a href="#">TransformQuantResidual_H264</a>	Performs forward transform and quantization for 4x4 residual blocks.

Function Short Name	Description
<a href="#">TransformLuma8x8Fwd_H264</a>	Performs forward 8x8 transform for 8x8 luma blocks without normalization.
<a href="#">QuantLuma8x8_H264</a>	Performs quantization for 8x8 luma block coefficients including 8x8 transform normalization.
<a href="#">GenScaleLevel8x8_H264</a>	Generates ScaleLevel matrices for forward and inverse quantization including normalization for 8x8 forward and inverse transform.
<b>CAVLC Functions</b>	
<a href="#">EncodeCoeffsCAVLC_H264</a>	Calculates characteristics of 4X4 block for CAVLC encoding.
<a href="#">EncodeChromaDcCoeffsCAVLC_H264</a>	Calculates characteristics of 2X2 chroma DC block for CAVLC encoding.
<b>CABAC Functions</b>	
<a href="#">CABACGetSize_H264</a>	Returns the size required for the CABAC state.
<a href="#">CABACGetContexts_H264</a>	Copies contexts from the CABAC state structure.
<a href="#">CABACInit_H264</a>	Initializes CABAC state and aligns bitstream to byte boundary by writing 1s.
<a href="#">CABACInitAlloc_H264</a>	Allocates and initializes CABAC state and aligns bitstream to byte boundary by writing 1s.
<a href="#">CABACFree_H264</a>	Releases CABAC state, allocated by <code>ippi-CABACInitAlloc_H264</code> .
<a href="#">CABACSetStream_H264</a>	Sets a new pointer for output stream.
<a href="#">CABACGetStreamSize_H264</a>	Returns the number of bits in the bitstream.
<a href="#">CABACEncodeBin_H264</a>	Encodes one bin with CABAC and writes it to the stream.
<a href="#">CABACEncodeBinBypass_H264</a>	Encodes bin with CABAC using bypass encoding process and writes it to the stream.

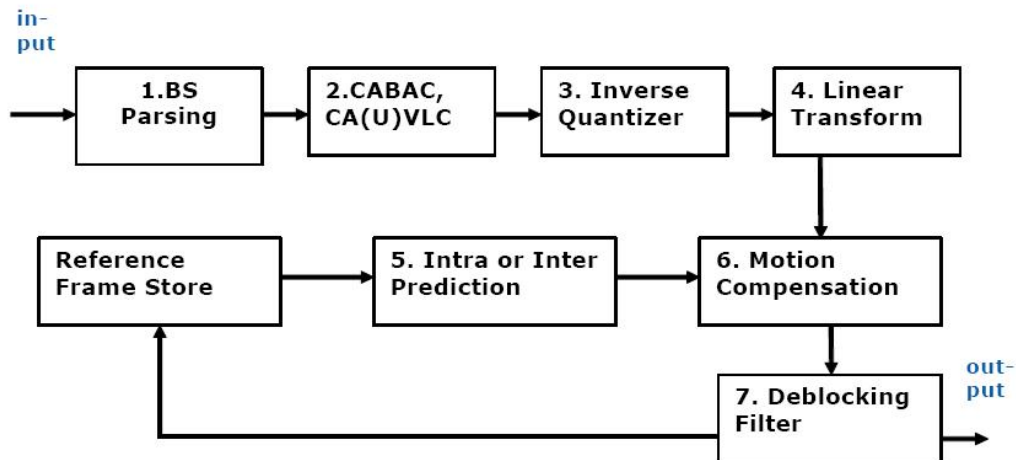
Function Short Name	Description
<code>CABACTerminateSlice_H264</code>	Encodes bin with CABAC using bypass encoding process and writes it to the stream.
<code>CABACEncodeResidualBlock_H264</code>	Encodes bin with CABAC using bypass encoding process and writes it to the stream.
<b>Inverse Quantization and Transform</b>	
<code>TransformDequantLumaDC_H264</code>	Performs integer inverse transformation and dequantization for 4x4 luma DC coefficients.
<code>TransformDequantChromaDC_H264</code>	Performs integer inverse transformation and dequantization for 2x2 chroma DC coefficients.
<code>DequantTransformResidualAndAdd_H264</code>	Performs dequantization, integer inverse transformation, shift for a 4x4 block of residuals with subsequent intra prediction or motion compensation.
<code>QuantLuma8x8Inv_H264</code>	Performs inverse quantization for 8x8 luma block coefficients including normalization of the following inverse 8x8 transform.
<code>TransformLuma8x8InvAddPred_H264</code>	Performs inverse 8x8 transform of 8x8 luma block coefficients with subsequent intra prediction or motion compensation.
<b>Deblocking</b>	
<code>FilterDeblockingLuma_VerEdge_H264</code>	Performs deblocking filtering on the vertical edges of the luma 16x16 macroblock.
<code>FilterDeblockingLuma_VerEdge_MB AFF_H264</code>	Performs deblocking filtering on the vertical edges of half of the luma 16x16 macroblock.
<code>FilterDeblockingLuma_HorEdge_H264</code>	Performs deblocking filtering on the horizontal edges of the luma 16x16 macroblock.
<code>FilterDeblockingChroma_VerEdge_H264, FilterDeblockingChroma422VerEdge_H264</code>	Performs deblocking filtering on the vertical edges of the chroma 8x8 macroblock.
<code>FilterDeblockingChroma_VerEdge_MB AFF_H264, FilterDeblockingChroma422VerEdgeMB AFF_H264</code>	Performs deblocking filtering on the vertical edges of half of the chroma 8x8 macroblock.

Function Short Name	Description
<code>FilterDeblockingChroma_HorEdge_H264</code> , <code>FilterDeblockingChroma422HorEdge_H264</code>	Performs deblocking filtering on the horizontal edges of the chroma 8x8 macroblock.

## H.264 Decoder Functions

This section describes the functions for H.264 Decoder in accordance with JVT-G050 ([JVTG050]) standard.

### H.264 Decoder Structure

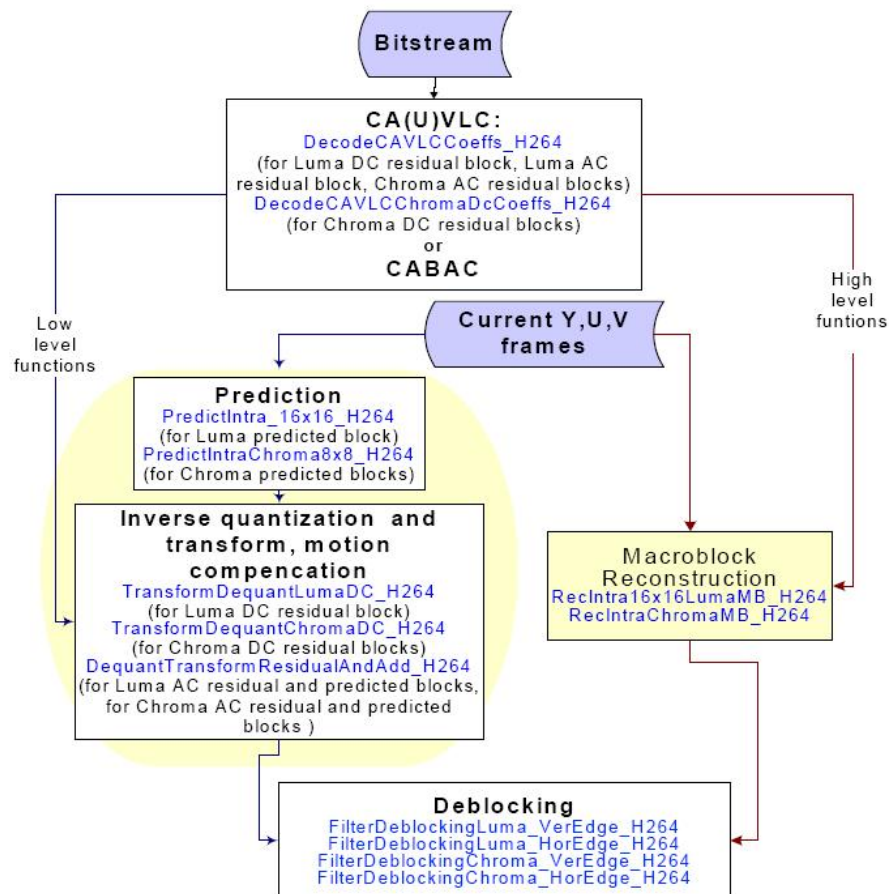


**NOTE.** This Motion Compensation process is performed by general motion compensation functions (See [General Functions](#)), which sum the predictor data and the residual and stores the result into the destination picture.



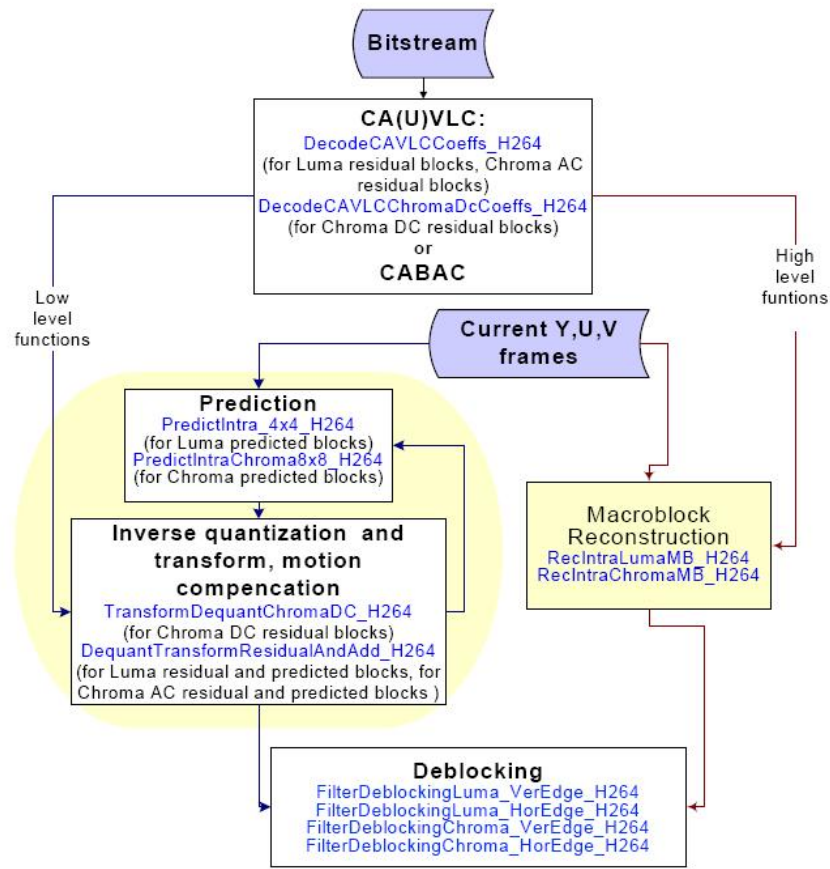
The following schemes show macroblock decoding with the help of video coding functions. The schemes for Intra16x16, intra 4x4, and P or B macroblock decoding show two paths of video coding functions using low and high level functions respectively. The blocks marked with yellow background color are interchangeable.

### Intra 16x16 Macroblock Decoding



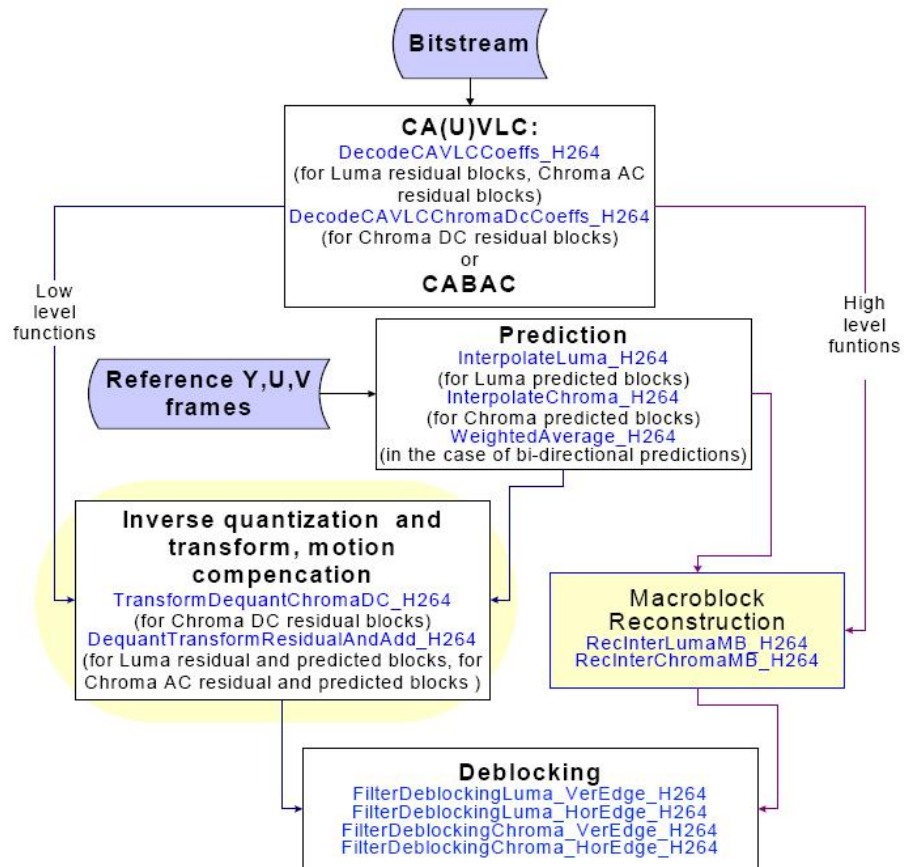
The following graph shows decoding process for I macroblock type in the case of Intra\_4x4 prediction mode.

### Intra 4x4 Macroblock Decoding



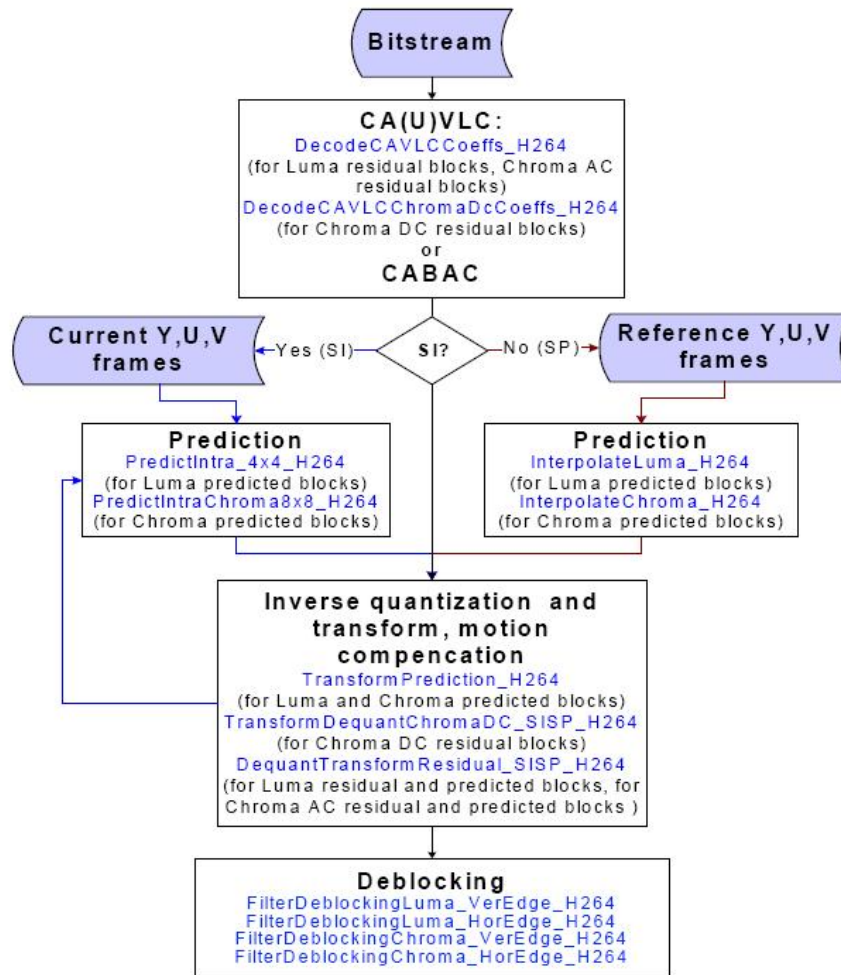
The following graph shows decoding process for P or B macroblock type.

### P or B Macroblock Decoding



The following graph shows decoding process for SI or SP macroblock type. Note that SI is an intra macroblock and SP is a predicted macroblock but their decoding schemes are similar.

### SI and SP Macroblocks Decoding



## CAVLC Parsing

### DecodeCAVLCCoeffs\_H264

Decodes any non-chroma DC coefficients CAVLC coded.

#### Syntax

```
IppStatus ippiDecodeCAVLCCoeffs_H264_1u16s(Ipp32u** ppBitStream, Ipp32s* pBitOffset,
Ipp16s* pNumCoeff, Ipp16s** ppDstCoeffs, const Ipp32u uVLCSelect, const Ipp16s
uMaxNumCoeff, const Ipp32s** ppTblCoeffToken, Ipp32s** ppTblTotalZeros, Ipp32s**
ppTblRunBefore, Ipp32s* pScanMatrix);

IppStatus ippiDecodeCAVLCCoeffs_H264_1u32s(Ipp32u** ppBitStream, Ipp32s* pBitOffset,
Ipp16s* pNumCoeff, Ipp32s** ppDstCoeffs, Ipp32u uVLCSelect, Ipp16s uMaxNumCoeff, const
Ipp32s** ppTblCoeffToken, const Ipp32s** ppTblTotalZeros, const Ipp32s** ppTblRunBefore,
const Ipp32s* pScanMatrix);
```

#### Parameters

<i>ppBitStream</i>	Double pointer to the current position in the bitstream. The pointer is updated by the function.
<i>pBitOffset</i>	Pointer to offset between the bit that <i>**ppBitStream</i> points to and the start of the code. The pointer is updated by the function.
<i>pNumCoeff</i>	Pointer to the output number of non-zero coefficients.
<i>ppDstCoeffs</i>	Double pointer to a 4x4 block of coefficients. These coefficients are calculated by the function. The function shifts pointer <i>**ppDstCoeffs</i> on 16.
<i>uVLCSelect</i>	Predictor on number of CoeffToken Table, which is designated in [JVTG050] as <i>nC</i> . It can be calculated in accordance with 9.2.1 of [JVTG050]
<i>uMaxNumCoeff</i>	Maximum coefficients in block (16 for intra 16x16, 15 for the rest).
<i>ppTblCoeffToken</i>	Double pointer to CoeffToken Tables.
<i>ppTblTotalZeros</i>	Double pointer to TotalZeros Tables.
<i>ppTblRunBefore</i>	Double pointer to RunBefore Tables.
<i>pScanMatrix</i>	Inverse scan matrix for coefficients in block.

#### Description

This function is declared in the `ippvc.h` header file. The functions `ippiDecodeCAVLCCoeffs_H264_1u16s` and `ippiDecodeCAVLCCoeffs_H264_1u32s` decode any non-chroma DC (chroma AC and Luma) coefficients CAVLC-coded in accordance with 9.2 of [JVTG050].

The table *ppTblCoeffToken* is an array of size 4. Each element of this array is a pointer to a table that contains codes, number of trailing one transform coefficient and total number of non-zero transform coefficients in accordance with Table 9-5 of [JVTG050].

- *ppTblCoeffToken*[0], where  $0 \leq uVLCSelect < 2$ ;

- `pTblCoeffToken[1]`, where  $2 \leq \text{uVLCSelect} < 4$ ;
- `pTblCoeffToken[2]`, where  $4 \leq \text{uVLCSelect} < 8$ ;
- `pTblCoeffToken[3]`, where  $\text{uVLCSelect} = -1$  (used only for [DecodeCAVLCChromaDcCoeffs\\_H264](#)) ;
- `pTblCoeffToken[4]`, where  $\text{uVLCSelect} = -2$  (used only for [DecodeCAVLCChroma422DcCoeffs\\_H264](#)) ;
- If  $\text{uVLCSelect} > 8$ , this function uses its own table in accordance with table 9-5 of [JVTG050],  $nC > 8$

Use function [HuffmanRunLevelTableInitAlloc](#) for creation of tables `pTblCoeffToken[i]`.

The table `ppTblTotalZeros` is an array of size 16. Each element of this array (except 0) is a pointer to a table that contains codes and values (`total_zeros` in [JVTG050]) in accordance with tables 9-7 and 9-8 of [JVTG050].

- `ppTblTotalZeros[0]` is not used.
- `ppTblTotalZeros[i]` contains codes and values that correspond to `TotalCoeff` (in [JVTG050]) =  $i$ ,  $0 < i < 16$ .

The table `ppTblRunBefore` is array of size 8. Each element of this array (except 0) is a pointer to a table that contains codes and values (`run_before` in [JVTG050]) in accordance with table 9-10 of [JVTG050].

- `ppTblRunBefore[0]` is not used.
- `ppTblRunBefore[i]` contains codes and values that correspond with `zerosLeft` (in [JVTG050]) =  $i$ ,  $0 < i < 7$ .
- `ppTblRunBefore[7]` contains codes and values that correspond with `zerosLeft` (in [JVTG050])  $> 6$ .

Use function [HuffmanTableInitAlloc](#) for creation of tables `PpTblTotalZeros[i]`, `PpTblRunBefore[i]`.



**WARNING.** For proper operation of the function, remove `emulation_prevention_three_byte` described in 7.4.1 of [JVTG050] from the stream that undergoes decoding.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when at least one input pointer is NULL.

## DecodeCAVLCChromaDcCoeffs\_H264, DecodeCAVLCChroma422DcCoeffs\_H264

*Decode chroma DC coefficients CAVLC coded.*

### Syntax

```
IppStatus ippiDecodeCAVLCChromaDcCoeffs_H264_1u16s(Ipp32u** ppBitStream, Ipp32s* pBitOffset,
Ipp16s* pNumCoeff, Ipp16s** ppDstCoeffs, const Ipp32s* pTblCoeffToken, const Ipp32s**
ppTblTotalZerosCR, const Ipp32s** ppTblRunBefore);
```

```
IppStatus ippiDecodeCAVLCChromaDcCoeffs_H264_1u32s(Ipp32u** ppBitStream, Ipp32s* pBitOffset,
Ipp16s* pNumCoeff, Ipp32s** ppDstCoeffs, const Ipp32s* pTblCoeffToken, const Ipp32s**
ppTblTotalZerosCR, const Ipp32s** ppTblRunBefore);
```

```
IppStatus ippiDecodeCAVLCChroma422DcCoeffs_H264_1u16s(Ipp32u** ppBitStream, Ipp32s*
pBitOffset, Ipp16s* pNumCoeff, Ipp16s** ppDstCoeffs, const Ipp32s* pTblCoeffToken, const
Ipp32s** ppTblTotalZerosCR, const Ipp32s** ppTblRunBefore);
```

```
IppStatus ippiDecodeCAVLCChroma422DcCoeffs_H264_1u32s(Ipp32u** ppBitStream, Ipp32s*
pBitOffset, Ipp16s* pNumCoeff, Ipp32s** ppDstCoeffs, const Ipp32s* pTblCoeffToken, const
Ipp32s** ppTblTotalZerosCR, const Ipp32s** ppTblRunBefore);
```

## Parameters

<i>ppBitStream</i>	Double pointer to the current position in the bitstream. The pointer is updated by the function.
<i>pBitOffset</i>	Pointer to offset between the bit that <i>ppBitStream</i> points to and the start of the code. The pointer is updated by the function.
<i>pNumCoeff</i>	Pointer to the output number of coefficients.
<i>ppDstCoeffs</i>	Double pointer to a 2x2 or 2x4 (for chroma 422) block of coefficients. These coefficients are calculated by the function. The function shifts pointer <i>ppDstCoeffs</i> by 4 or 8 (for chroma 422).
<i>pTblCoeffToken</i>	Pointer to chroma DC CoeffToken Table.
<i>ppTblTotalZerosCR</i>	Double pointer to chroma DC TotalZeros Tables.
<i>ppTblRunBefore</i>	Double pointer to RunBefore Tables.

## Description

These functions are declared in the `ippvc.h` header file. The functions `ippiDecodeCAVLCChromaDcCoeffs_H264_1u16s` and `ippiDecodeCAVLCChromaDcCoeffs_H264_1u32s` decode chroma DC coefficients and `ippiDecodeCAVLCChroma422DcCoeffs_H264_1u16s` and `ippiDecodeCAVLCChroma422DcCoeffs_H264_1u32s` decode Chroma422 (4:2:2 chroma mode) DC coefficients, CAVLC-coded in accordance with 9.2 of [JVTG050] in the case of  $nC = -1$  and  $nC = -2$  (in Chroma422 case).

*pTblCoeffToken* is equal to *ppTblCoeffToken*[3] or *ppTblCoeffToken*[4] for chroma422. *ppTblCoeffToken* and *ppTblRunBefore* tables are defined in the same way as in `DecodeCAVLCCoeffs_H264` function.

The table *ppTblTotalZerosCR* is an array of size 4 (size 8 for Chroma422). Each element of this array is a pointer to a table that contains codes and values (*total\_zeros* in [JVTG050]) in accordance with tables 9-9 of [JVTG050].

- *ppTblTotalZerosCR*[0] is not used.
- *ppTblTotalZerosCR*[i] contains codes and values that correspond to *TotalCoeff* (in [JVTG050]) = i,  $0 < i < 4$  ( $0 < i < 8$  for Chroma422).



**WARNING.** For proper operation of the function, remove `emulation_prevention_three_byte` described in 7.4.1 of [JVTG050] from the stream that undergoes decoding.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when at least one input pointer is NULL.

## DecodeExpGolombOne\_H264

*Decodes one Exp-Golomb code according to 9.1 of H.264 standard.*

---

### Syntax

```
IppStatus ippiDecodeExpGolombOne_H264_1u16s (Ipp32u** ppBitStream, Ipp32s* pBitOffset,
Ipp16s* pDst, Ipp8u isSigned);
```

### Parameters

<i>ppBitStream</i>	Double pointer to the current position in the bitstream.
<i>pBitOffset</i>	Pointer to the offset between the bit pointed by <i>pBitStream</i> and the start of the code.
<i>pDst</i>	Pointer to the destination result.
<i>isSigned</i>	Flag that indicates if output value has to be decoded as signed.

### Description

The function `ippiDecodeExpGolombOne_H264_1u16s` is declared in the `ippvc.h` header file. The function decodes one Exp-Golomb code in accordance with 9.1 of the H.264 standard.

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when at least one input pointer is <code>NULL</code> .

## Inverse Quantization and Inverse Transform

## TransformDequantLumaDC\_H264

*Performs integer inverse transformation and dequantization for 4x4 luma DC coefficients.*

---

### Syntax

```
IppStatus ippiTransformDequantLumaDC_H264_16s_C1I(Ipp16s* pSrcDst, Ipp32s QP);
```

### Parameters

<i>pSrcDst</i>	Pointer to the initial coefficients and resultant DC (4x4 block).
<i>QP</i>	Quantization parameter.

### Description

The function `ippiTransformDequantLumaDC_H264_16s_C1I` is declared in the `ippvc.h` file. This function performs integer inverse transformation and dequantization for 4x4 luma DC coefficients in accordance with 8.5.6 of [JVTG050].

The argument *QP* stands here for a luma quantization parameter, which is designated in JVT-G050 as  $QP_Y$ .



## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if at least one of the specified pointers is NULL.
<code>ippStsOutOfRangeErr</code>	Indicates an error condition if $QP$ is less than 1 or greater than 51.

## TransformDequantChromaDC\_H264

*Performs integer inverse transformation and dequantization for 2x2 chroma DC coefficients.*

### Syntax

```
IppStatus ippTransformDequantChromaDC_H264_16s_C1I(Ipp16s* pSrcDst, Ipp32s QP);
```

### Parameters

<code>pSrcDst</code>	Pointer to the initial coefficients and resultant chroma DC (2x2 block).
<code>QP</code>	Quantization parameter.

### Description

The function `ippTransformDequantChromaDC_H264_16s_C1I` is declared in the `ippvc.h` file. This function performs integer inverse transformation and dequantization for 2x2 chroma DC coefficients in accordance with 8.5.7 of [JVTG050].

The argument  $QP$  stands here for a chroma quantization parameter, which is designated in JVT-G050 as  $Q_{PC}$ .

This function is used in the H.264 encoder included into Intel IPP Samples. See [introduction](#) to the H.264 section.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if at least one of the specified pointers is NULL.
<code>ippStsOutOfRangeErr</code>	Indicates an error condition if $QP$ is less than 1 or greater than 51.

## TransformResidual4x4Inv\_H264

*Performs integer inverse transformation for a 4x4 block of residuals.*

### Syntax

```
IppStatus ippTransformResidual4x4Inv_H264_16s_C1(const Ipp16s* pSrc, const Ipp16s* pDst);
IppStatus ippTransformResidual4x4Inv_H264_32s_C1(const Ipp32s* pSrc, const Ipp32s* pDst);
```

## Parameters

<i>pSrc</i>	Pointer to the initial residual coefficients 4x4 block (AC luma or AC chroma or luma) - source array of size 16.
<i>pDst</i>	Pointer to the resulting transformed 4x4 block - destination array of size 16.

## Description

The functions `ippiTransformResidual4x4Inv_H264_16s_C1` and `ippiTransformResidual4x4Inv_H264_32s_C1` are declared in the `ippvc.h` file and perform inverse transformation for a residual 4x4 block described in 8.5.8 of [JVTG050].

For transformation of a chroma block these functions are called after `TransformDequantChromaDC_H264`. In the case of the 16x16 intra prediction mode, these functions should be called after `TransformDequantLumaDC_H264` for transforming a luma block.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if at least one of the specified pointers is NULL.

## DequantTransformResidual\_H264

*Performs dequantization, integer inverse transformation, and shift for a 4x4 block of residuals.*

---

## Syntax

```
IppStatus ippiDequantTransformResidual_H264_16s_C1I(Ipp16s* pSrcDst, Ipp32s step, Ipp16s* pDC, Ipp32s AC, Ipp32s QP);
```

## Parameters

<i>pSrcDst</i>	Pointer to the initial residual coefficients 4x4 block (AC luma or AC chroma or luma) and resultant transformed 4x4 block - source&destination array of size 16.				
<i>step</i>	Distance in bytes between starts of the consecutive lines in the source/destination plane.				
<i>pDC</i>	Pointer to the DC coefficient. If the case of luma block (not intra 16x16 macroblock type), <i>pDC</i> is set to NULL.				
<i>AC</i>	Flag that is not equal to zero, if at least one AC coefficient exists, and is equal to zero otherwise. In the case of luma block, which is not intra 16x16 macroblock type, AC should be: <table data-bbox="506 1709 1347 1841"> <tr> <td>0</td><td>if only <i>pSrcDst</i>[0] is not equal to 0 and other components are equal to 0</td></tr> <tr> <td>1</td><td>if any <i>pSrcDst</i>[<i>i</i>] is non-zero (<i>i</i> is within range [1, 15])</td></tr> </table>	0	if only <i>pSrcDst</i> [0] is not equal to 0 and other components are equal to 0	1	if any <i>pSrcDst</i> [ <i>i</i> ] is non-zero ( <i>i</i> is within range [1, 15])
0	if only <i>pSrcDst</i> [0] is not equal to 0 and other components are equal to 0				
1	if any <i>pSrcDst</i> [ <i>i</i> ] is non-zero ( <i>i</i> is within range [1, 15])				

$QP$ 

Quantization parameter for luma or for chroma.

## Description

The function `ippiDequantTransformResidual_H264_16s_C1I` is declared in the `ippvc.h` file and performs scaling, inverse transformation, and shift for a residual 4x4 block described in 8.5.8 of [JVTG050].

For transformation of a chroma block this function is called after `TransformDequantChromaDC_H264`. In the case of the 16x16 intra prediction mode, this function should be called after `TransformDequantLumaDC_H264` for transforming a luma block.

If  $pDC$  is not equal to NULL, the function, before starting the scaling and transformation process, places the DC coefficient specified by  $pDC$  in the top left corner of the source/destination array, which is designated as  $c_{00}$  in JVT-G050. Also during the scaling and transformation process the formula 8-266 ([JVTG050]) is used when calculating  $w_{00}$ .

If the pointer  $pDC$  is NULL, DC coefficient is dequantized like all the other coefficients. During the scaling and transformation process the formula 8-267 ([JVTG050]) is used when calculating  $w_{00}$ .

When calling the function, the argument  $QP$ , which is designated as  $qP$  in JVT-G050, should be calculated from the formulas 8-262 through 8-265 ([JVTG050]).

If  $AC$  is not equal to zero, formulas 8-278 through 8-277 ([JVTG050]) are applied to perform Inverse Transformations.

If  $AC = 0$ , all the coefficients are set equal to DC. The final shift is specified by formula 8-278.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if at least one of the specified pointers is NULL.
<code>ippStsOutOfRangeErr</code>	Indicates an error condition if $QP$ is less than 1 or greater than 51.
<code>ippStsStepErr</code>	Indicates an error condition if $srcStep$ value is less than 8.

## DequantTransformResidualAndAdd\_H264

*Performs dequantization, integer inverse transformation, shift for a 4x4 block of residuals with subsequent intra prediction or motion compensation.*

## Syntax

```
ippStatus ippiDequantTransformResidualAndAdd_H264_16s_C1I(const Ipp8u* pPred, Ipp16s*
pSrcDst, Ipp16s* pDC, Ipp8u* pDst, Ipp32s PredStep, Ipp32s DstStep, Ipp32s QP, Ipp32s
AC);
```

## Parameters

$pPred$	Pointer to the reference 4x4 block, which is used for intra prediction or motion compensation.
---------	--

<i>pSrcDst</i>	If not equal to 0, pointer to the initial residual 4x4 block (AC luma or AC chroma or luma) and resultant transformed 4x4 block - array of size 16. If all coefficients of the residual AC block are equal to 0, <i>pSrcDst</i> should be 0 (for chroma block or luma block of 16x16 intra macroblock type).				
<i>pDC</i>	Pointer to the DC coefficient. If luma block is not intra 16x16 macroblock type, <i>pDC</i> is set to NULL.				
<i>pDst</i>	Pointer to the destination 4x4 block.				
<i>PredStep</i>	Distance in bytes between starts of the consecutive lines in the reference frame.				
<i>DstStep</i>	Distance in bytes between starts of the consecutive lines in the destination frame.				
<i>QP</i>	Quantization parameter for luma and for chroma.				
<i>AC</i>	Flag that is not equal to zero, if at least one AC coefficient exists, and is equal to zero otherwise. In the case of luma block, which is not intra 16x16 macroblock type, AC should be: <table> <tr> <td>0</td><td>if only <i>pSrcDst</i>[0] is not equal to 0 and other components are equal to 0</td></tr> <tr> <td>1</td><td>if any <i>pSrcDst</i>[<i>i</i>] is non-zero (<i>i</i> is within range [1, 15]).</td></tr> </table>	0	if only <i>pSrcDst</i> [0] is not equal to 0 and other components are equal to 0	1	if any <i>pSrcDst</i> [ <i>i</i> ] is non-zero ( <i>i</i> is within range [1, 15]).
0	if only <i>pSrcDst</i> [0] is not equal to 0 and other components are equal to 0				
1	if any <i>pSrcDst</i> [ <i>i</i> ] is non-zero ( <i>i</i> is within range [1, 15]).				

## Description

The function `ippiDequantTransformResidualAndAdd_H264_16s_C1I` is declared in the `ippvc.h` file and performs scaling, transformation, and shift for a residual 4x4 block described in 8.5.8 of [JVTG050] with subsequent intra prediction or motion compensation (the formula 8-245 of [JVTG050]).

For transformation of a chroma block this function is called after `TransformDequantChromaDC_H264`. In the case of the 16x16 intra prediction mode, this function should be called after `TransformDequantLumaDC_H264` for transforming a luma block.

If *pDC* is not equal to NULL, the function, before starting the scaling and transformation process, places the DC coefficient specified by *pDC* in the top left corner of the initial coefficients block, which is designated as  $c_{00}$  in JVT-G050. Also during the scaling and transformation process the formula 8-266 ([JVTG050]) is used when calculating  $w_{00}$ .

If the pointer *pDC* is NULL, DC coefficient is dequantized like all the other coefficients. During the scaling and transformation process the formula 8-267 ([JVTG050]) is used when calculating  $w_{00}$ .

When calling the function, the argument *QP*, which is designated as  $qP$  in JVT-G050, should be calculated from the formulas 8-262 through 8-265 ([JVTG050]).

If *AC* is not equal to zero, formulas 8-278 through 8-277 ([JVTG050]) are applied to perform Inverse Transformations.

If  $AC = 0$ , all the coefficients are set equal to DC. The final shift is specified by formula 8-278.

This function is used in the H.264 encoder included into Intel IPP Samples. See [introduction](#) to the H.264 section.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if at least one of the specified pointers is NULL.
<code>ippStsOutOfRangeErr</code>	Indicates an error condition if $QP$ is less than 1 or greater than 51.

## TransformPrediction\_H264

*Performs inverse transform of inter prediction samples for the current macroblock in decoding process for P macroblocks in SP slices or SI macroblocks.*

### Syntax

```
ippStatus ippiTransformPrediction_H264_8u16s_C1(const Ipp8u* pSrc, Ipp32s step, Ipp16s* pDst);
```

### Parameters

<code>pSrc</code>	Pointer to the array of inter prediction samples for the current macroblock (4x4 block).
<code>step</code>	Distance in bytes between starts of the consecutive lines in the source buffer.
<code>pDst</code>	Destination array of size 16.

### Description

The function `ippiTransformPrediction_H264_8u16s_C1` is declared in the `ippvc.h` file. The function performs inverse transform of inter prediction samples for the current macroblock in decoding process for P macroblocks in SP slices or SI macroblocks (the formula 8-286 of [JVTG050]).

This function is used in the H.264 decoder included into Intel IPP Samples. See [introduction](#) to the H.264 section.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if at least one of the specified pointers is NULL.

## DequantTransformResidual\_SISP\_H264

*Performs integer inverse transformation and dequantization of one block in P macroblocks in SP slices or SI macroblocks.*

### Syntax

```
ippStatus ippiDequantTransformResidual_SISP_H264_16s_C1I(Ipp16s* pSrcDst, const Ipp16s* pPredictBlock, const Ipp16s* pDC, Ipp32s AC, Ipp32s qp, Ipp32s qs, Ipp32s Switch);
```

## Parameters

<i>pSrcDst</i>	Pointer to array (size 16) of the prediction residual transform coefficient (luma or chroma) - source/destination array.
<i>pPredictBlock</i>	Pointer to array (size 16) of the of inter prediction samples for the current macroblock after inverse transform. They can be calculated using <a href="#">TransformPrediction_H264</a> function.
<i>pDC</i>	Pointer to the DC coefficient. In the case of luma block type <i>pDC</i> must be 0. In the case of chroma block <i>pDC</i> is pointer to array (size 4) of ChromaDC after inverse transform. They can be calculated using <a href="#">TransformDequantChromaDC_SISP_H264</a> function.
<i>AC</i>	Flag that is not equal to zero, if at least one AC coefficient exists, and is equal to zero otherwise.
<i>qp</i>	Quantization parameter <i>qp</i> , which is used in the case of non-switching pictures.
<i>qs</i>	Quantization parameter <i>qs</i> .
<i>Switch</i>	Flag that is not equal to zero, if it switches pictures, and is equal to zero otherwise.

## Description

The function `ippiDequantTransformResidual_SISP_H264_16s_C1I` is declared in the `ippvc.h` file. The function performs integer inverse transformation and dequantization of one block in P macroblocks in SP slices or SI macroblocks (8-6 of [JVTG050]).

This function is used in the H.264 decoder included into Intel IPP Samples. See [introduction](#) to the H.264 section.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if at least one of the specified pointers is NULL.

## TransformDequantChromaDC\_SISP\_H264

*Performs integer inverse transformation and dequantization for 2x2 chroma DC coefficients in P macroblocks in SP slices or SI macroblocks.*

---

## Syntax

```
ippStatus ippiTransformDequantChromaDC_SISP_H264_16s_C1I(Ipp16s* pSrcDst, const Ipp16s* pDCPredict, Ipp32s qp, Ipp32s qs, Ipp32s Switch);
```

## Parameters

<i>pSrcDst</i>	Pointer to array (size 4) of the prediction residual transform chroma DC coefficient - source/destination array.
----------------	--

<i>pDCPredict</i>	Pointer to array (size 4) of the of inter prediction DC samples for the current macroblock after inverse transform. Inverse transform of the inter prediction samples can be calculated using <a href="#">TransformPrediction_H264</a> function for each 4x4 block.
<i>qp</i>	Quantization parameter ( <i>QP</i> . in [JVTG050]), which is used in the case of non-switching pictures.
<i>qs</i>	Quantization parameter ( <i>QS</i> . in [JVTG050]).
<i>Switch</i>	Flag that is not equal to zero, if it switches pictures, and is equal to zero otherwise.

## Description

The function `ippiTransformDequantChromaDC_SISP_H264_16s_C1I` is declared in the `ippvc.h` file. The function performs integer inverse transformation and dequantization for 2x2 chroma coefficients in P macroblocks in SP slices or SI macroblocks (8.6.1.2, 8.6.2.2 of [JVTG050]).

This function is used in the H.264 decoder included into Intel IPP Samples. See [introduction](#) to the H.264 section.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if at least one of the specified pointers is NULL.

## Intra Prediction

### PredictIntra\_4x4\_H264

Performs intra prediction for a 4x4 luma component.

## Syntax

```
ippStatus ippiPredictIntra_4x4_H264_8u_C1IR(Ipp8u* pSrcDst, Ipp32s srcdstStep,
IppIntra4x4PredMode predMode, Ipp32s availability);
```

## Parameters

<i>pSrcDst</i>	Pointer to the source and destination array.
<i>srcdstStep</i>	Distance in bytes between starts of the consecutive lines in the source and destination arrays. (Y-frame width in bytes).
<i>predMode</i>	Prediction mode of the <code>Intra_4x4</code> prediction process for luma samples.
<i>availability</i>	Flag that specifies availability of the samples used for prediction.

## Description

The function `ippiPredictIntra_4x4_H264_8u_C1IR` is declared in the `ippvc.h` file. This function performs intra prediction for a 4x4 luma component in accordance with 8.3.1.2 of [JVTG050] in a given prediction mode *predMode* (See enumeration `IppIntra4x4PredMode`).

The flag *availability* is computed as follows:

$B1*IPP\_LEFT + B2*IPP\_UPPER\_LEFT + B3*IPP\_UPPER + B4*IPP\_UPPER\_RIGHT$ ,

where the constants `IPP_LEFT`, `IPP_UPPER_LEFT`, `IPP_UPPER`, `IPP_UPPER_RIGHT` are from `IppLayoutFlag` and the variables `B1`, `B2`, `B3`, `B4` may take the values of

- 0 - if the block is unavailable for 4x4 prediction,
- 1 - if the block is available for 4x4 prediction.

### Layout of Blocks Used for Prediction

2- <code>IPP_UPPER_LEFT</code>	3- <code>IPP_UPPER</code>	4- <code>IPP_UPPER_RIGHT</code>
1- <code>IPP_LEFT</code>	<code>SrcDst</code>	

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if at least one of the specified pointers is NULL.
<code>ippStsStepErr</code>	Indicates an error condition if <code>srcdstStep</code> value is less than 4.
<code>ippStsOutOfRangeErr</code>	Indicates an error condition if <code>predMode</code> value falls outside [0,8].
<code>ippStsResFloor</code>	Indicates a warning condition if <code>predMode</code> is not allowed for this block.

### Example

The code example below shows how to use `PredictIntra_4x4_H264` function.

```
// declare block parameters structure
typedef struct H264Block4x4Info_t
{
    Ipp32s x;
    Ipp32s y;
    Ipp32s existNeighbour;
    Ipp32s absentNeighbour;
} H264Block4x4Info_t;

// define parameters for all 16 blocks of a macroblock
static
H264Block4x4Info_t H264Block4x4Info[16] =
{
    { 0, 0, 0, 0},
    { 4, 0, IPP_LEFT, 0},
    { 0, 4, IPP_UPPER | IPP_UPPER_RIGHT, 0},
    { 4, 4, IPP_UPPER | IPP_UPPER_LEFT, IPP_UPPER_RIGHT},

    { 8, 0, IPP_LEFT, 0},
    {12, 0, IPP_LEFT, 0},
    { 8, 4, IPP_UPPER | IPP_UPPER_LEFT | IPP_UPPER_RIGHT, 0},
    {12, 4, IPP_UPPER | IPP_UPPER_LEFT, IPP_UPPER_RIGHT},

    { 0, 8, IPP_UPPER | IPP_UPPER_RIGHT, 0},
```



```

    { 4, 8, IPP_UPPER | IPP_UPPER_LEFT | IPP_UPPER_RIGHT, 0},
    { 0, 12, IPP_UPPER | IPP_UPPER_RIGHT, 0},
    { 4, 12, IPP_UPPER | IPP_UPPER_LEFT, IPP_UPPER_RIGHT},

    { 8, 8, IPP_UPPER | IPP_UPPER_LEFT | IPP_UPPER_RIGHT, 0},
    { 12, 8, IPP_UPPER | IPP_UPPER_LEFT, IPP_UPPER_RIGHT},
    { 8, 12, IPP_UPPER | IPP_UPPER_LEFT | IPP_UPPER_RIGHT, 0},
    { 12, 12, IPP_UPPER | IPP_UPPER_LEFT, IPP_UPPER_RIGHT}
};

void DoIntraPrediction4x4Granulation(void)
{
    // given variables
    Ipp32s mbX, mbY, mbWidth;
    Ipp8u* pMb;
    Ipp32s imageStep;
    // working variables
    Ipp32u luma4x4BlkIdx;
    Ipp32s edgeType;
    Ipp32s mbAvailability;

    //...

    // calculate neighbouring macroblock availability
    edgeType = 0;
    if (mbX)
    {
        if (left_block_belongs_the_same_slice)
            edgeType |= IPP_LEFT;
    }
    if (mbY)
    {
        if ((mbX - 1 < mbWidth) &&
            (top_right_block_belongs_the_same_slice))
            mbAvailability |= IPP_UPPER_RIGHT;
        if (top_block_belongs_the_same_slice)
            mbAvailability |= IPP_UPPER;
        if ((mbX) &&
            (top_left_block_belongs_the_same_slice))
            mbAvailability |= IPP_UPPER_LEFT;
    }

    // main working cycle
    for (luma4x4BlkIdx = 0; luma4x4BlkIdx < 16; luma4x4BlkIdx += 1)
    {
        IppIntra4x4PredMode_H264 predMode;
        Ipp32s blockAvailability;
        Ipp8u* pBlock4x4;

        // get intra prediction mode
        predMode = (IppIntra4x4PredMode_H264)
        Intra4x4PredMode[luma4x4BlkIdx];

        // get block parameters
        {
            H264Block4x4Info_t blockInfo =
            H264Block4x4Info[luma4x4BlkIdx];

            // get block pointer
            pBlock4x4 = pMb + blockInfo.y* imageStep + blockInfo.x;

            // get block neighbours

```

```

        blockAvailability = mbAvailability &
~(blockInfo.absentNeighbour);
        blockAvailability |= blockInfo.existNeighbour;
    }

    // do intra prediction
    ippiPredictIntra_4x4_H264_8u_C1IR(pBlock4x4,
                                     imageStep,
                                     predMode,
                                     blockAvailability);

    // do block reconstruction
    // ...
}

} // void DoIntraPrediction4x4Granulation(void)

```

## PredictIntra\_16x16\_H264

*Performs intra prediction for a 16x16 luma component.*

### Syntax

```

IppStatus ippiPredictIntra_16x16_H264_8u_C1IR(Ipp8u* pSrcDst, Ipp32s srcdstStep,
IppIntra16x16PredMode predMode, Ipp32s availability);

```

### Parameters

<i>pSrcDst</i>	Pointer to the destination array.
<i>srcdstStep</i>	Distance in bytes between starts of the consecutive lines in the destination array.
<i>predMode</i>	Prediction mode of the Intra_16x16 prediction process for luma samples.
<i>availability</i>	Flag that specifies the availability of the macroblocks used for prediction.

### Description

The function `ippiPredictIntra_16x16_H264_8u_C1IR` is declared in the `ippvc.h` file. This function performs intra prediction for a 16x16 luma component in accordance with 8.3.2 of [JVTG050] in a given prediction mode *predMode* (See enumeration [IppIntra16x16PredMode](#)).

The flag *availability* is computed as follows:

```
B1*IPP_LEFT + B2*IPP_UPPER_LEFT + B3*IPP_UPPER,
```

where the constants `IPP_LEFT`, `IPP_UPPER_LEFT`, `IPP_UPPER` are from [IppLayoutFlag](#) and the variables `B1`, `B2`, `B3` may take the values of

0 - if the macroblock is unavailable for 16x16 prediction,

1 - if the macroblock is available for 16x16 prediction.

### Layout of Blocks Used for Prediction

2- IPP_UPPER_LEFT	3- IPP_UPPER
1- IPP_LEFT	<i>SrcDst</i>

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if at least one of the specified pointers is NULL.
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcDstStep</i> value is less than 16.
<code>ippStsOutOfRangeErr</code>	Indicates an error condition if <i>predMode</i> value falls outside [0,3].
<code>ippStsLPCCalcErr</code>	Indicates an error condition if <i>predMode</i> is not allowed for this block.

## PredictIntraChroma8x8\_H264

*Performs intra prediction for a 8x8 chroma component.*

### Syntax

```
IppStatus ippPredictIntraChroma8x8_H264_8u_C1IR(Ipp8u* pSrcDst, Ipp32s srcDstStep,
IppIntraChroma8x8PredMode predMode, Ipp32s availability);
```

### Parameters

<i>pSrcDst</i>	Pointer to the 8x8 chroma-block, which is inside U-frame or V-frame and which is under reconstruction (destination array).
<i>srcDstStep</i>	Distance in bytes between starts of the consecutive lines in the destination chroma-frame.
<i>predMode</i>	Prediction mode of the <code>Intra_chroma</code> prediction process for chroma samples.
<i>availability</i>	Flag that specifies the availability of the macroblocks used for prediction.

### Description

The function `ippPredictIntraChroma8x8_H264_8u_C1IR` is declared in the `ippvc.h` file. This function performs intra prediction for a 8x8 chroma component in accordance with 8.3.3 of [JVTG050] in a given prediction mode *predMode* (See enumeration [IppIntraChromaPredMode\\_H264](#)).

The flag *availability* is computed as follows:

```
B1*IPP_LEFT + B2*IPP_UPPER_LEFT + B3*IPP_UPPER,
```

where the constants `IPP_LEFT`, `IPP_UPPER_LEFT`, `IPP_UPPER` are from `IppLayoutFlag` and the variables `B1`, `B2`, `B3` may take the values of

- 0 - if the macroblock is unavailable for intra chroma prediction,
- 1 - if the macroblock is available for intra chroma prediction.

### Layout of Blocks Used for Prediction

2- <code>IPP_UPPER_LEFT</code>	3- <code>IPP_UPPER</code>
1- <code>IPP_LEFT</code>	<code>SrcDst</code>

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if at least one of the specified pointers is <code>NULL</code> .
<code>ippStsOutOfRangeErr</code>	Indicates an error condition if <code>predMode</code> value falls outside <code>[0,3]</code> .
<code>ippStsStepErr</code>	Indicates an error condition if <code>srcDstStep</code> value is less than 8.
<code>ippStsLPCCalcErr</code>	Indicates an error condition if <code>predMode</code> is not allowed for this block.

### Inter Prediction

Note that the functions with `_16u_` and `_8u_` data types use the following structures respectively:

```
typedef struct _IppVCInterpolate_16u
{
    const Ipp16u* pSrc;
    Ipp32s      srcStep;
    Ipp16u*     pDst;
    Ipp32s      dstStep;
    Ipp32s      dx;
    Ipp32s      dy;
    IppiSize    roiSize;
    Ipp32s      bitDepth;
} IppVCInterpolate_16u;
```

```
typedef struct _IppVCInterpolate_8u
{
    const Ipp8u* pSrc;
    Ipp32s      srcStep;
    Ipp8u*     pDst;
    Ipp32s      dstStep;
    Ipp32s      dx;
    Ipp32s      dy;
}
```

```

    IppiSize    roiSize;
    Ipp32s      roundControl;
} IppVCInterpolate_8u;

```

where

*pSrc*  
*srcStep*

Pointer to the source.

Distance in items between starts of the consecutive lines in the source image.

*pDst*  
*dstStep*

Pointer to the destination.

Distance in items between starts of the consecutive lines in the destination image.

*dx*, *dy*

Fractional parts of the motion vector in 1/4 pel units for luma (0, 1, 2, or 3) and 1/8 pel units for chroma (0, 1, ..., 7)

*roiSize*

Flag that specifies the dimensions of the region of interest. See structure [IppiSize](#) in Chapter 2.

*roundControl*

Frame level rounding control value as described in section 8.3.7 of [SMPTE421M], should be equal to 0 or 1. (For VC1 only, reserved for H.264.)

*bitDepth*

Number of bits of the plane sample with the range [8.. 14].

```

typedef struct _IppVCBidir_16u
{
    const Ipp16u* pSrc1;
    Ipp32s      srcStep1;
    const Ipp16u* pSrc2;
    Ipp32s      srcStep2;
    Ipp16u*      pDst;
    Ipp32s      dstStep;
    IppiSize     roi;
    Ipp32s      bitDepth;
} IppVCBidir_16u;

```

```

typedef struct _IppVCBidir_8u
{
    const Ipp8u* pSrc1;
    Ipp32s      srcStep1;
    const Ipp8u* pSrc2;
    Ipp32s      srcStep2;
    Ipp8u*      pDst;
    Ipp32s      dstStep;
    IppiSize     roiSize;
} IppVCBidir_8u;

```

where

*pSrc1*  
*srcStep1*

Pointer to the first input block

Distance in items between starts of the consecutive lines in the first input block.

*pSrc2*  
*srcStep2*

Pointer to the second input block.

Distance in items between starts of the consecutive lines in the second input block.

*pDst*  
*dstStep*

Pointer to the first output block.

Distance in items between starts of the consecutive lines in the destination block.

<i>roiSize</i>	Flag that specifies the dimensions of the region of interest. See structure <a href="#">IppiSize</a> in Chapter 2.
<i>bitDepth</i>	Number of bits of the plane sample with range [8.. 14].

The [InterpolateLumaBlock\\_H264](#) and [InterpolateChromaBlock\\_H264](#) functions use the following structures:

```
typedef struct _IppVCInterpolateBlock_16u
{
    const Ipp16u* pSrc[2]; /* pointers to reference image planes */
    Ipp32s srcStep;      /* step of the reference image planes */
    Ipp16u* pDst[2];     /* pointers to destination image planes */
    Ipp32s dstStep;      /* step of the destination image planes */
    IppiSize sizeFrame;   /* dimensions of the reference image planes */
    IppiSize sizeBlock;   /* dimensions of the block to be interpolated */
    IppiPoint pointBlockPos; /* current position of the block in the
                           image being interpolated */
    IppiPoint pointVector; /* relative difference between current
                           position and reference data to be used */
    Ipp32s bitDepth      /* data capacity depth in range 8..14 */
} IppVCInterpolateBlock_16u;
```

```
typedef struct _IppVCInterpolateBlock_8u
{
    const Ipp8u* pSrc[2]; /* pointers to reference image planes */
    Ipp32s srcStep;      /* step of the reference image planes */
    Ipp8u* pDst[2];     /* pointers to destination image planes */
    Ipp32s dstStep;      /* step of the destination image planes */
    IppiSize sizeFrame;   /* dimensions of the reference image planes */
    IppiSize sizeBlock;   /* dimensions of the block to be interpolated */
    IppiPoint pointBlockPos; /* current position of the block in the
                           image being interpolated */
    IppiPoint pointVector; /* relative difference between current
                           position and reference data to be used */
} IppVCInterpolateBlock_8u;
```

where

<i>pSrc</i>	Array of pointers to reference planes; <a href="#">InterpolateLumaBlock_H264</a> uses only first pointer of the array, <a href="#">InterpolateChromaBlock_H264</a> uses both pointers.
<i>srcStep</i>	Stride of the reference planes (the distance between the nearest image lines)
<i>pDst</i>	Array of pointers to destination buffers; <a href="#">InterpolateLumaBlock_H264</a> uses only first pointer of the array, <a href="#">InterpolateChromaBlock_H264</a> uses both pointers. The array should be big enough to hold the interpolated block.
<i>dstStep</i>	Stride of the destination buffer (the distance between the nearest image lines)
<i>sizeFrame</i>	Dimensions of the reference planes that are pointed to by the source pointers
<i>sizeBlock</i>	Dimensions of the block to interpolate
<i>pointBlockPos</i>	Current position of the block in the frame under reconstruction
<i>pointVector</i>	Motion vector

*bitDepth*

Bit capacity of data that is being processed.

## ExpandPlane\_H264

*Expands plane.*

---

### Syntax

```
IppStatus ippiExpandPlane_H264_8u_C1R(Ipp8u* StartPtr, Ipp32u uFrameWidth, Ipp32u uFrameHeight, Ipp32u uPitch, Ipp32u uPels, IppvcFrameFieldFlag uFrameFieldFlag);
```

### Parameters

<i>StartPtr</i>	Pointer to the frame source data.
<i>uFrameWidth</i>	Frame width in pixels.
<i>uFrameHeight</i>	Frame height in pixels.
<i>uPitch</i>	Frame width in bytes.
<i>uPels</i>	Number of pixels filled to the right/left and up/down in process of expansion.
<i>uFrameFieldFlag</i>	Flag that defines expansion method for various image types: frame, top field, bottom field. See the corresponding enumerator <code>IPPVC_FRAME_FIELD_FLAG</code> .

### Description

The function `ippiExpandPlane_H264_8u_C1R` is declared in the `ippvc.h` file. This function expands the plane by adding pixels to the top and bottom rows and to the left and right columns of the frame.

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>StartPtr</i> is NULL.

### Example

The code example below shows how to use `ExpandPlane_H264` function.

```
{
    Ipp8u* pPlane = pointer_on_frame;
    Ipp32s width, height, pitch, padding;

    width = width_of_frame; // without padding
    height = height_of_frame; // without padding
    pitch = pitch_of_frame; // with padding

    padding = 32; // padding size.

    IppvcFrameFieldFlag ff_flag;

    if (!is_field_flag)
    {
        ff_flag = IPPVC_FRAME;
    }
    else
```

```

{
    if(!is_bottom_field)
    {
        ff_flag = IPPVC_TOP_FIELD;
    }
    else
    {
        ff_flag = IPPVC_BOTTOM_FIELD;
    }
}

ippiExpandPlane_H264_8u_C1R(pPlane,
                             width,
                             height,
                             picth,
                             padding,
                             ff_flag);
}

```

## InterpolateLuma\_H264

*Performs interpolation for motion estimation of the luma component using fractional part of motion vector.*

---

### Syntax

```

IppStatus ippiInterpolateLuma_H264_8u_C1R(const Ipp8u* pSrc, Ipp32s srcStep, Ipp8u* pDst,
Ipp32s dstStep, Ipp32s dx, Ipp32s dy, IppiSize roiSize);

```

```

IppStatus ippiInterpolateLuma_H264_16u_C1R(const IppVCInterpolate_16u* interpolateInfo);

```

### Parameters

<i>pSrc</i>	Pointer to the source.
<i>srcStep</i>	Distance in items between starts of the consecutive lines in the source block.
<i>pDst</i>	Pointer to the destination.
<i>dstStep</i>	Distance in items between starts of the consecutive lines in the destination block.
<i>dx, dy</i>	Fractional parts of the motion vector in 1/4 pel units (0, 1, 2, or 3).
<i>roiSize</i>	Flag that specifies the dimensions of the region of interest (could be 16, 8 or 4 in each dimension). See structure <a href="#">IppiSize</a> in Chapter 2.
<i>interpolateInfo</i>	Pointer to the <a href="#">IppVCInterpolate_16u</a> structure.

### Description

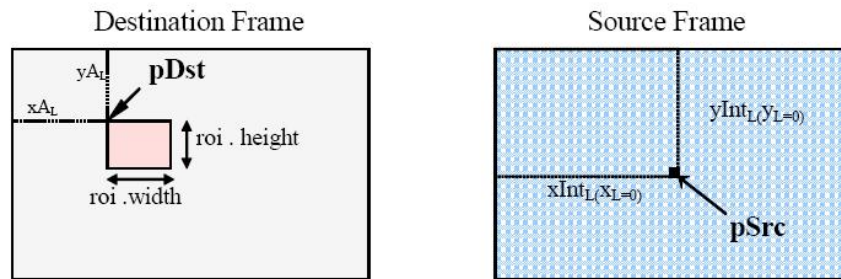
The functions `ippiInterpolateLuma_H264_8u_C1R` and `ippiInterpolateLuma_H264_16u_C1R` are declared in the `ippvc.h` file. These functions perform interpolation (convolution with 6x6 kernel) for motion estimation of the luma component in accordance with 8.4.2.2.1 of [JVTG050].

The functions use only the fractional part of the motion vector. The integer part is used when finding the position (pointed to by *pSrc*) in the source frame.



The `ippiInterpolateLuma_H264` functions do not provide any padding, therefore the whole data region indicated by the pointer should be real and increased by padding, if necessary. The functions can use up to 3 pixels up and left from the pointer and maximum 2 pixels down and right from the pointed data size.

### Interpolation for Motion Estimation of Luma Component



**NOTE.** Make sure all samples that are used for interpolation are within the boundaries of the source frame. To solve the problem, enlarge the source frame by using boundary samples or use `InterpolateLumaTop_H264` or `InterpolateLumaBottom_H264` functions.

These functions are used in the H.264 decoder and encoder included into Intel IPP Samples. See [introduction](#) to the H.264 section.

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if at least one of the specified pointers is NULL.
<code>ippStsStepErr</code>	Indicates an error condition if <code>srcStep</code> value is less than 16.
<code>ippStsBadArgErr</code>	Indicates an error condition if <code>dx</code> or <code>dy</code> values fall outside [0,3].
<code>ippStsSizeErr</code>	Indicates an error condition if <code>roi.width</code> or <code>roi.height</code> take values other than {16, 8, 4}.

### Example

The code example below shows how to use `InterpolateLuma_H264` function.

```

Ipp32s iHVector, iVVector;
IppiSize roi = {width_of_luma_block, height_of_luma_block};
Ipp32s iLumaXPos = x_position_of_current_luma_block;
Ipp32s iLumaYPos = y_position_of_current_luma_block;
Ipp8u* pRefYPlane = corresponding_luma_block_of_reference_luma_frame;
Ipp32s iPitch = reference_luma_frame_pitch;
Ipp8u* pDstY = destination_luma_block;
Ipp32s iDstPitch = destination_luma_frame_pitch;

// set motion vectors
iHVector = horizontal_motion_vector;

```

```

iVVector = vertical_motion_vector;

{
    Ipp32s iHFraction, iVFraction;
    Ipp32s width, height;

    width = width_of_frame;
    height = height_of_frame;

    // prepare the horizontal vector, extract an interpolation factor
    iHFraction = iHVector & 3;
    iHVector >>= 2;

    // saturate the horizontal vector
    {
        Ipp32s iLeft = ((iHFraction) ? (1) : (0));
        Ipp32s iRight;

        // set amount of additional pixels
        iRight = iLeft* 3;
        iLeft = iLeft* 2;

        if (((unsigned) (width - (iRight + iLeft) - roi.width)) <
            ((unsigned) (iHVector + iLumaXPos - iLeft)))
        {
            // saturate to the left bound
            if (- iRight - roi.width >
                iHVector + iLumaXPos)
            {
                iHVector = - iLumaXPos - roi.width - 3;
            }
            // saturate to the right bound
            else if (iLeft + width <
                    iHVector + iLumaXPos)
            {
                iHVector = width - iLumaXPos + 2;
            }
        }
    }

    // prepare the vertical vector, extract an interpolation factor
    iVFraction = iVVector & 3;
    iVVector >>= 2;
    // most usual prediction
    // NOTE: the height is decremented to avoid factor's influence
    {
        Ipp32s iTop = ((iVFraction) ? (1) : (0));
        Ipp32s iBottom;

        // set amount of additional pixels
        iBottom = iTop* 3;
        iTop = iTop* 2;

        if (((unsigned) (height - (iTop + iBottom) - roi.height)) >=
            ((unsigned) (iVVector + iLumaYPos - iTop)))
        {
            ippiInterpolateLuma_H264_8u_C1R(pRefYPlane + iVVector * iPitch + iHVector,
                iPitch,
                pDstY,
                iDstPitch,
                iHFraction,
                iVFraction,
                roi);
        }
    }
}

```

```

    }
    // prediction from the top of the frame
    else if (0 > iVVector + iLumaYPos - iTop)
    {
        // sometimes the vector points to nothing
        ippiInterpolateLumaTop_H264_8u_C1R(pRefYPlane + iVVector * iPitch + iHVector,
                                           iPitch,
                                           pDstY,
                                           iDstPitch,
                                           iHFraction,
                                           - iVVector - iLumaYPos,
                                           roi);
    }
    // prediction from the bottom of the frame
    else
    {
        // sometimes the vector points to nothing
        ippiInterpolateLumaBottom_H264_8u_C1R(pRefYPlane + iVVector * iPitch + iHVector,
                                              iPitch,
                                              pDstY,
                                              iDstPitch,
                                              iHFraction,
                                              iVFraction,
                                              iVVector + iLumaYPos + roi.height - height,
                                              roi);
    }
}
}
}

```

## InterpolateLumaTop\_H264

*Performs interpolation for motion estimation of the luma component at the frame top boundary.*

### Syntax

```

IppStatus ippiInterpolateLumaTop_H264_8u_C1R(const Ipp8u* pSrc, Ipp32s srcStep, Ipp8u*
pDst, Ipp32s dstStep, Ipp32s dx, Ipp32s dy, Ipp32s outPixels, IppiSize roiSize);

IppStatus ippiInterpolateLumaTop_H264_16u_C1R(const IppVCInterpolate_16u* interpolateInfo,
Ipp32s outPixels);

```

### Parameters

<i>pSrc</i>	Pointer to the source.
<i>srcStep</i>	Distance in items between starts of the consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination.
<i>dstStep</i>	Distance in items between starts of the consecutive lines in the destination image.
<i>dx, dy</i>	Fractional parts of the motion vector in 1/4 pel units (0, 1, 2, or 3).
<i>outPixels</i>	Number of pixels by which the data specified by <i>pSrc</i> reaches over the frame top boundary.

<i>roiSize</i>	Flag that specifies the dimensions of the region of interest (could be 16, 8 or 4 in each dimension). See <a href="#">IppiSize</a> in Chapter 2.
<i>interpolateInfo</i>	Pointer to the <a href="#">IppVCInterpolate_16u</a> structure.

### Description

The functions `ippiInterpolateLumaTop_H264_8u_C1R` and `ippiInterpolateLumaTop_H264_16u_C1R` are declared in the `ippvc.h` file. These functions perform interpolation (convolution with 6x6 kernel) for motion estimation of the luma component near the top boundary of the frame in accordance with 8.4.2.2.1 of [JVTG050]. Instead of data lines located in the invalid area, the closest line in the valid area is used.

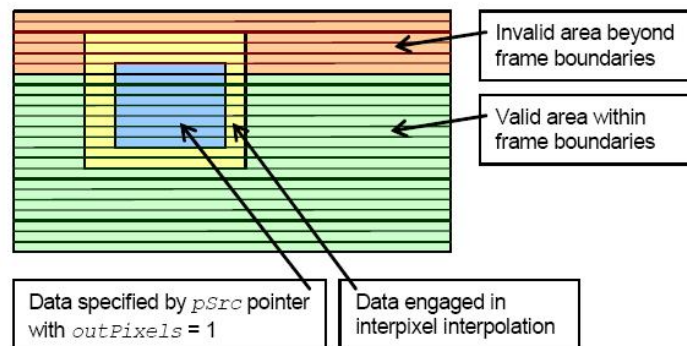
The functions use only the fractional part of the motion vector. The integer part is used when finding the position (pointed to by *pSrc*) in the source frame.

The functions are intended for processing data close to the frame top border and do not require any vertical padding. Horizontal padding is required. The function can use up to 3 pixels left from the pointer and maximum 2 pixels right from the pointed data size.

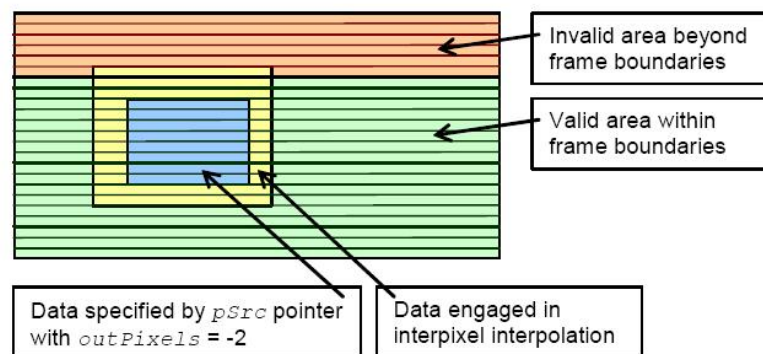
See [Example](#) for a usage example.

Figure "Luma Interpolation at Top Boundary with Positive *outPixels* Value" shows the data specified by *pSrc* and the data actually used in interpixel interpolation at the picture top boundary with a positive *outPixels* value. Figure "Luma Interpolation at Top Boundary with Negative *outPixels* Value" shows the data with a negative *outPixels* value.

### Luma Interpolation at Top Boundary with Positive *outPixels* Value



### Luma Interpolation at Top Boundary with Negative *outPixels* Value



### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if at least one of the specified pointers is NULL.
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> value is less than 16.

<code>ippStsBadArgErr</code>	Indicates an error condition if <code>dx</code> or <code>dy</code> values fall outside [0,3].
<code>ippStsSizeErr</code>	Indicates an error condition if <code>roi.width</code> or <code>roi.height</code> take values other than {16, 8, 4}.

## InterpolateLumaBottom\_H264

*Performs interpolation for motion estimation of the luma component at the frame bottom boundary.*

---

### Syntax

```
ippStatus ippiInterpolateLumaBottom_H264_8u_C1R(const Ipp8u* pSrc, Ipp32s srcStep, Ipp8u* pDst, Ipp32s dstStep, Ipp32s dx, Ipp32s dy, Ipp32s outPixels, IppiSize roiSize);
```

```
ippStatus ippiInterpolateLumaBottom_H264_16u_C1R(const IppVCInterpolate_16u* interpolateInfo, Ipp32s outPixels);
```

### Parameters

<code>pSrc</code>	Pointer to the source.
<code>srcStep</code>	Distance in items between starts of the consecutive lines in the source block.
<code>pDst</code>	Pointer to the destination.
<code>dstStep</code>	Distance in items between starts of the consecutive lines in the destination block.
<code>dx, dy</code>	Fractional parts of the motion vector in 1/4 pel units (0, 1, 2, or 3).
<code>outPixels</code>	Number of pixels by which the data specified by <code>pSrc</code> reaches over the frame bottom boundary.
<code>roiSize</code>	Flag that specifies the dimensions of the region of interest (could be 16, 8 or 4 in each dimension). See structure <a href="#">IppiSize</a> in Chapter 2.
<code>interpolateInfo</code>	Pointer to the <a href="#">IppVCInterpolate_16u</a> structure.

### Description

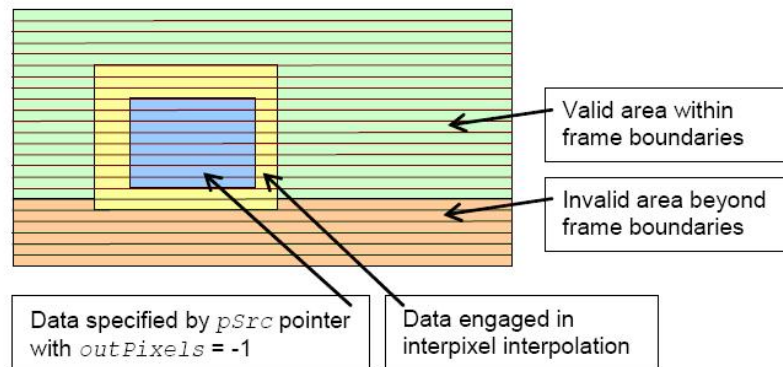
The functions `ippiInterpolateLumaBottom_H264_8u_C1R` and `ippiInterpolateLumaBottom_H264_16u_C1R` are declared in the `ippvc.h` file. These functions perform interpolation (convolution with 6x6 kernel) for motion estimation of the luma component near the bottom boundary of the frame in accordance with 8.4.2.2.1 of [JVTG050]. Instead of data lines located in the invalid area, the closest line in the valid area is used.

The functions use only the fractional part of the motion vector. The integer part is used when finding the position (pointed to by `pSrc`) in the source frame.

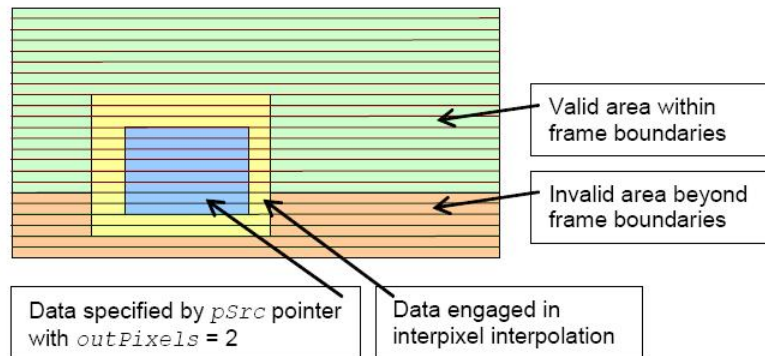
See [Example 16-18](#) for a usage example.

[Figure "Luma Interpolation at Bottom Boundary with Positive outPixels Value"](#) shows the data specified by `pSrc` and the data actually used in interpixel interpolation at the picture top boundary with a positive `outPixels` value. [Figure "Luma Interpolation at Bottom Boundary with Negative outPixels Value"](#) shows the data with a negative `outPixels` value.

### Luma Interpolation at Bottom Boundary with Positive *outPixels* Value



### Luma Interpolation at Bottom Boundary with Negative *outPixels* Value



### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if at least one of the specified pointers is NULL.
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> value is less than 16.
<code>ippStsBadArgErr</code>	Indicates an error condition if <i>dx</i> or <i>dy</i> values fall outside [0,3].
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roi.width</i> or <i>roi.height</i> take values other than {16, 8, 4}.

## InterpolateLumaBlock\_H264

*Performs interpolation for motion estimation of the luma component using entire motion vector.*

---

### Syntax

```
IppStatus ippiInterpolateLumaBlock_H264_8u_P1R(const IppVCInterpolateBlock_8u*  
interpolateInfo);
```

```
IppStatus ippiInterpolateLumaBlock_H264_16u_P1R(const IppVCInterpolateBlock_16u*  
interpolateInfo);
```

### Parameters

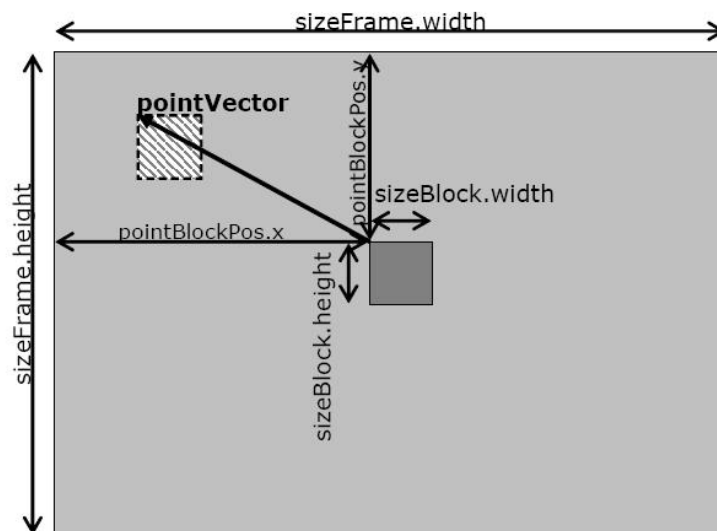
*interpolateInfo*                      Pointer to an instance of the structure holding interpolation parameters. See [IppVCInterpolateBlock\\_8u](#) and [IppVCInterpolateBlock\\_16u](#) structures.

### Description

The functions `ippiInterpolateLumaBlock_H264_8u_P1R` and `ippiInterpolateLumaBlock_H264_16u_P1R` are declared in the `ippvc.h` file. These functions perform interpolation (convolution with 6x6 kernel) for motion estimation of the luma component in accordance with 8.4.2.2.1 of [JVTG050]. Unlike [InterpolateLuma\\_H264](#), the functions use an entire motion vector when they calculate fractional and integer parts of the vectors. The functions also handle cases of overlapping, when the source block pointed to by the motion vector lies out of the source frame. Non-existing samples are cloned from the nearest existing samples.

### Interpolation of Luminance Component Block for Motion Estimation

---





## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if at least one of the specified pointers is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <code>roi.width</code> or <code>roi.height</code> take values other than {16, 8, 4}.

## Example

The code example below shows how to use `ippiInterpolateLumaBlock_H264_8u_P1R` function.

```
{
    IppVCInterpolateBlock_8u interpolateInfo;

    interpolateInfo.pSrc[0] = reference_luma_frame;
    interpolateInfo.srcStep = reference_luma_frame_pitch;

    // create vectors & factors
    interpolateInfo.pointVector.x = horizontal_motion_vector;
    interpolateInfo.pointVector.y = vertical_motion_vector;

    // fill parameters
    interpolateInfo.pDst[0] = destination_luma_block;
    interpolateInfo.dstStep = destination_luma_frame_pitch;
    interpolateInfo.sizeFrame = {width_of_luma_frame, height_of_luma_frame};
    interpolateInfo.sizeBlock = { width_of_luma_block, height_of_luma_block};
    interpolateInfo.pointBlockPos.x = x_position_of_current_luma_block;
    interpolateInfo.pointBlockPos.y = y_position_of_current_luma_block;

    ippiInterpolateLumaBlock_H264_8u_P1R(&interpolateInfo);
}
```

## InterpolateChroma\_H264

*Performs interpolation for motion estimation of the chroma component using fractional part of motion vector.*

### Syntax

```
IppStatus ippiInterpolateChroma_H264_8u_C1R(const Ipp8u* pSrc, Ipp32s srcStep, Ipp8u*
pDst, Ipp32s dstStep, Ipp32s dx, Ipp32s dy, IppiSize roiSize);

IppStatus ippiInterpolateChroma_H264_8u_C2P2R(const Ipp8u* pSrcUV, Ipp32s srcStep, Ipp8u*
pDstU, Ipp8u* pDstV, Ipp32s dstStep, Ipp32s dx, Ipp32s dy, IppiSize roi);

IppStatus ippiInterpolateChroma_H264_16u_C1R(const IppVCInterpolate_16u* interpolateInfo);
```

### Parameters

<code>pSrc</code>	Pointer to the source
<code>pSrcUV</code>	Pointer to the source block (chrominance part of NV12 plane)
	<pre> 0  UV UV UV UV UV UV UV UV 1  UV UV UV UV UV UV UV UV ... 7  UV UV UV UV UV UV UV UV</pre>

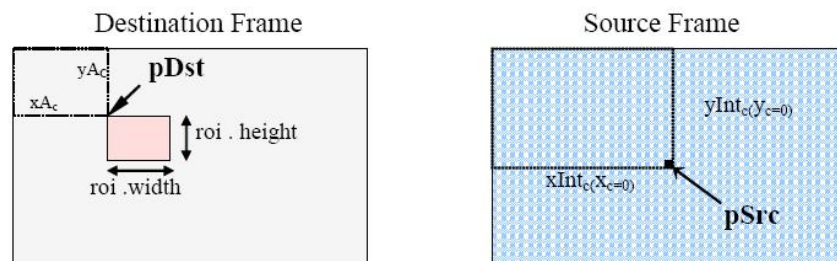
<i>srcStep</i>	Distance in items between starts of the consecutive lines in the source image
<i>pDst</i>	Pointer to the destination buffer
<i>pDstU</i>	Pointer to the destination buffer for interpolated <i>u</i> coefficients
<i>pDstV</i>	Pointer to the destination buffer for interpolated <i>v</i> coefficients
<i>dstStep</i>	Distance in items between starts of the consecutive lines in the destination image
<i>dx, dy</i>	Fractional parts of the motion vector in 1/8 pel units (0, 1, ..., 7)
<i>roiSize</i>	Flag that specifies the dimensions of the region of interest (could be 16, 8, 4 or 2 in height dimension and 8, 4 or 2 in width dimension). See structure <a href="#">IppiSize</a> in Chapter 2.
<i>roi</i>	Size value that specifies the dimensions of the ROI; could be 16, 8 or 4 in each dimension)
<i>interpolateInfo</i>	Pointer to the <a href="#">IppVCInterpolate_16u</a> structure

## Description

The functions `ippiInterpolateChroma_H264_8u_C1R`, `ippiInterpolateChroma_H264_8u_C2P2R`, and `ippiInterpolateChroma_H264_16u_C1R` are declared in the `ippvc.h` file. These functions perform interpolation for motion estimation of the chroma component in accordance with 8.4.2.2.2 of [JVTG050].

The functions use only the fractional part of the motion vector. The integer part is used when finding the position (pointed to by *pSrc*) in the source frame.

## Interpolation for Motion Estimation of Chroma Component



**NOTE.** Make sure all samples that are used for interpolation are within the boundaries of the source frame. To solve the problem, enlarge the source frame by using boundary samples or use [InterpolateChromaTop\\_H264](#) or [InterpolateChromaBottom\\_H264](#) functions.

`ippiInterpolateChroma_H264_8u_C2P2R` is a clone of `ippiInterpolateChroma_H264_8u_C1R` but the source image is a chrominance part of an NV12 plane:

```
YY YY YY YY
YY YY YY YY
UV UV UV UV - chrominance part of NV12 plane
```

.

These functions are used in the H.264 decoder and encoder included into Intel IPP Samples. See [introduction](#) to the H.264 section.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if at least one of the specified pointers is NULL.
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> value is less than 8.
<code>ippStsBadArgErr</code>	Indicates an error condition if <i>dx</i> or <i>dy</i> values fall outside [0,7].
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roi.width</i> or <i>roi.height</i> take values other than 16, 8, 4 or 2.

## Example

The code example below shows how to use `InterpolateChroma_H264_8u_C1R` function.

```
Ipp32s iHVector, iVVector;
IppiSize roi_chroma = {width_of_chroma_block, height_of_chroma_block};
Ipp32s iChromaXPos = x_position_of_current_chroma_block;
Ipp32s iChromaYPos = y_position_of_current_chroma_block;
Ipp8u* pRefUPlane = corresponding_u_block_of_reference_u_frame;
Ipp8u* pRefVPlane = corresponding_v_block_of_reference_v_frame;
Ipp32s iPitch = reference_chroma_frame_pitch;
Ipp8u* pDstU = destination_u_block;
Ipp8u* pDstV = destination_v_block;
Ipp32s iDstPitch = destination_chroma_frame_pitch;

// create motion vectors
iHVector = horizontal_motion_vector;
iVVector = vertical_motion_vector;

{
    Ipp32s iHFraction, iVFraction;
    Ipp32s width, height;

    width = width_of_chroma_frame;
    height = height_of_chroma_frame;

    // prepare the horizontal vector, extract an interpolation fraction
    iHVector <<= ((Ipp32s) (3 <= color_format));
    iHFraction = iHVector & 7;
    iHVector >>= 3;

    // saturate the horizontal vector
    if (((unsigned) (width - 1 - roi_chroma.width)) <
        ((unsigned) (iHVector + iChromaXPos)))
    {
        // saturate to the left bound
        if (- 1 - roi_chroma.width >
```

```

        iHVector + iChromaXPos)
    {
        iHVector = - iChromaXPos - roi_chroma.width - 1;
    }
    // saturate to the right bound
    else if (width <
        iHVector + iChromaXPos)
    {
        iHVector = width - iChromaXPos;
    }
}

// prepare the vertical vector, extract an interpolation factor
iVVector <<=((Ipp32s) (2 <= color_format));
iVFraction = iVVector & 7;
iVVector >>= 3;

// most usual prediction
// NOTE: we decrement the height to avoid factor's influence
if (((unsigned) (height - 1 - roi_chroma.height)) >=
    ((unsigned) (iVVector + iChromaYPos)))
{
    ippiInterpolateChroma_H264_8u_C1R(pRefVPlane + iVVector* iPitch + iHVector,
        iPitch,
        pDstV,
        iDstPitch,
        iHFraction,
        iVFraction,
        roi_chroma);

    ippiInterpolateChroma_H264_8u_C1R(pRefUPlane + iVVector* iPitch + iHVector,
        iPitch,
        pDstU,
        iDstPitch,
        iHFraction,
        iVFraction,
        roi_chroma);
}
// prediction from the top of the frame
else if (0 > iVVector)
{
    // sometimes the vector points nothing
    ippiInterpolateChromaTop_H264_8u_C1R(pRefVPlane + iVVector* iPitch + iHVector,
        iPitch,
        pDstV,
        iDstPitch,
        iHFraction,
        iVFraction,
        - iVVector - iChromaYPos,
        roi_chroma);

    ippiInterpolateChromaTop_H264_8u_C1R(pRefUPlane + iVVector* iPitch + iHVector,
        iPitch,
        pDstU,
        iDstPitch,
        iHFraction,
        iVFraction,
        - iVVector - iChromaYPos,
        roi_chroma);
}
// prediction from the bottom of the frame

```

```

else
{
    // sometimes the vector points nothing
    ippiInterpolateChromaBottom_H264_8u_C1R(pRefVPlane + iVVector* iPitch + iHVector,
                                             iPitch,
                                             pDstV,
                                             iDstPitch,
                                             iHFraction,
                                             iVFraction,
                                             iVVector + roi_chroma.height +
                                             iChromaYPos - height,
                                             roi_chroma);

    ippiInterpolateChromaBottom_H264_8u_C1R(pRefUPlane + iVVector* iPitch + iHVector,
                                             iPitch,
                                             pDstU,
                                             iDstPitch,
                                             iHFraction,
                                             iVFraction,
                                             iVVector + roi_chroma.height +
                                             iChromaYPos - height,
                                             roi_chroma);
}
}

```

The code example below shows how to use `InterpolateChroma_H264_8u_C2P2R` function.

```

Ipp32s iHVector, iVVector;
IppiSize roi_chroma = {width_of_chroma_block, height_of_chroma_block};
Ipp32s iChromaXPos = x_position_of_current_chroma_block;
Ipp32s iChromaYPos = y_position_of_current_chroma_block;
Ipp8u* pRefUVPlane = corresponding_uv_block_of_reference_uv_frame;
Ipp32s iPitch = reference_chroma_frame_pitch;
Ipp8u* pDstU = destination_u_block;
Ipp8u* pDstV = destination_v_block;
Ipp32s iDstUPitch = destination_uv_block_pitch;

// create motion vectors
iHVector = horizontal_motion_vector;
iVVector = vertical_motion_vector;

{
    Ipp32s iHFraction, iVFraction;
    Ipp32s width, height;

    width = width_of_chroma_frame;
    height = height_of_chroma_frame;

    // prepare the horizontal vector, extract an interpolation fraction
    iHVector <= ((Ipp32s) (3 <= color_format));
    iHFraction = iHVector & 7;
    iHVector >= 3;

    // saturate the horizontal vector
    if (((unsigned) (width - 1 - roi_chroma.width)) <
        ((unsigned) (iHVector + iChromaXPos)))
    {
        // saturate to the left bound
        if (- 1 - roi_chroma.width >
            iHVector + iChromaXPos)
        {
            iHVector = - iChromaXPos - roi_chroma.width - 1;
        }
    }
}

```

```

        // saturate to the right bound
        else if (width <
                iHVector + iChromaXPos)
        {
            iHVector = width - iChromaXPos;
        }
    }

    // prepare the vertical vector, extract an interpolation factor
    iVVector <=<((Ipp32s) (2 <= color_format));
    iVFraction = iVVector & 7;
    iVVector >>= 3;

    // most usual prediction
    // NOTE: we decrement the height to avoid factor's influence
    if (((unsigned) (height - 1 - roi_chroma.height)) >=
        ((unsigned) (iVVector + iChromaYPos)))
    {
        ippiInterpolateChroma_H264_8u_C2P2R(pRefUVPlane + iVVector* iPitch + 2*iHVector,
            iPitch,
            pDstU,
            pDstV,
            iDstUVPitch,
            iHFraction,
            iVFraction,
            roi_chroma);
    }
    // prediction from the top of the frame
    else if (0 > iVVector)
    {
        // ....
    }
}

```

## InterpolateChromaTop\_H264

*Performs interpolation for motion estimation of the chroma component at the frame top boundary.*

---

### Syntax

```
IppStatus ippiInterpolateChromaTop_H264_8u_C1R(const Ipp8u* pSrc, Ipp32s srcStep, Ipp8u* pDst, Ipp32s dstStep, Ipp32s dx, Ipp32s dy, Ipp32s outPixels, IppiSize roiSize);
```

```
IppStatus ippiInterpolateChromaTop_H264_16u_C1R(const IppVCInterpolate_16u* interpolateInfo, Ipp32s outPixels);
```

### Parameters

<i>pSrc</i>	Pointer to the source.
<i>srcStep</i>	Distance in items between starts of the consecutive lines in the source block.
<i>pDst</i>	Pointer to the destination.
<i>dstStep</i>	Distance in items between starts of the consecutive lines in the destination block.
<i>dx</i> , <i>dy</i>	Fractional parts of the motion vector in 1/8 pel units (0, 1, ..., 7).

<i>outPixels</i>	Number of pixels by which the data specified by <i>pSrc</i> reaches over the frame top boundary.
<i>roiSize</i>	Flag that specifies the dimensions of the region of interest (could be 16, 8, 4 or 2 in height dimension and 8, 4 or 2 in width dimension) See <a href="#">IppiSize</a> in Chapter 2.
<i>interpolateInfo</i>	Pointer to the <a href="#">IppVCInterpolate_16u</a> structure.

## Description

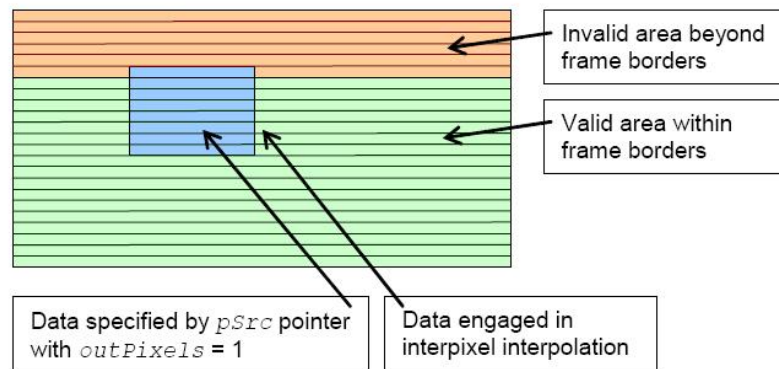
The functions `ippiInterpolateChromaTop_H264_8u_C1R` and `ippiInterpolateChromaTop_H264_16u_C1R` are declared in the `ippvc.h` file. These functions perform interpolation for motion estimation of the chroma component near the top boundary of the frame in accordance with 8.4.2.2.2 of [JVTG050]. Instead of data lines located in the invalid area, the closest line in the valid area is used.

The functions use only the fractional part of the motion vector. The integer part is used when finding the position (pointed to by *pSrc*) in the source frame.

See [Example 16-20](#) for a usage example.

Figure "Chroma Interpolation at Top Boundary" shows the data specified by *pSrc* in interpixel interpolation at the picture top boundary.

## Chroma Interpolation at Top Boundary



## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if at least one of the specified pointers is NULL.
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> value is less than 16.
<code>ippStsBadArgErr</code>	Indicates an error condition if <i>dx</i> or <i>dy</i> values fall outside [0,3].
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roi.width</i> or <i>roi.height</i> take values other than {16, 8, 4}.

## InterpolateChromaBottom\_H264

*Performs interpolation for motion estimation of the chroma component at the frame bottom boundary.*

---

### Syntax

```
IppStatus ippiInterpolateChromaBottom_H264_8u_C1R(const Ipp8u* pSrc, Ipp32s srcStep, Ipp8u* pDst, Ipp32s dstStep, Ipp32s dx, Ipp32s dy, Ipp32s outPixels, IppiSize roiSize);
```

```
IppStatus ippiInterpolateChromaBottom_H264_16u_C1R(const IppVCInterpolate_16u* interpolateInfo, Ipp32s outPixels);
```

### Parameters

<i>pSrc</i>	Pointer to the source.
<i>srcStep</i>	Distance in items between starts of the consecutive lines in the source block.
<i>pDst</i>	Pointer to the destination.
<i>dstStep</i>	Distance in items between starts of the consecutive lines in the destination block.
<i>dx, dy</i>	Fractional parts of the motion vector in 1/8 pel units (0, 1, ..., 7).
<i>outPixels</i>	Number of pixels by which the data specified by <i>pSrc</i> reaches over the frame bottom boundary.
<i>roiSize</i>	Flag that specifies the dimensions of the region of interest (could be 16, 8, 4 or 2 in height dimension and 8, 4 or 2 in width dimension). See structure <a href="#">IppiSize</a> in Chapter 2.
<i>interpolateInfo</i>	Pointer to the <a href="#">IppVCInterpolate_16u</a> structure.

### Description

The functions `ippiInterpolateChromaBottom_H264_8u_C1R` and `ippiInterpolateChromaBottom_H264_16u_C1R` are declared in the `ippvc.h` file. These functions perform interpolation for motion estimation of the chroma component near the bottom boundary of the frame in accordance with 8.4.2.2.2 of [JVTG050]. Instead of data lines located in the invalid area, the closest line in the valid area is used.

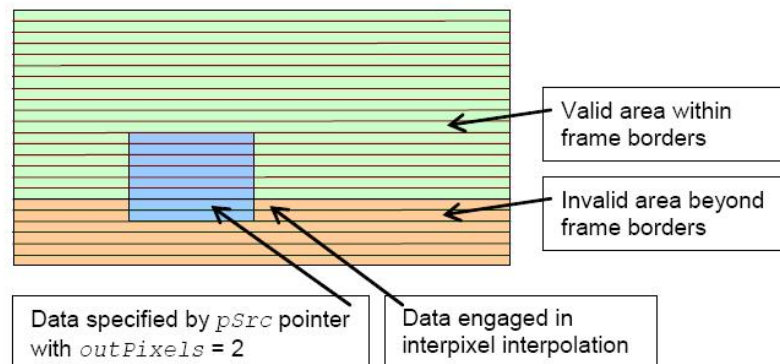
The functions use only the fractional part of the motion vector. The integer part is used when finding the position (pointed to by *pSrc*) in the source frame.

See [Example](#) for a usage example.



Figure "Chroma Interpolation at Bottom Boundary" shows the data specified by *pSrc* in interpixel interpolation at the picture top boundary.

### Chroma Interpolation at Bottom Boundary



### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if at least one of the specified pointers is NULL.
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> value is less than 16.
<code>ippStsBadArgErr</code>	Indicates an error condition if <i>dx</i> or <i>dy</i> values fall outside [0,3].
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roi.width</i> or <i>roi.height</i> take values other than {16, 8, 4}.

## InterpolateChromaBlock\_H264

*Performs interpolation for motion estimation of the chroma component using entire motion vector.*

### Syntax

```

IppStatus ippiInterpolateChromaBlock_H264_8u_P2R(const IppVCInterpolateBlock_8u*
interpolateInfo);

IppStatus ippiInterpolateChromaBlock_H264_8u_C2P2R(const IppVCInterpolateBlock_8u*
interpolateInfo);

IppStatus ippiInterpolateChromaBlock_H264_8u_C2C2R(const IppVCInterpolateBlock_8u*
interpolateInfo);

IppStatus ippiInterpolateChromaBlock_H264_16u_P2R(const IppVCInterpolateBlock_16u*
interpolateInfo);

```

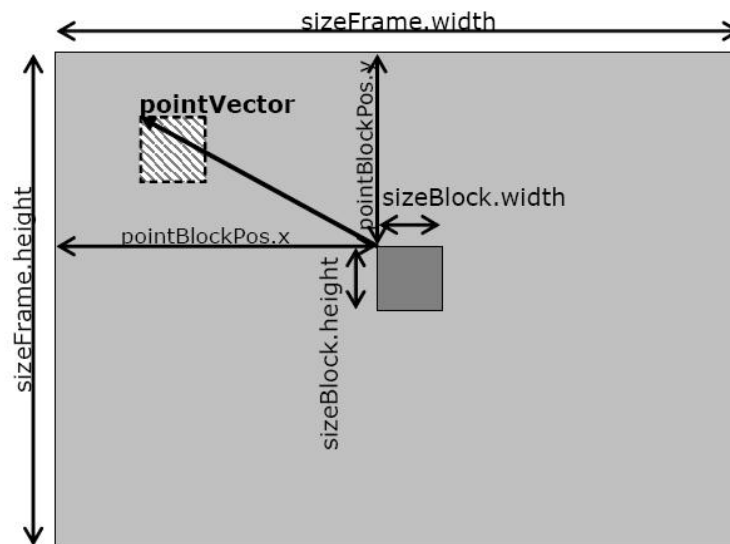
## Parameters

<i>interpolateInfo</i>	Pointer to an instance of the structure holding interpolation parameters. See <a href="#">IppVCInterpolateBlock_8u</a> and <a href="#">IppVCInterpolateBlock_16u</a> structures. For <code>ippiInterpolateChromaBlock_H264_8u_C2P2R</code> :																																								
<i>interpolateInfo-&gt;pSrc[0]</i>	Pointer to the source block (chrominance part of the NV12 plane). <div><table><tr><td>0</td><td>UV</td><td>UV</td><td>UV</td><td>UV</td><td>UV</td><td>UV</td><td>UV</td><td>UV</td><td>UV</td></tr><tr><td>1</td><td>UV</td><td>UV</td><td>UV</td><td>UV</td><td>UV</td><td>UV</td><td>UV</td><td>UV</td><td>UV</td></tr><tr><td>...</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr><tr><td>7</td><td>UV</td><td>UV</td><td>UV</td><td>UV</td><td>UV</td><td>UV</td><td>UV</td><td>UV</td><td>UV</td></tr></table></div>	0	UV	UV	UV	UV	UV	UV	UV	UV	UV	1	UV	UV	UV	UV	UV	UV	UV	UV	UV	...										7	UV	UV	UV	UV	UV	UV	UV	UV	UV
0	UV	UV	UV	UV	UV	UV	UV	UV	UV																																
1	UV	UV	UV	UV	UV	UV	UV	UV	UV																																
...																																									
7	UV	UV	UV	UV	UV	UV	UV	UV	UV																																
<i>interpolateInfo-&gt;pSrc[1]</i>	Ignored																																								
<i>interpolateInfo-&gt;pDst[0]=pDstU</i>	Pointer to the destination buffer for interpolated U coefficients. Chrominance part of the NV12 plane for <code>8u_C2P2R</code> and <code>8u_C2C2R</code> functions.																																								
<i>interpolateInfo-&gt;pDst[1]=pDstV</i>	Pointer to the destination buffer for interpolated v coefficients. Ignored for <code>8u_C2P2R</code> and <code>8u_C2C2R</code> functions.																																								

## Description

The functions `ippiInterpolateChromaBlock_H264_8u_P2R`, `ippiInterpolateChromaBlock_H264_8u_C2P2R`, `ippiInterpolateChromaBlock_H264_8u_C2C2R`, and `ippiInterpolateChromaBlock_H264_16u_P2R` are declared in the `ippvc.h` file. These functions perform interpolation (convolution with 2x2 kernel) for motion estimation of the chrominance component in accordance with 8.4.2.2.2 of [JVTG050]. Unlike [InterpolateChroma\\_H264](#), the functions use an entire motion vector when they calculate fractional and integer parts of the vectors. The functions also handle cases of overlapping, when the source block pointed to by the motion vector lies out of the source frame. Non-existing samples are cloned from the nearest existing samples.

## Interpolation of Chrominance Component Block for Motion Estimation



`ippiInterpolateChromaBlock_H264_8u_C2P2R` and `ippiInterpolateChromaBlock_H264_8u_C2C2R` are clones of `ippiInterpolateChromaBlock_H264_8u_P2R` but the source image is a chrominance part of an NV12 plane:

```
YY YY YY YY
YY YY YY YY
UV UV UV UV - chrominance part of NV12 plane
```

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if at least one of the specified pointers is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <code>roi.width</code> or <code>roi.height</code> take values other than 16, 8, 4, or 2.

### Example

The code example below shows how to use `ippiInterpolateChromaBlock_H264_8u_P2R` function.

```
{
    IppVCInterpolateBlock_8u interpolateInfo;

    // get reference frame & pitch
    interpolateInfo.pSrc[0] = reference_chroma_u_frame;
    interpolateInfo.pSrc[1] = reference_chroma_v_frame;
    interpolateInfo.srcStep = reference_chroma_frame_pitch;
```

```

// save vector
interpolateInfo.pointVector.x = horizontal_motion_vector;
interpolateInfo.pointVector.y = vertical_motion_vector;

// scale motion vector for chroma
interpolateInfo.pointVector.x <= ((Ipp32s) (3 <= color_format));
interpolateInfo.pointVector.y <= ((Ipp32s) (2 <= color_format));

// fill parameters
interpolateInfo.pDst[0] = destination_chroma_u_block;
interpolateInfo.pDst[1] = destination_chroma_v_block;
interpolateInfo.dstStep = destination_chroma_frame_pitch;
interpolateInfo.sizeFrame = {width_of_chroma_frame, height_of_chroma_frame};
interpolateInfo.sizeBlock = {width_of_chroma_block, height_of_chroma_block};
interpolateInfo.pointBlockPos.x = x_position_of_current_chroma_block;
interpolateInfo.pointBlockPos.y = y_position_of_current_chroma_block;

ippiInterpolateChromaBlock_H264_8u_P2R(&interpolateInfo);
}

```

The code example below shows how to use `ippiInterpolateChromaBlock_H264_8u_C2P2R` function.

```

{
    IppVCInterpolateBlock_8u interpolateInfo;

    // get reference frame & pitch
    interpolateInfo.pSrc[0] = reference_chroma_uv_frame;
    interpolateInfo.pSrc[1] = NULL; /* not used by the function */
    interpolateInfo.srcStep = reference_chroma_frame_uv_pitch;

    // save vector
    interpolateInfo.pointVector.x = horizontal_motion_vector;
    interpolateInfo.pointVector.y = vertical_motion_vector;

    // scale motion vector for chroma
    interpolateInfo.pointVector.x <= ((Ipp32s) (3 <= color_format));
    interpolateInfo.pointVector.y <= ((Ipp32s) (2 <= color_format));

    // fill parameters
    interpolateInfo.pDst[0] = destination_chroma_u_block;
    interpolateInfo.pDst[1] = destination_chroma_v_block;
    interpolateInfo.dstStep = destination_chroma_frame_pitch;
    interpolateInfo.sizeFrame = {width_of_chroma_frame, height_of_chroma_frame};
    interpolateInfo.sizeBlock = {width_of_chroma_block, height_of_chroma_block};
    interpolateInfo.pointBlockPos.x = x_position_of_current_chroma_block;
    interpolateInfo.pointBlockPos.y = y_position_of_current_chroma_block;
    ippiInterpolateChromaBlock_H264_8u_C2P2R(&interpolateInfo);
}

```

## InterpolateBlock\_H264, Bidir\_H264

*Calculate average value for each source pair values in block.*

---

### Syntax

```

IppStatus ippiInterpolateBlock_H264_8u_P2P1R(const Ipp8u* pSrc1, const Ipp8u* pSrc2, Ipp8u*
pDst, Ipp32u width, Ipp32u height, Ipp32u pitch);

```

```

IppStatus ippiInterpolateBlock_H264_8u_P3P1R(const Ipp8u* pSrc1, const Ipp8u* pSrc2, Ipp8u*
pDst, Ipp32u width, Ipp32u height, Ipp32s srcStep1, Ipp32s srcStep2, Ipp32s dstStep);

IppStatus ippiBidir_H264_16u_P2P1R(IppVCBidir_16u* bidirInfo);

```

## Parameters

<i>pSrc1</i>	Pointer to the first input block.
<i>pSrc2</i>	Pointer to the second input block.
<i>pDst</i>	Pointer to the first output block.
<i>width</i>	Number of values in the block line.
<i>height</i>	Number of lines in the block.
<i>srcStep1</i>	Distance in items between starts of the consecutive lines in the first input block.
<i>srcStep2</i>	Distance in items between starts of the consecutive lines in the second input block.
<i>dstStep</i>	Distance in items between starts of the consecutive lines in the first output block.
<i>pitch</i>	Memory pitch.
<i>bidirInfo</i>	Pointer to the <code>IppVCBidir_16u</code> structure.

## Description

The functions `ippiInterpolateBlock_H264_8u_P2P1R`, `ippiInterpolateBlock_H264_8u_P3P1R`, and `ippiBidir_H264_16u_P2P1R` are declared in the `ippvc.h` file. These functions calculate the average value for each corresponding source pair values in a block with height lines and width values per line.

$$pDst[i] = \frac{pSrc1[i] + pSrc2[i] + 1}{2}.$$

These functions are used in the H.264 decoder included into Intel IPP Samples. See [introduction](#) to the H.264 section.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if at least one of the specified pointers is <code>NULL</code> .

## WeightedAverage\_H264

*Averages two blocks with weights.*

### Syntax

```
ippStatus ippiWeightedAverage_H264_8u_C1IR(const Ipp8u* pSrc1, Ipp8u* pSrc2Dst, Ipp32s
srcDstStep, Ipp32s weight1, Ipp32s weight2, Ipp32s shift, Ipp32s offset, IppiSize roiSize);
```

### Parameters

<i>pSrc1</i>	Pointer to the first source (output of preceding functions).
<i>pSrc2Dst</i>	Pointer to the second source and result.
<i>srcDstStep</i>	Distance in bytes between starts of the consecutive lines in the source and result images.
<i>weight1, weight2</i>	Weights.
<i>shift</i>	Shift.
<i>offset</i>	Offset.
<i>roiSize</i>	Flag that specifies the region of interest (could be 16, 8, 4 or 2 in each dimension).

### Description

The function `ippiWeightedAverage_H264_8u_C1IR` is declared in the `ippvc.h` file. This function averages two blocks with weights (for weighted bi-directional predictions) as specified in 8.4.2.3.2 of [JVTG050] when `predFlagL0` and `predFlagL1` are equal to 1.

The function uses the following formula:

$$\begin{aligned} \text{Source2}[x, y] = \\ \text{Clip1}((\text{weight1} * \text{Source1}[x, y] + \text{weight2} * \text{Source2}[x, y] + 1 \ll (\text{shift} - 1)) \gg \text{shift} + \text{offset}) \end{aligned}$$

where  $x \in [0, \text{roi.width}-1]$ ,  $y \in [0, \text{roi.height}-1]$

$$\text{Clip1}(z) = \begin{cases} 0 & ; z < 0 \\ 255 & ; z > 255 \\ z & ; \text{otherwise} \end{cases}$$

The above formula corresponds to formula 8-220 of JVT-G050. See [Table "Color Conversion Functions"](#) for matching the function arguments and the standard variables.

### WeightedAverage\_H264 Arguments vs JTV-G050 Variables

<b>ippiWeightedAverage</b>	<b>8.4.2.3.2 JVT-G050</b>
<i>weight1</i>	$w_0$

<b>ippiWeightedAverage</b>	<b>8.4.2.3.2 JVT-G050</b>
<i>weight2</i>	$w_1$
<i>shift</i>	$\log_{2}WD + 1$
<i>offset</i>	$(o_0 + o_1 + 1) \gg 1$
<i>roi.width</i>	<i>partWidth</i>
<i>roi.height</i>	<i>partHeight</i>

Values of  $w_0$ ,  $w_1$ ,  $\log_{2}WD$ ,  $o_0$ ,  $o_1$  are calculated from formulas 8-221 through 8-244.

This function is used in the H.264 decoder included into Intel IPP Samples. See [introduction](#) to the H.264 section.

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if at least one of the specified pointers is NULL.
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcDstStep</i> value is less than <i>roi.width</i>

## UnidirWeightBlock\_H264, UnidirWeight\_H264

*Weigh source block.*

### Syntax

```

IppStatus ippiUnidirWeightBlock_H264_8u_C1IR(Ipp8u* pSrcDst, Ipp32u srcDstStep, Ipp32u
uLog2wd, Ipp32s iWeight, Ipp32s iOffset, IppiSize roi);

IppStatus ippiUnidirWeightBlock_H264_8u_C2R(IppVCWeightBlock_8u* pIppVCWeightBlock, const
IppVCWeightParams_8u* pIppVCWeightParams);

IppStatus ippiUnidirWeight_H264_16u_IP2P1R(Ipp16u* pSrcDst, Ipp32u srcDstStep, Ipp32u
uLog2wd, Ipp32s iWeight, Ipp32s iOffset, IppiSize roi, Ipp32s bitDepth);

```

### Parameters

<i>pSrcDst</i>	Pointer to the source and the result.
<i>srcDstStep</i>	Distance in items between starts of the consecutive lines in the source and result images.
<i>uLog2wd</i>	$\log_2$ weight denominator.
<i>iWeight</i>	Weight.
<i>iOffset</i>	Offset.
<i>roi</i>	Flag that specifies the region of interest (could be 16, 8, 4 or 2 in each dimension).
<i>bitDepth</i>	Number of bits of the plane sample with range [8.. 14].
<i>pIppVCWeightBlock</i>	Pointer to the structure that holds parameters for both source and destination planes. Only <i>pDst</i> , <i>dstStep</i> , and <i>roiSize</i> are used.
<i>pIppVCWeightParams</i>	Pointer to the structure that holds weighting parameters for both channels (U and V) of the source plane.

## Description

The functions `ippiUnidirWeightBlock_H264_8u_C1IR`, `ippiUnidirWeightBlock_H264_8u_C2R`, and `ippiUnidirWeight_H264_16u_IP2P1R` are declared in the `ippvc.h` file. These functions perform weighted prediction as specified in 8.4.2.3.2 of [ITUH264] when `predFlagL0` is equal to 1 and `predFlagL1` are equal to 0 or when `predFlagL0` is equal to 0 and `predFlagL1` are equal to 1.

`ippiUnidirWeightBlock_H264_8u_C2R` is intended for a chrominance combined plane as in the N12 format.

The functions use the following formula:

```
if(logWD >= 1)
```

```
    pSrcDst[x,y]= ClipC(((pSrcDst[x,y]· iWeight + 2ulog2wd-1) » ulog2wd) + iOffset)
else
    pSrcDst[x,y]= ClipC((pSrcDst[x,y]· iWeight) + iOffset)
```

where  $x \in [0, roi.width-1]$ ,  $y \in [0, roi.height-1]$ ,

$$\text{ClipI}(z) = \begin{cases} 0 & ; z < 0 \\ 255 & ; z > 255 \\ z & ; \text{otherwise} \end{cases}$$

The above formula corresponds to formula 8-270 and 8-271 of ITUH264.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if at least one of the specified pointers is NULL.
<code>ippStsStepErr</code>	Indicates an error condition if <code>srcDstStep</code> value is less than <code>roi.width</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <code>roi.width</code> or <code>roi.height</code> value is not equal to 2 or 4 or 8 or 16.

## BiDirWeightBlock\_H264, BidirWeight\_H264

Average two blocks with two weights and two offsets.

### Syntax

```
IppStatus ippiBiDirWeightBlock_H264_8u_P2P1R(const Ipp8u* pSrc1, const Ipp8u* pSrc2, Ipp8u* pDst, Ipp32u srcStep, Ipp32u dstStep, Ipp32u uLog2wd, Ipp32s iWeight1, Ipp32s iOffset1, Ipp32s iWeight2, Ipp32s iOffset2, IppiSize roi);
```

```
IppStatus ippiBiDirWeightBlock_H264_8u_C2R(IppVCWeightBlock_8u* pIppVCWeightBlock, const IppVCWeightParams_8u* pIppVCWeightParamP1, const IppVCWeightParams_8u* pIppVCWeightParamP2);
```



```
IppStatus ippiBiDirWeight_H264_16u_P2P1R(const IppVCBidir_16u* bidirInfo, Ipp32u ulog2wd,
Ipp32s iWeight1, Ipp32s iOffset1, Ipp32s iWeight2, Ipp32s iOffset2);
```

## Parameters

<i>pSrc1</i>	Pointer to the first source.
<i>pSrc2</i>	Pointer to the second source.
<i>pDst</i>	Pointer to the result.
<i>srcStep</i>	Distance in items between starts of the consecutive lines in the source images.
<i>dstStep</i>	Distance in items between starts of the consecutive lines in the result image.
<i>ulog2wd</i>	log2 weight denominator.
<i>iWeight1, iWeight2</i>	Weights.
<i>iOffset1, iOffset2</i>	Offsets.
<i>roi</i>	Flag that specifies the region of interest (could be 16, 8, 4 or 2 in each dimension).
<i>bidirInfo</i>	Pointer to the <i>IppVCBidir_16u</i> structure.
<i>pIppVCWeightBlock</i>	Pointer to the structure that holds parameters for both source and destination chrominance combined planes.
<i>pIppVCWeightParamP1</i>	Pointer to the structure that holds parameters for both channels (U and V) of the first source plane.
<i>pIppVCWeightParamP2</i>	Pointer to the structure that holds parameters for both channels (U and V) of the second source plane.

## Description

The functions `ippiBiDirWeightBlock_H264_8u_P2P1R`, `ippiBiDirWeightBlock_H264_8u_C2R` and `ippiBiDirWeight_H264_16u_P2P1R` are declared in the `ippvc.h` file. These functions perform weighted prediction as specified in 8.4.2.3.2 of [ITUH264] when both `predFlagL0` and `predFlagL1` are equal to 1.

`ippiBiDirWeightBlock_H264_8u_C2R` is intended for a chrominance combined plane as in the N12 format.

The functions use the following formula:

$$pDst[x,y] = Clip1(((pSrc1[x,y] \cdot iWeight1 + pSrc2[x,y] \cdot iWeight2 + 2^{ulog2wd-1}) \gg (ulog2wd + 1) + ((iOffset1 + iOffset2 + 1) \gg 2))$$

where  $x \in [0, roi.width-1]$ ,  $y \in [0, roi.height-1]$ ,

$$Clip1(z) = \begin{cases} 0 & ; z < 0 \\ 255 & ; z > 255 \\ z & ; \text{otherwise} \end{cases}$$

The above formula corresponds to formula 8-272 of ITUH264.

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if at least one of the specified pointers is NULL.
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> value or <i>dstStep</i> value is less than <i>roi.width</i> .
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roi.width</i> or <i>roi.height</i> value is not equal to 2 or 4 or 8 or 16.

## BiDirWeightBlockImplicit\_H264, BidirWeightImplicit\_H264

Average two blocks with weights if *uLog2wd* is equal to 5.

---

### Syntax

```
IppStatus ippBiDirWeightBlockImplicit_H264_8u_P2PlR(const Ipp8u* pSrc1, const Ipp8u*
pSrc2, Ipp8u* pDst, Ipp32u srcStep, Ipp32u dstStep, Ipp32s iWeight1, Ipp32s iWeight2,
IppiSize roi);
```

```
IppStatus ippBiDirWeightImplicit_H264_16u_P2PlR(const IppVCBDir_16u* bidirInfo, Ipp32s
iWeight1, Ipp32s iWeight2);
```

### Parameters

<i>pSrc1</i>	Pointer to the first source.
<i>pSrc2</i>	Pointer to the second source.
<i>pDst</i>	Pointer to the result.
<i>srcStep</i>	Distance in items between starts of the consecutive lines in the source images.
<i>dstStep</i>	Distance in items between starts of the consecutive lines in the result image.
<i>iWeight1</i> , <i>iWeight2</i>	Weights.
<i>roi</i>	Flag that specifies the region of interest (could be 16, 8, 4 or 2 in each dimension).
<i>bidirInfo</i>	Pointer to the <code>IppVCBDir_16u</code> structure.

### Description

The functions `ippBiDirWeightBlockImplicit_H264_8u_P2PlR` and `ippBiDirWeightImplicit_H264_16u_P2PlR` are declared in the `ippvc.h` file. These functions perform implicit mode weighted prediction as specified in 8.4.2.3.2 of [ITUH264].

These functions use formula 8-272 of [ITUH264], but according to 8-273, 8-274, and 8-275 of ITUH264 the  $\log_2$  weight denominator *uLog2wd* is equal to 5, and both *iOffset1*, *iOffset2* offsets are equal to zero. So the functions use the following formula:

---

```
pDst[x,y]= Clip1c((pSrc1[x,y]·iWeight1+pSrc2[x,y]·iWeight2+32)»6)
```

where  $x \in [0, roi.width-1], y \in [0, roi.height-1],$

$$\text{Clip1}(z) = \begin{cases} 0 & ; z < 0 \\ 255 & ; z > 255 \\ z & ; \text{otherwise} \end{cases}$$

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if at least one of the specified pointers is NULL.
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> value or <i>dstStep</i> value is less than <i>roi.width</i> .
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roi.width</i> or <i>roi.height</i> value is not equal to 2 or 4 or 8 or 16.

## Macroblock Reconstruction

Note that the functions with `_IP2R` descriptor use the following structures:

```
typedef struct _IppiMBReconstructHigh_32s16u
```

```
{
    Ipp32s** ppSrcDstCoeff;
    Ipp16u* pSrcDstPlane;
    Ipp32s srcDstStep;
    Ipp32u cbp;
    Ipp32s qp;
    Ipp16s* pQuantTable;
    Ipp32s bypassFlag;
    Ipp32s bitDepth;
} IppiReconstructHighMB_32s16u;
```

```
typedef struct _IppiReconstructHighMB_16s8u
```

```
{
    Ipp16s** ppSrcDstCoeff;
    Ipp8u* pSrcDstPlane;
    Ipp32s srcDstStep;
    Ipp32u cbp;
    Ipp32s qp;
    Ipp16s* pQuantTable;
    Ipp32s bypassFlag;
} IppiReconstructHighMB_16s8u;
```

with parameters:

*ppSrcDstCoeff*

Pointer to the residual coefficients; pointer is updated by the function.

*pSrcDstPlane*

Pointer to the macroblock that is reconstructed in the current plane.

*srcDstStep*

Distance in items between starts of the consecutive lines in the source/destination images.

*cbp*

Coded block pattern.

- If `cbp & IPPVC_CBP_DC` is not equal to 0, DC block is not zero-filled and it exists in *ppSrcDstCoeff*.
- If `cbp & (1<<(IPPVC_CBP_1ST_AC_BITPOS+i))` is not equal to 0 ( $0 < i < N$ ), *i*-th AC block is not zero-filled and it exists in *ppSrcDstCoeff*.

	The value of $N$ depends on a function. For Luma, $N$ is 16, for chroma - $N$ is 4, and for Chroma422 - $N$ is 8. Sizes of DC/AC blocks depend on a function.
<i>qp</i>	Quantizer with range depending on the function (Luma/Chroma). For <code>_32s16u_IP2R</code> functions, the range depends on <i>bitDepth</i> . It should be $[0 \dots 39 + (6 * ((\text{bitDepth}) - 8))]$ for chroma and $[0 \dots 51 + (6 * ((\text{bitDepth}) - 8))]$ for luma. For <code>_16s8u_IP2R</code> functions, the range equals $[0..39]$ for chroma and $[0..51]$ for luma.
<i>pQuantTable</i>	Quantization table. Pointer to the quantization table for plane ( <i>LevelScale</i> ( <i>qp</i> %6, <i>i</i> , <i>j</i> ) in [ITUH264])
<i>bypassFlag</i>	Flag enabling lossless coding. Lossless coding is enabled only if <i>bypassFlag</i> is 1 and <i>qp</i> is 0. (See <i>qp_prime_y_zero_transform_bypass_flag</i> in [ITUH264])
<i>bitDepth</i>	Number of bits of the plane sample with range $[8..14]$ .

## ReconstructChromaInterMB\_H264

*Reconstructs inter chroma macroblock.*

### Syntax

```
IppStatus ippiReconstructChromaInterMB_H264_16s8u_P2R(Ipp16s** ppSrcCoeff, Ipp8u*
pSrcDstUPlane, Ipp8u* pSrcDstVPlane, const Ipp32u srcDstStep, const Ipp32u cbp4x4, const
Ipp32u ChromaQP);
```

```
IppStatus ippiReconstructChromaInterMB_H264_16s8u_C2R(Ipp16s** ppSrcDstCoeff, Ipp8u*
pSrcDstUPlane, Ipp32s srcDstUVStep, Ipp32u cbp4x4, Ipp32u ChromaQP);
```

### Parameters

<i>ppSrcCoeff</i> , <i>ppSrcDstCoeff</i>	Pointer to the order of 4x4 blocks of residual coefficients for this macroblock, which are taken as a result of Huffman decoding (2x2 DC <i>U</i> -block, 2x2 DC <i>V</i> -block, 4x4 AC <i>U</i> -blocks, 4x4 AC <i>V</i> -blocks if the block is not zero-filled) in the same order as is shown in Figure 8-7 of [JVTG050]. Pointer is updated by the function and points to the blocks for the next macroblock.
<i>pSrcDstUPlane</i>	Pointer to macroblock that is reconstructed in current <i>U</i> plane. This macroblock must contain inter prediction samples.
<i>pSrcDstVPlane</i>	Pointer to macroblock that is reconstructed in current <i>V</i> plane. This macroblock must contain inter prediction samples.
<i>pSrcDstUVPlane</i>	Pointer to current <i>U</i> and <i>V</i> planes in the N12 format, which are reconstructed.
<i>srcDstStep</i>	Plane step.
<i>cbp4x4</i>	Coded block pattern. If <i>cbp4x4</i> & $(1 < (IPPVC\_CBP\_1ST\_CHROMA\_DC\_BITPOS))$ is not equal to 0, 2x2 DC <i>U</i> -block is not zero-filled and exists in <i>ppSrcCoeff</i> . If <i>cbp4x4</i> & $(1 < (IPPVC\_CBP\_1ST\_CHROMA\_DC\_BITPOS + 1))$ is not equal to 0, 2x2 DC <i>V</i> -block is not zero-filled and exists in <i>ppSrcCoeff</i> . If <i>cbp4x4</i>

$\& (1 < (IPPVC\_CBP\_1ST\_CHROMA\_AC\_BITPOS + i))$  is not equal to 0,  $(0 \leq i < 4)$ ,  $i$ -th 4x4 AC  $U$ -block is not zero-filled and exists in  $ppSrcCoeff$ . If  $cbp4x4 \& (1 < (IPPVC\_CBP\_1ST\_CHROMA\_AC\_BITPOS + i + 4))$  is not equal to 0,  $(0 \leq i < 4)$ ,  $i$ -th 4x4 AC  $V$ -block is not zero-filled and exists in  $ppSrcCoeff$ .

*ChromaQP*

chroma quantizer ( $QP_C$  in [JVTG050]). It must be within the range  $[0;39]$ .

## Description

The functions `ippiReconstructChromaInterMB_H264_16s8u_P2R` and `ippiReconstructChromaInterMB_H264_16s8u_C2R` are declared in the `ippvc.h` file.

`ippiReconstructChromaInterMB_H264_16s8u_P2R` reconstructs an intra chroma macroblock (an 8x8  $U$  macroblock and an 8x8  $V$  macroblock).

`ippiReconstructChromaInterMB_H264_16s8u_C2R` reconstructs an intra chroma macroblock (a 16x8  $UV$  macroblock in the N12 chroma format).

Each of the functions works as follows:

- Performs integer inverse transformation and dequantization for 2x2  $U$  DC coefficients and for 2x2  $V$  DC coefficients in accordance with 8.5.7 of [JVTG050] in the same way as `TransformDequantChromaDC_H264` function does.
- Performs scaling, integer inverse transformation, and shift for 4x4 AC  $U$ -blocks and shift for 4x4 AC  $V$ -blocks in accordance with 8.5.8 of [JVTG050] in the same way as `DequantTransformResidual_H264` function does. This process is performed on all 4x4 chroma blocks in the same order as is shown in Figure 8-7 of [JVTG050].
- `ippiReconstructChromaInterMB_H264_16s8u_P2R` performs adding of 8x8 intra prediction blocks and 8x8 residual blocks in accordance with 8-247 of [JVTG050]. `ippiReconstructChromaInterMB_H264_16s8u_C2R` performs adding of 16x8 intra prediction blocks and 16x8 residual blocks in accordance with 8-247 of [JVTG050].

These functions are used in the H.264 decoder included into Intel IPP Samples. See [introduction](#) to the H.264 section.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if at least one of the specified pointers is NULL.
<code>ippStsOutOfRangeErr</code>	<i>ChromaQP</i> is less than 0 or greater than 39.
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcDstUVStep</i> is less than 0.

## ReconstructChromaIntraHalvesMB\_H264

*Reconstructs two halves of intra chroma macroblock.*

### Syntax

```

IppStatus ippiReconstructChromaIntraHalvesMB_H264_16s8u_P2R(Ipp16s** ppSrcCoeff, Ipp8u*
pSrcDstUPlane, Ipp8u* pSrcDstVPlane, Ipp32u srcDstUVStep, IppIntraChromaPredMode_H264
intraChromaMode, Ipp32u cbp4x4, Ipp32u ChromaQP, Ipp8u edgeTypeTop, Ipp8u edgeTypeBottom);

IppStatus ippiReconstructChromaIntraHalvesMB_H264_16s8u_C2R(Ipp16s** ppSrcDstCoeff, Ipp8u*
pSrcDstUPlane, Ipp32u srcDstUVStep, IppIntraChromaPredMode_H264 intraChromaMode, Ipp32u
cbp4x4, Ipp32u ChromaQP, Ipp8u edgeTypeTop, Ipp8u edgeTypeBottom);

```

### Parameters

<i>ppSrcCoeff, ppSrcDstCoeff</i>	Pointer to the order of 4x4 blocks of residual coefficients for this macroblock, which are taken as a result of Huffman decoding (2x2 DC <i>U</i> -block, 2x2 DC <i>V</i> -block, 4x4 AC <i>U</i> -blocks, 4x4 AC <i>V</i> -blocks if the block is not zero-filled) in the same order as is shown in Figure 8-7 of [JVTG050]. Pointer is updated by the function and points to the blocks for the next macroblock.
<i>pSrcDstUPlane</i>	Pointer to current <i>U</i> plane that is reconstructed.
<i>pSrcDstVPlane</i>	Pointer to current <i>V</i> plane that is reconstructed.
<i>pSrcDstUVPlane</i>	Pointer to current <i>U</i> and <i>V</i> planes in the N12 format, which are reconstructed.
<i>srcDstUVStep</i>	Plane step.
<i>intraChromaMode</i>	Prediction mode of the Intra_chroma prediction process for chroma samples. It is defined in the same way as in <a href="#">PredictIntraChroma8x8_H264</a> function.
<i>cbp4x4</i>	Coded block pattern. If <i>cbp4x4</i> & (1<<(IPPVC_CBP_1ST_CHROMA_DC_BITPOS)) is not equal to 0, 2x2 DC <i>U</i> -block is not zero-filled and exists in <i>ppSrcCoeff</i> . If <i>cbp4x4</i> & (1<<(IPPVC_CBP_1ST_CHROMA_DC_BITPOS+ 1)) is not equal to 0, 2x2 DC <i>U</i> -block is not zero-filled and exists in <i>ppSrcCoeff</i> . If <i>cbp4x4</i> & (1<<(IPPVC_CBP_1ST_CHROMA_AC_BITPOS+ <i>i</i> )) is not equal to 0, (0 ≤ <i>i</i> < 4), <i>i</i> -th 4x4 AC <i>U</i> -block is not zero-filled and exists in <i>ppSrcCoeff</i> . If <i>cbp4x4</i> & (1<<(IPPVC_CBP_1ST_CHROMA_AC_BITPOS+ <i>i</i> +4)) is not equal to 0, (0 ≤ <i>i</i> < 4), <i>i</i> -th 4x4 AC <i>U</i> -block is not zero-filled and exists in <i>ppSrcCoeff</i> .
<i>ChromaQP</i>	Chroma quantizer ( $Q_{PC}$ in [JVTG050]). It must be within the range [0;39].
<i>edgeTypeTop, edgeTypeBottom</i>	Flags that specify the availability of the macroblocks used for prediction. Upper and lower halves of the macroblock may have different prediction vectors, so use two flags: one for the upper half of the macroblock, the other - for the lower half.

## Description

The functions `ippiReconstructChromaIntraHalvesMB_H264_16s8u_P2R` and `ippiReconstructChromaIntraHalvesMB_H264_16s8u_C2R` are declared in the `ippvc.h` file.

`ippiReconstructChromaIntraHalvesMB_H264_16s8u_P2R` reconstructs an intra chroma macroblock (8x8 *U* macroblock and 8x8 *V* macroblock), divided into upper and lower halves, 16x8 each, differing only in prediction, that is, having independent prediction vectors. Otherwise their performance is the same.

`ippiReconstructChromaIntraHalvesMB_H264_16s8u_C2R` reconstructs an intra chroma macroblock in the N12 format (16x8 *UV*) divided into upper and lower halves, 16x4 each, differing only in prediction, that is, having independent prediction vectors. Otherwise their performance is the same.

Each of the functions works as follows:

- Performs integer inverse transformation and dequantization for 2x2 *U* DC coefficients and for 2x2 *V* DC coefficients in accordance with 8.5.7 of [JVTG050] in the same way as `TransformDequantChromaDC_H264` function does.
- Performs scaling, integer inverse transformation, and shift for 4x4 AC *U*-blocks and shift for 4x4 AC *V*-blocks in accordance with 8.5.8 of [JVTG050] in the same way as `DequantTransformResidual_H264` function does. This process is performed on all 4x4 chroma blocks in the same order as is shown in Figure 8-7 of [JVTG050].
- `ippiReconstructChromaIntraHalvesMB_H264_16s8u_P2R` performs intra prediction for an 8x8 *U*-component and for an 8x8 *V*-component in accordance with 8.3.2 of [JVTG050] in the same way as `PredictIntraChroma8x8_H264` function does. `ippiReconstructChromaIntraHalvesMB_H264_16s8u_C2R` performs intra prediction for a 16x8 *UV*-component in accordance with 8.3.2 of [JVTG050].
- `ippiReconstructChromaIntraHalvesMB_H264_16s8u_P2R` performs adding of 8x8 intra prediction blocks and 8x8 residual blocks in accordance with 8-247 of [JVTG050]. `ippiReconstructChromaIntraHalvesMB_H264_16s8u_C2R` performs adding of 16x8 intra prediction blocks and 16x8 residual blocks in accordance with 8-247 of [JVTG050].

These functions are used in the H.264 decoder included into Intel IPP Samples. See [introduction](#) to the H.264 section.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if at least one of the specified pointers is <code>NULL</code> .
<code>ippStsOutOfRangeErr</code>	<i>ChromaQP</i> is less than 0 or greater than 39.
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcDstUVStep</i> is less than 0.

## ReconstructChromaIntraMB\_H264

*Reconstructs intra chroma macroblock.*

### Syntax

```
ippStatus ippiReconstructChromaIntraMB_H264_16s8u_P2R(Ipp16s** ppSrcCoeff, Ipp8u*
pSrcDstUPlane, Ipp8u* pSrcDstVPlane, const Ipp32u srcDstUVStep, const
IppIntraChromaPredMode_H264 intraChromaMode, const Ipp32u cbp4x4, const Ipp32u ChromaQP,
const Ipp8u edgeType);
```



```
IppStatus ippiReconstructChromaIntraMB_H264_16s8u_C2R(Ipp16s** ppSrcDstCoeff, Ipp8u*
pSrcDstUVPlane, Ipp32s srcDstUVStep, IppIntraChromaPredMode_H264 intraChromaMode, Ipp32u
cbp4x4, Ipp32u ChromaQP, Ipp32u edgeType);
```

## Parameters

<i>ppSrcCoeff</i> , <i>ppSrcDstCoeff</i>	Pointer to the order of 4x4 blocks of residual coefficients for this macroblock, which are taken as a result of Huffman decoding (2x2 DC <i>U</i> -block, 2x2 DC <i>V</i> -block, 4x4 AC <i>U</i> -blocks, 4x4 AC <i>V</i> -blocks if the block is not zero-filled) in the same order as shown in Figure 8-7 of [JVTG050]. Pointer is updated by the function and points to the blocks for the next macroblock.
<i>pSrcDstUPlane</i>	Pointer to current <i>U</i> plane that is reconstructed.
<i>pSrcDstVPlane</i>	Pointer to current <i>V</i> plane that is reconstructed.
<i>pSrcDstUVPlane</i>	Pointer to current <i>U</i> and <i>V</i> planes in the N12 format, which are reconstructed.
<i>srcDstUVStep</i>	Plane step.
<i>intraChromaMode</i>	Prediction mode of the Intra_chroma prediction process for chroma samples. It is defined in the same way as in <a href="#">PredictIntraChroma8x8_H264</a> function.
<i>cbp4x4</i>	Coded block pattern. If <i>cbp4x4</i> & (1<<(IPPVC_CBP_1ST_CHROMA_DC_BITPOS)) is not equal to 0, 2x2 DC <i>U</i> -block is not zero-filled and exists in <i>ppSrcCoeff</i> . If <i>cbp4x4</i> & (1<<(IPPVC_CBP_1ST_CHROMA_DC_BITPOS+ 1)) is not equal to 0, 2x2 DC <i>U</i> -block is not zero-filled and exists in <i>ppSrcCoeff</i> . If <i>cbp4x4</i> & (1<<(IPPVC_CBP_1ST_CHROMA_AC_BITPOS+ <i>i</i> )) is not equal to 0, (0 ≤ <i>i</i> < 4), <i>i</i> -th 4x4 AC <i>U</i> -block is not zero-filled and exists in <i>ppSrcCoeff</i> . If <i>cbp4x4</i> & (1<<(IPPVC_CBP_1ST_CHROMA_AC_BITPOS+ <i>i</i> +4)) is not equal to 0, (0 ≤ <i>i</i> < 4), <i>i</i> -th 4x4 AC <i>U</i> -block is not zero-filled and exists in <i>ppSrcCoeff</i> .
<i>ChromaQP</i>	Chroma quantizer ( <i>Q<sub>P</sub></i> in [JVTG050]). It must be within the range [0;39].
<i>edgeType</i>	Flag that specifies the availability of the macroblocks used for prediction. If the upper macroblock is not available for 16x16 intra prediction, <i>edgeType</i> &IPPVC_TOP_EDGE must be non-zero. If the left macroblock is not available for 16x16 intra prediction, <i>edgeType</i> &IPPVC_LEFT_EDGE must be non-zero. If the upper-left macroblock is not available for 16x16 intra prediction, <i>edgeType</i> &IPPVC_TOP_LEFT_EDGE must be non-zero.

## Description

The functions `ippiReconstructChromaIntraMB_H264_16s8u_P2R` and `ippiReconstructChromaIntraMB_H264_16s8u_C2R` are declared in the `ippvc.h` file.

`ippiReconstructChromaIntraMB_H264_16s8u_P2R` reconstructs an intra chroma macroblock (an 8x8 *U* macroblock and an 8x8 *V* macroblock).

`ippiReconstructChromaIntraMB_H264_16s8u_C2R` reconstructs an intra chroma macroblock (a 16x8 *UV* macroblock in the N12 chroma format).

Each of the functions works as follows:

- Performs integer inverse transformation and dequantization for 2x2 *U* DC coefficients and for 2x2 *V* DC coefficients in accordance with 8.5.7 of [JVTG050] in the same way as [TransformDequantChromaDC\\_H264](#) function does.
- Performs scaling, integer inverse transformation, and shift for 4x4 AC *U*-blocks and shift for 4x4 AC *V*-blocks in accordance with 8.5.8 of [JVTG050] in the same way as [DequantTransformResidual\\_H264](#) function does. This process is performed on all 4x4 chroma blocks in the same order as is shown in Figure 8-7 of [JVTG050].
- `ippiReconstructChromaIntraMB_H264_16s8u_P2R` performs intra prediction for an 8x8 *U*-component and for an 8x8 *V*-component in accordance with 8.3.2 of [JVTG050] in the same way as [PredictIntraChroma8x8\\_H264](#) function does. `ippiReconstructChromaIntraMB_H264_16s8u_C2R` performs intra prediction for a 16x8 *UV*-component in accordance with 8.3.2 of [JVTG050].
- `ippiReconstructChromaIntraMB_H264_16s8u_P2R` performs adding of 8x8 intra prediction blocks and 8x8 residual blocks in accordance with 8-247 of [JVTG050]. `ippiReconstructChromaIntraMB_H264_16s8u_C2R` performs adding of 16x8 intra prediction blocks and 16x8 residual blocks in accordance with 8-247 of [JVTG050].

These functions are used in the H.264 decoder included into Intel IPP Samples. See [introduction](#) to the H.264 section.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if at least one of the specified pointers is NULL.
<code>ippStsOutOfRangeErr</code>	<i>ChromaQP</i> is less than 0 or greater than 39.
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcDstUVStep</i> is less than 0.

## ReconstructChromaInter4x4MB\_H264, ReconstructChromaInter4x4\_H264High

Reconstruct 4X4 inter chroma macroblock for high profile.

### Syntax

```

IppStatus ippiReconstructChromaInter4x4MB_H264_16s8u_P2R(Ipp16s** ppSrcDstCoeff, Ipp8u*
pSrcDstUPlane, Ipp8u* pSrcDstVPlane, Ipp32u srcDstUVStep, Ipp32u cbp4x4, Ipp32s ChromaQPU,
Ipp32u ChromaQPV, const Ipp16s* pQuantTableU, const Ipp16s* pQuantTableV, Ipp8u bypassFlag);

IppStatus ippiReconstructChromaInter4x4MB_H264_16s8u_C2R(Ipp16s** ppSrcDstCoeff, Ipp8u*
pSrcDstUPlane, Ipp8u* pSrcDstVPlane, Ipp32u srcDstUVStep, Ipp32u cbp4x4, Ipp32s ChromaQPU, Ipp32u ChromaQPV,
const Ipp16s* pQuantTableU, const Ipp16s* pQuantTableV, Ipp8u bypassFlag);

IppStatus ippiReconstructChromaInter4x4_H264High_32s16u_IP2R(const
IppiReconstructHighMB_32s16u* pReconstructInfo[2]);

```

## Parameters

<i>ppSrcDstCoeff</i>	Pointer to the order of 4x4 blocks of residual coefficients for this macroblock, which are taken as a result of Huffman decoding (2x2 DC <i>U</i> -block, 2x2 DC <i>V</i> -block, 4x4 AC <i>U</i> -blocks, 4x4 AC <i>V</i> -blocks if the block is not zero-filled) in the same order as is shown in Figure 8-7 of [ITUH264]. Pointer is updated by the function and points to the blocks for the next macroblock.
<i>pSrcDstUPlane</i>	Pointer to a macroblock that is reconstructed in the current <i>U</i> plane. This macroblock must contain inter prediction samples.
<i>pSrcDstVPlane</i>	Pointer to a macroblock that is reconstructed in the current <i>V</i> plane. This macroblock must contain inter prediction samples.
<i>pSrcDstUVPlane</i>	Pointer to a macroblock that is reconstructed in the current <i>UV</i> plane.
<i>srcDstUVStep</i>	Plane step.
<i>cbp4x4</i>	Coded block pattern. If <i>cbp4x4</i> & (1<<(IPPVC_CBP_1ST_CHROMA_DC_BITPOS)) is not equal to 0, 2x2 DC <i>U</i> -block is not zero-filled and exists in <i>ppSrcDstCoeff</i> . If <i>cbp4x4</i> & (1<<(IPPVC_CBP_1ST_CHROMA_DC_BITPOS+ 1)) is not equal to 0, 2x2 DC <i>U</i> -block is not zero-filled and exists in <i>ppSrcDstCoeff</i> . If <i>cbp4x4</i> & (1<<(IPPVC_CBP_1ST_CHROMA_AC_BITPOS+ <i>i</i> )) is not equal to 0, (0 ≤ <i>i</i> < 4), <i>i</i> -th 4x4 AC <i>U</i> -block is not zero-filled and exists in <i>ppSrcDstCoeff</i> . If <i>cbp4x4</i> & (1<<(IPPVC_CBP_1ST_CHROMA_AC_BITPOS+ <i>i</i> +4)) is not equal to 0, (0 ≤ <i>i</i> < 4), <i>i</i> -th 4x4 AC <i>U</i> -block is not zero-filled and exists in <i>ppSrcDstCoeff</i> .
<i>ChromaQPU</i>	Chroma quantizer for <i>U</i> plane ( $QP_C$ in [ITUH264]). It must be within the range [0;39].
<i>ChromaQPV</i>	Chroma quantizer for <i>V</i> plane ( $QP_C$ in [ITUH264]). It must be within the range [0;39].
<i>pQuantTableU</i>	Pointer to the quantization table for <i>U</i> plane ( <i>LevelScale</i> ( <i>qP</i> %6, <i>i</i> , <i>j</i> ) in [ITUH264]).
<i>pQuantTableV</i>	Pointer to the quantization table for <i>V</i> plane ( <i>LevelScale</i> ( <i>qP</i> %6, <i>i</i> , <i>j</i> ) in [ITUH264]).
<i>bypassFlag</i>	Flag enabling lossless coding.
<i>pReconstructInfo</i> [2]	Pointer to the <i>IppiReconstructHighMB_32s16u</i> structure. The first pointer <i>pReconstructInfo</i> [0] points to the structure with parameters for restructuring chroma macroblock of plane <i>U</i> , and the second pointer <i>pReconstructInfo</i> [1] points to the structure with parameters for restructuring Chroms macroblock of plane <i>V</i> .

## Description

The function *ippiReconstructChromaInter4x4MB\_H264\_16s8u\_P2R*, *ippiReconstructChromaInter4x4MB\_H264\_16s8u\_C2R*, and *ippiReconstructChromaInter4x4\_H264High\_32s16u\_IP2R* are declared in the *ippvc.h* file.

`ippiReconstructChromaInter4x4MB_H264_16s8u_C2R` is intended for a chrominance combined plane as in the N12 format (UVUVUVUV...).

These functions reconstruct 4x4 inter chroma macroblock (8x8 *U* macroblock and 8x8 *V* macroblock) for high profile:

- Perform integer inverse transformation and dequantization for 2x2 *U* DC coefficients and for 2x2 *V* DC coefficients in accordance with 8.5.9 of [ITUH264] according to 8-323 when `chroma_format_idc` is equal to 1.
- Perform scaling, integer inverse transformation, and shift for 4x4 AC *U*-blocks and shift for 4x4 AC *V*-blocks in accordance with 8.5.10 of [ITUH264]. This process is performed on all 4x4 chroma blocks in the same order as is shown in Figure 8-7(a) of [ITUH264] when `chroma_format_idc` is equal to 1.
- Perform adding of 4x4 inter prediction blocks and 4x4 residual blocks in accordance with 8-306 of [ITUH264].

These functions are used in the H.264 decoder included into Intel IPP Samples. See [introduction](#) to the H.264 section.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if at least one of the specified pointers is NULL.
<code>ippStsOutOfRangeErr</code>	<i>ChromaQPU</i> or <i>ChromaQPV</i> is less than 0 or greater than 39.

## Example

The code example below shows how to use `ReconstructChromaInter4x4MB_H264` function.

```
{
#define D_CBP_CHROMA_DC 0x00001
#define D_CBP_CHROMA_AC 0x1fffe
#define D_CBP_CHROMA_AC_420 0x0001e
#define D_CBP_1ST_CHROMA_DC_BITPOS 17
#define D_CBP_1ST_CHROMA_AC_BITPOS 19

Ipp8u  planeU[8*8] =
{ 0x68, 0xaf, 0x16, 0x9a, 0x38, 0x8f, 0x7f, 0x6a,
  0x7e, 0x08, 0x01, 0x8c, 0x76, 0x87, 0xa4, 0xc6,
  0x9d, 0x47, 0x0f, 0xa1, 0x90, 0x0b, 0x36, 0x47,
  0x87, 0x28, 0x0f, 0xb3, 0x24, 0xa4, 0x5c, 0x81,
  0x4b, 0x96, 0x33, 0xa3, 0x9f, 0x25, 0x5c, 0x44,
  0x61, 0x92, 0x36, 0x9c, 0x07, 0x05, 0x82, 0x9b,
  0xc7, 0xb7, 0x99, 0x48, 0x33, 0x9a, 0x75, 0x03,
  0xb6, 0x53, 0x9d, 0x03, 0x9c, 0x3a, 0x0b, 0x59 };

Ipp8u  planeV[8*8] = {
  0x94, 0x22, 0x8a, 0xbc, 0xbb, 0x23, 0x60, 0xbb,
  0xba, 0x73, 0x65, 0x14, 0x9a, 0xc6, 0x34, 0xae,
  0xaf, 0x16, 0x22, 0x9a, 0x38, 0x6c, 0x54, 0x8f,
  0x08, 0x01, 0xb9, 0x8c, 0x76, 0x7b, 0x16, 0x87,
  0x47, 0x0f, 0x9f, 0xa1, 0x90, 0x5f, 0x1d, 0x0b,
  0x28, 0x0f, 0x4a, 0xb3, 0x24, 0x83, 0x14, 0xa4,
  0x96, 0x33, 0x65, 0xa3, 0x9f, 0x3b, 0x08, 0x25,
  0x59, 0x2c, 0xb6, 0x30, 0x24, 0x7b, 0x47, 0x6b };

Ipp32s  stepUV = 8;
```

```

Ipp16s quantTableU[16] = {22,12,34,50,34,33,46,21,18,19,34,28,28,31,15,3};
Ipp16s quantTableV[16] = {32,50,34, 2,46,21,18, 9,34,28,28,31,15,3,12,44};

Ipp16s coef[4+16*4+4+16*4] = {
0, 0, 0, 0,
0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, -1, 0,
1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1,
0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, -1, 0, 1,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
-3, -3, -3, -3,
-1, -1, -1, -1, 4, 4, 4, 4, -6, -6, -6, -6, -6, -6, -6, -6,
-6, -6, -6, -6, -6, -6, -6, -6, -64, -64, -64, -64, 7, 1, -4, -3,
7, 1, -4, -3, 2, -2, -2, 2, -1, 5, 10, 11, 11, 15, 15, 9,
-9, -10, -13, 12, -7, -7, -8, 12, -3, -3, 3, 13, -1, -1, 8, 14 };

Ipp32u cbpU = 0x00000014, cbpV = 0x00000014;
Ipp32s QPU = 17, QPV = 31;
Ipp8u bypass_flag = 0;
IppStatus result;
Ipp16s* pCoef;
Ipp32u cbp4x4 = (((cbpU & D_CBP_CHROMA_DC) | ((cbpV & D_CBP_CHROMA_DC) <<
1)) << D_CBP_1ST_CHROMA_DC_BITPOS) | ((cbpU & D_CBP_CHROMA_AC_420) <<
(D_CBP_1ST_CHROMA_AC_BITPOS - 1)) | ((cbpV & D_CBP_CHROMA_AC_420) <<
(D_CBP_1ST_CHROMA_AC_BITPOS + 4 - 1));

pCoef = coef;

result = ippiReconstructChromaInter4x4MB_H264_16s8u_P2R(
&pCoef,
planeU, planeV,
stepUV,
cbp4x4,
QPU,QPV,
quantTableU,quantTableV,
bypass_flag);
}

```

## ReconstructChroma422Inter4x4\_H264High

*Reconstructs 4X4 inter chroma macroblock for 4:2:2 chroma mode.*

### Syntax

```

IppStatus ippiReconstructChroma422Inter4x4_H264High_32s16u_IP2R(const
IppiReconstructHighMB_32s16u* pReconstructInfo[2], Ipp32u levelScaleDCU, Ipp32u
levelScaleDCV);

IppStatus ippiReconstructChroma422Inter4x4_H264High_16s8u_IP2R(const
IppiReconstructHighMB_16s8u* pReconstructInfo[2], Ipp32u levelScaleDCU, Ipp32u
levelScaleDCV);

```

### Parameters

*ppSrcDstCoef*      Pointer to the order of 4x4 blocks of residual coefficients for this macroblock, which are taken as a result of Huffman decoding (2x2 DC U-block, 2x2 DC V-block, 4x4 AC U-blocks, 4x4 AC V-blocks if the block

	is not zero-filled) in the same order as is shown in Figure 8-7(b) of [ITUH264]. Pointer is updated by the function and points to the blocks for the next macroblock.
<i>pSrcDstUPlane</i>	Pointer to macroblock that is reconstructed in current <i>U</i> plane. This macroblock must contain inter prediction samples.
<i>pSrcDstVPlane</i>	Pointer to macroblock that is reconstructed in current <i>V</i> plane. This macroblock must contain inter prediction samples.
<i>srcDstUVStep</i>	Plane step.
<i>cbp4x4</i>	Coded block pattern. If <i>cbp4x4</i> & (1<<(IPPVC_CBP_1ST_CHROMA_DC_BITPOS)) is not equal to 0, 2x2 DC <i>U</i> -block is not zero-filled and exists in <i>ppSrcDstCoeff</i> . If <i>cbp4x4</i> & (1<<(IPPVC_CBP_1ST_CHROMA_DC_BITPOS+ 1)) is not equal to 0, 2x2 DC <i>U</i> -block is not zero-filled and exists in <i>ppSrcDstCoeff</i> . If <i>cbp4x4</i> & (1<<(IPPVC_CBP_1ST_CHROMA_AC_BITPOS+ <i>i</i> )) is not equal to 0, (0 ≤ <i>i</i> < 4), <i>i</i> -th 4x4 AC <i>U</i> -block is not zero-filled and exists in <i>ppSrcDstCoeff</i> . If <i>cbp4x4</i> & (1<<(IPPVC_CBP_1ST_CHROMA_AC_BITPOS+ <i>i</i> +4)) is not equal to 0, (0 ≤ <i>i</i> < 4), <i>i</i> -th 4x4 AC <i>U</i> -block is not zero-filled and exists in <i>ppSrcDstCoeff</i> .
<i>ChromaQPU</i>	Chroma quantizer for <i>U</i> plane ( <i>QP<sub>C</sub></i> in [ITUH264]). It must be within the range [0;39].
<i>ChromaQPV</i>	Chroma quantizer for <i>V</i> plane ( <i>QP<sub>C</sub></i> in [ITUH264]). It must be within the range [0;39].
<i>levelScaleDCU</i>	Chroma quantizer for <i>U</i> plane ( <i>LevelScale</i> ( <i>QP<sub>C</sub></i> , <i>DC</i> %6, 0, 0)) in [ITUH264]) for chroma DC coefficients.
<i>levelScaleDCV</i>	Chroma quantizer for <i>V</i> plane ( <i>LevelScale</i> ( <i>QP<sub>C</sub></i> , <i>DC</i> %6, 0, 0)) in [ITUH264]) for chroma DC coefficients.
<i>pQuantTableU</i>	Pointer to the quantization table for <i>U</i> plane ( <i>LevelScale</i> ( <i>qP</i> %6, <i>i</i> , <i>j</i> ) in [ITUH264]).
<i>pQuantTableV</i>	Pointer to the quantization table for <i>V</i> plane ( <i>LevelScale</i> ( <i>qP</i> %6, <i>i</i> , <i>j</i> ) in [ITUH264]).
<i>bypassFlag</i>	Flag enabling lossless coding.
<i>pReconstructInfo</i>	[2] Pointer to the <i>IppiReconstructHighMB_32s16u</i> and <i>IppiReconstructHighMB_16s8u</i> structures. The first pointer <i>pReconstructInfo</i> [0] points to the structure with parameters for restructuring chroma macroblock of plane <i>U</i> , and the second pointer <i>pReconstructInfo</i> [1] points to the structure with parameters for restructuring Chroms macroblock of plane <i>V</i> .

## Description

The functions *ippiReconstructChroma422Inter4x4\_H264High\_32s16u\_IP2R* and *ippiReconstructChroma422Inter4x4\_H264High\_16s8u\_IP2R* are declared in the *ippvc.h* file. These functions reconstruct eight 4x4 inter chroma macroblocks (16x8 *U* macroblock and 16x8 *V* macroblock) for chroma format 4:2:2 (*chroma\_format\_idc* is equal to 2):

- Perform integer inverse transformation and dequantization for 2x2  $U$  DC coefficients and for 2x2  $V$  DC coefficients in accordance with 8.5.7 of [ITUH264].
- Perform scaling, integer inverse transformation, and shift for 4x4 AC  $U$ -blocks and shift for 4x4 AC  $V$ -blocks in accordance with 8.5.8 of [ITUH264]. This process is performed on all 4x4 chroma blocks in the same order as is shown in Figure 8-7(b) of [ITUH264].
- Perform adding of 16x8 inter prediction blocks and 16x8 residual blocks in accordance with 8-306 of [ITUH264].

These functions are used in the H.264 decoder included into Intel IPP Samples. See [introduction](#) to the H.264 section.

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if at least one of the specified pointers is NULL.
<code>ippStsOutOfRangeErr</code>	<code>ChromaQPU</code> or <code>ChromaQPV</code> is less than 0 or greater than 39.

## ReconstructChromaIntraHalves4x4MB\_H264, ReconstructChromaIntraHalf4x4\_H264High

*Reconstruct two independent halves of 4X4 intra chroma macroblock for high profile.*

### Syntax

```
ippStatus ippiReconstructChromaIntraHalves4x4MB_H264_16s8u_P2R(Ipp16s** ppSrcDstCoeff,
Ipp8u* pSrcDstUPlane, Ipp8u* pSrcDstVPlane, Ipp32u srcDstUVStep, IppIntraChromaPredMode_H264
intraChromaMode, Ipp32u cbp4x4, Ipp32s ChromaQPU, Ipp32u ChromaQPV, Ipp8u edgeTypeTop,
Ipp8u edgeTypeBottom, const Ipp16s* pQuantTableU, const Ipp16s* pQuantTableV, Ipp8u
bypassFlag);
```

```
ippStatus ippiReconstructChromaIntraHalf4x4_H264High_32s16u_IP2R(const
IppiReconstructHighMB_32s16u* pReconstructInfo[2], IppIntraChromaPredMode_H264
intraChromaMode, Ipp32u edgeTypeTop, Ipp32u edgeTypeBottom);
```

### Parameters

<code>ppSrcDstCoeff</code>	Pointer to the order of 4x4 blocks of residual coefficients for this macroblock, which are taken as a result of Huffman decoding (2x2 DC $U$ -block, 2x2 DC $V$ -block, 4x4 AC $U$ -blocks, 4x4 AC $V$ -blocks if the block is not zero-filled) in the same order as is shown in Figure 8-7 of [ITUH264]. Pointer is updated by the function and points to the blocks for the next macroblock.
<code>pSrcDstUPlane</code>	Pointer to current $U$ plane that is reconstructed.
<code>pSrcDstVPlane</code>	Pointer to current $V$ plane that is reconstructed.
<code>srcDstUVStep</code>	Plane step.
<code>intraChromaMode</code>	Prediction mode of the Intra_chroma prediction process for chroma samples. This mode is defined in the same way as in <a href="#">PredictIntra-Chroma8x8_H264</a> function.

<i>cbp4x4</i>	Coded block pattern. If <i>cbp4x4</i> & $(1 \ll (\text{IPPVC\_CBP\_1ST\_CHROMA\_DC\_BITPOS}))$ is not equal to 0, 2x2 DC <i>U</i> -block is not zero-filled and exists in <i>ppSrcDstCoeff</i> . If <i>cbp4x4</i> & $(1 \ll (\text{IPPVC\_CBP\_1ST\_CHROMA\_DC\_BITPOS} + 1))$ is not equal to 0, 2x2 DC <i>U</i> -block is not zero-filled and exists in <i>ppSrcDstCoeff</i> . If <i>cbp4x4</i> & $(1 \ll (\text{IPPVC\_CBP\_1ST\_CHROMA\_AC\_BITPOS} + i))$ is not equal to 0, $(0 \leq i < 4)$ , <i>i</i> -th 4x4 AC <i>U</i> -block is not zero-filled and exists in <i>ppSrcDstCoeff</i> . If <i>cbp4x4</i> & $(1 \ll (\text{IPPVC\_CBP\_1ST\_CHROMA\_AC\_BITPOS} + i + 4))$ is not equal to 0, $(0 \leq i < 4)$ , <i>i</i> -th 4x4 AC <i>U</i> -block is not zero-filled and exists in <i>ppSrcDstCoeff</i> .
<i>ChromaQPU</i>	Chroma quantizer for <i>U</i> plane ( <i>QP<sub>C</sub></i> in [ITUH264]). It must be within the range [0;39].
<i>ChromaQPV</i>	Chroma quantizer for <i>V</i> plane ( <i>QP<sub>C</sub></i> in [ITUH264]). It must be within the range [0;39].
<i>edgeTypeTop</i> , <i>edgeTypeBottom</i>	Flags that specify the availability of the macroblocks used for prediction. Upper and lower halves of the macroblock may have different prediction vectors, so use two flags: one for the upper half of the macroblock, the other - for the lower half.
<i>pQuantTableU</i>	Pointer to the quantization table for <i>U</i> plane ( <i>LevelScale</i> ( <i>QP<sub>C</sub></i> , <i>DC</i> %6, 0, 0)) in [ITUH264]).
<i>pQuantTableV</i>	Pointer to the quantization table for <i>V</i> plane ( <i>LevelScale</i> ( <i>QP<sub>C</sub></i> , <i>DC</i> %6, 0, 0)) in [ITUH264]).
<i>bypassFlag</i>	Flag enabling lossless coding.
<i>pReconstructInfo</i> [2]	Pointer to the <i>IppiReconstructHighMB_32s16u</i> structure. The first pointer <i>pReconstructInfo</i> [0] points to the structure with parameters for restructuring chroma macroblock of plane <i>U</i> , and the second pointer <i>pReconstructInfo</i> [1] points to the structure with parameters for restructuring Chroms macroblock of plane <i>V</i> .

## Description

The functions *ippiReconstructChromaIntraHalves4x4MB\_H264\_16s8u\_P2R* and *ippiReconstructChromaIntraHalf4x4\_H264High\_32s16u\_IP2R* are declared in the *ippvc.h* file. These functions reconstruct 4x4 intra chroma macroblock (8x8 *U* macroblock and 8x8 *V* macroblock) for high profile. The macroblock is divided into upper and lower halves, 16x8 each. The halves differ only in prediction, that is, have independent prediction vectors. Otherwise their performance is the same:

- Perform integer inverse transformation and dequantization for 2x2 *U* DC coefficients and for 2x2 *V* DC coefficients in accordance with 8.5.9 of [ITUH264] according to 8-323 when *chroma\_format\_idc* is equal to 1.
- Perform scaling, integer inverse transformation, and shift for 4x4 AC *U*-blocks and shift for 4x4 AC *V*-blocks in accordance with 8.5.10 of [ITUH264]. This process is performed on all 4x4 chroma blocks in the same order as shown in Figure 8-7(a) of [ITUH264] when *chroma\_format\_idc* is equal to 1.
- Perform adding of 4x4 inter prediction blocks and 4x4 residual blocks in accordance with 8-306 of [ITUH264].



These functions are used in the H.264 decoder included into Intel IPP Samples. See [introduction](#) to the H.264 section.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if at least one of the specified pointers is NULL.
<code>ippStsOutOfRangeErr</code>	<i>ChromaQPU</i> or <i>ChromaQPV</i> is less than 0 or greater than 39.

## ReconstructChromaIntra4x4MB\_H264, ReconstructChromaIntra4x4\_H264High

Reconstruct 4X4 intra chroma macroblock for high profile.

## Syntax

```
ippStatus ippiReconstructChromaIntra4x4MB_H264_16s8u_P2R(Ipp16s** ppSrcDstCoeff, Ipp8u*
pSrcDstUPlane, Ipp8u* pSrcDstVPlane, Ipp32u srcDstUVStep, IppIntraChromaPredMode_H264
intraChromaMode, Ipp32u cbp4x4, Ipp32s ChromaQPU, Ipp32u ChromaQPV, Ipp8u edgeType, const
Ipp16s* pQuantTableU, const Ipp16s* pQuantTableV, Ipp8u bypassFlag);
```

```
ippStatus ippiReconstructChromaIntra4x4MB_H264_16s8u_C2R(Ipp16s** ppSrcDstCoeff, Ipp8u*
pSrcDstUVPlane, Ipp32u srcDstUVStep, IppIntraChromaPredMode_H264 intraChromaMode, Ipp32u
cbp4x4, Ipp32s ChromaQPU, Ipp32u ChromaQPV, Ipp8u edgeType, const Ipp16s* pQuantTableU,
const Ipp16s* pQuantTableV, Ipp8u bypassFlag);
```

```
ippStatus ippiReconstructChromaIntra4x4_H264High_32s16u_IP2R(const
IppiReconstructHighMB_32s16u* pReconstructInfo[2], IppIntraChromaPredMode_H264
intraChromaMode, Ipp32u edgeType);
```

## Parameters

<i>ppSrcDstCoeff</i>	Pointer to the order of 4x4 blocks of residual coefficients for this macroblock, which are taken as a result of Huffman decoding (2x2 DC <i>U</i> -block, 2x2 DC <i>V</i> -block, 4x4 AC <i>U</i> -blocks, 4x4 AC <i>V</i> -blocks if the block is not zero-filled) in the same order as is shown in Figure 8-7 of [ITUH264]. Pointer is updated by the function and points to the blocks for the next macroblock.
<i>pSrcDstUPlane</i>	Pointer to a macroblock that is reconstructed in the current <i>U</i> plane.
<i>pSrcDstVPlane</i>	Pointer to a macroblock that is reconstructed in the current <i>V</i> plane.
<i>pSrcDstUVPlane</i>	Pointer to a macroblock that is reconstructed in the current <i>UV</i> plane.
<i>srcDstUVStep</i>	Plane step.
<i>intraChromaMode</i>	Prediction mode of the Intra_chroma prediction process for chroma samples. This mode is defined in the same way as in <a href="#">PredictIntraChroma8x8_H264</a> function.
<i>cbp4x4</i>	Coded block pattern. If <i>cbp4x4</i> & (1<<(IPPVC_CBP_1ST_CHROMA_DC_BITPOS)) is not equal to 0, 2x2 DC <i>U</i> -block is not zero-filled and exists in <i>ppSrcDstCoeff</i> . If <i>cbp4x4</i> & (1<<(IPPVC_CBP_1ST_CHROMA_DC_BITPOS+ 1)) is not equal to 0,

	2x2 DC $U$ -block is not zero-filled and exists in <i>ppSrcDstCoeff</i> . If $\text{cbp4x4} \ \& \ (1 < (\text{IPPVC\_CBP\_1ST\_CHROMA\_AC\_BITPOS} + i))$ is not equal to 0, $(0 \leq i < 4)$ , $i$ -th 4x4 AC $U$ -block is not zero-filled and exists in <i>ppSrcDstCoeff</i> . If $\text{cbp4x4} \ \& \ (1 < (\text{IPPVC\_CBP\_1ST\_CHROMA\_AC\_BITPOS} + i + 4))$ is not equal to 0, $(0 \leq i < 4)$ , $i$ -th 4x4 AC $V$ -block is not zero-filled and exists in <i>ppSrcDstCoeff</i> .
<i>ChromaQPU</i>	Chroma quantizer for $U$ plane ( $QP_C$ in [ITUH264]). It must be within the range [0;39].
<i>ChromaQPV</i>	Chroma quantizer for $V$ plane ( $QP_C$ in [ITUH264]). It must be within the range [0;39].
<i>edgeType</i>	Flag that specifies the availability of the macroblocks used for prediction. If the upper macroblock is not available for 16x16 intra prediction, $\text{edgeType} \ \& \ \text{IPPVC\_TOP\_EDGE}$ must be non-zero. If the left macroblock is not available for 16x16 intra prediction, $\text{edgeType} \ \& \ \text{IPPVC\_LEFT\_EDGE}$ must be non-zero. If the upper-left macroblock is not available for 16x16 intra prediction, $\text{edgeType} \ \& \ \text{IPPVC\_TOP\_LEFT\_EDGE}$ must be non-zero.
<i>pQuantTableU</i>	Pointer to the quantization table for $U$ plane ( $\text{LevelScale}(qP\%6, 0, 0)$ ) in [ITUH264]).
<i>pQuantTableV</i>	Pointer to the quantization table for $V$ plane ( $\text{LevelScale}(qP\%6, 0, 0)$ ) in [ITUH264]).
<i>bypassFlag</i>	Flag enabling lossless coding.
<i>pReconstructInfo[2]</i>	Pointer to the <i>IppiReconstructHighMB_32s16u</i> structure. The first pointer <i>pReconstructInfo[0]</i> points to the structure with parameters for restructuring chroma macroblock of plane $U$ , and the second pointer <i>pReconstructInfo[1]</i> points to the structure with parameters for restructuring Chroms macroblock of plane $V$ .

## Description

The functions *ippiReconstructChromaIntra4x4MB\_H264\_16s8u\_P2R*, *ippiReconstructChromaIntra4x4MB\_H264\_16s8u\_C2R*, and *ippiReconstructChromaIntra4x4\_H264High\_32s16u\_IP2R* are declared in the *ippvc.h* file.

*ippiReconstructChromaIntra4x4MB\_H264\_16s8u\_C2R* is intended for a chrominance combined plane as in the N12 format (UVUVUVUV...).

These functions reconstruct 4x4 intra chroma macroblock (8x8  $U$  macroblock and 8x8  $V$  macroblock) for high profile:

- Perform integer inverse transformation and dequantization for 2x2  $U$  DC coefficients and for 2x2  $V$  DC coefficients in accordance with 8.5.9 of [ITUH264] according to 8-323 when *chroma\_format\_idc* is equal to 1.
- Perform scaling, integer inverse transformation, and shift for 4x4 AC  $U$ -blocks and shift for 4x4 AC  $V$ -blocks in accordance with 8.5.10 of [ITUH264]. This process is performed on all 4x4 chroma blocks in the same order as shown in Figure 8-7(a) of [ITUH264] when *chroma\_format\_idc* is equal to 1.
- Perform adding of 4x4 inter prediction blocks and 4x4 residual blocks in accordance with 8-306 of [ITUH264].

These functions are used in the H.264 decoder included into Intel IPP Samples. See [introduction](#) to the H.264 section.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if at least one of the specified pointers is NULL.
<code>ippStsOutOfRangeErr</code>	<i>ChromaQPU</i> or <i>ChromaQPV</i> is less than 0 or greater than 39.

## ReconstructChroma422IntraHalf4x4\_H264High

*Reconstructs two independent halves of 4x4 intra chroma macroblock for 4:2:2 chroma mode.*

## Syntax

```
IppStatus ippiReconstructChroma422IntraHalf4x4_H264High_32s16u_IP2R(const
IppiReconstructHighMB_32s16u* pReconstructInfo[2], IppIntraChromaPredMode_H264
intraChromaMode, Ipp32u edgeTypeTop, Ipp32u edgeTypeBottom, Ipp32u levelScaleDCU, Ipp32u
levelScaleDCV);
```

```
IppStatus ippiReconstructChroma422IntraHalf4x4_H264High_16s8u_IP2R(const
IppiReconstructHighMB_16s8u* pReconstructInfo[2], IppIntraChromaPredMode_H264
intraChromaMode, Ipp32u edgeTypeTop, Ipp32u edgeTypeBottom, Ipp32u levelScaleDCU, Ipp32u
levelScaleDCV);
```

## Parameters

<i>ppSrcDstCoeff</i>	Pointer to the order of 4x4 blocks of residual coefficients for this macroblock, which are taken as a result of Huffman decoding (2x2 DC <i>U</i> -block, 2x2 DC <i>V</i> -block, 4x4 AC <i>U</i> -blocks, 4x4 AC <i>V</i> -blocks if the block is not zero-filled) in the same order as is shown in Figure 8-7(b) of [ITUH264]. Pointer is updated by the function and points to the blocks for the next macroblock.
<i>pSrcDstUPlane</i>	Pointer to the current <i>U</i> plane that is reconstructed.
<i>pSrcDstVPlane</i>	Pointer to the current <i>V</i> plane that is reconstructed.
<i>srcDstUVStep</i>	Plane step.
<i>intraChromaMode</i>	Prediction mode of the Intra_chroma prediction process for chroma samples. This mode is defined in the same way as in <a href="#">PredictIntraChroma8x8_H264</a> function.
<i>cbp4x4</i>	Coded block pattern. If <i>cbp4x4</i> & (1<<(IPPVC_CBP_1ST_CHROMA_DC_BITPOS)) is not equal to 0, 2x2 DC <i>U</i> -block is not zero-filled and exists in <i>ppSrcDstCoeff</i> . If <i>cbp4x4</i> & (1<<(IPPVC_CBP_1ST_CHROMA_DC_BITPOS+ 1)) is not equal to 0, 2x2 DC <i>V</i> -block is not zero-filled and exists in <i>ppSrcDstCoeff</i> . If <i>cbp4x4</i> & (1<<(IPPVC_CBP_1ST_CHROMA_AC_BITPOS+ <i>i</i> )) is not equal to 0, (0 ≤ <i>i</i> < 8), <i>i</i> -th 4x4 AC <i>U</i> -block is not zero-filled and exists

	in <i>ppSrcDstCoeff</i> . If <i>cbp4x4</i> & $(1 < (IPPVC\_CBP\_1ST\_CHROMA\_AC\_BITPOS + i + 4))$ is not equal to 0, $(0 \leq i < 8)$ , <i>i</i> -th 4x4 AC <i>U</i> -block is not zero-filled and exists in <i>ppSrcDstCoeff</i> .
<i>ChromaQPU</i>	Chroma quantizer for <i>U</i> plane ( $QP_C$ in [ITUH264]). It must be within the range [0;39].
<i>ChromaQPV</i>	Chroma quantizer for <i>V</i> plane ( $QP_C$ in [ITUH264]). It must be within the range [0;39].
<i>levelScaleDCU</i>	Chroma quantizer for <i>U</i> plane ( $LevelScale(QP_C, DC \% 6, 0, 0)$ in [ITUH264]) for chroma DC coefficients.
<i>levelScaleDCV</i>	Chroma quantizer for <i>V</i> plane ( $LevelScale(QP_C, DC \% 6, 0, 0)$ in [ITUH264]) for chroma DC coefficients.
<i>edgeTypeTop</i> , <i>edgeTypeBottom</i>	Flags that specify the availability of the macroblocks used for prediction. Upper and lower halves of the macroblock may have different prediction vectors, so use two flags: one for the upper half of the macroblock, the other - for the lower half.
<i>pQuantTableU</i>	Pointer to the quantization table for <i>U</i> plane ( $LevelScale(qP \% 6, i, j)$ in [ITUH264]).
<i>pQuantTableV</i>	Pointer to the quantization table for <i>V</i> plane ( $LevelScale(qP \% 6, i, j)$ in [ITUH264]).
<i>bypassFlag</i>	Flag enabling lossless coding.
<i>pReconstructInfo</i> [2]	Pointer to the <i>IppiReconstructHighMB_32s16u</i> and <i>IppiReconstructHighMB_16s8u</i> structures. The first pointer <i>pReconstructInfo</i> [0] points to the structure with parameters for restructuring chroma macroblock of plane <i>U</i> , and the second pointer <i>pReconstructInfo</i> [1] points to the structure with parameters for restructuring Chroms macroblock of plane <i>V</i> .

## Description

The functions *ippiReconstructChroma422IntraHalf4x4\_H264High\_32s16u\_IP2R* and *ippiReconstructChroma422IntraHalf4x4\_H264High\_16s8u\_IP2R* are declared in the *ippvc.h* file. These functions reconstruct 4x4 intra chroma macroblocks (16x8 *U* macroblock and 16x8 *V* macroblock) divided into upper and lower 8x8 halves for chroma format 4:2:2 (*chroma\_format\_idc* is equal to 2). The halves differ only in prediction, that is, have independent prediction vectors. Otherwise their performance is the same:

- Perform integer inverse transformation and dequantization for 2x2 *U* DC coefficients and for 2x2 *V* DC coefficients in accordance with 8.5.7 of [ITUH264] in the same way as *TransformDequantChromaDC\_H264* function does.
- Perform scaling, integer inverse transformation, and shift for 4x4 AC *U*-blocks and shift for 4x4 AC *V*-blocks in accordance with 8.5.8 of [ITUH264] in the same way as *DequantTransformResidual\_H264* function does. This process is performed on all 4x4 chroma blocks in the same order as is shown in Figure 8-7(b) of [ITUH264].
- Perform adding of 16x8 intra prediction blocks and 16x8 residual blocks in accordance with 8-306 of [ITUH264].

These functions are used in the H.264 decoder included into Intel IPP Samples. See [introduction](#) to the H.264 section.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if at least one of the specified pointers is NULL.
<code>ippStsOutOfRangeErr</code>	<i>ChromaQPU</i> or <i>ChromaQPV</i> is less than 0 or greater than 39.

## ReconstructChroma422Intra4x4\_H264High

Reconstructs 4X4 intra chroma macroblock for 4:2:2 chroma mode.

### Syntax

```

IppStatus ippiReconstructChroma422Intra4x4_H264High_32s16u_IP2R(const
IppiReconstructHighMB_32s16u* pReconstructInfo[2], IppIntraChromaPredMode_H264
intraChromaMode, Ipp32u edgeType, Ipp32u levelScaleDCU, Ipp32u levelScaleDCV);

IppStatus ippiReconstructChroma422Intra4x4_H264High_16s8u_IP2R(const
IppiReconstructHighMB_16s8u* pReconstructInfo[2], IppIntraChromaPredMode_H264
intraChromaMode, Ipp32u edgeType, Ipp32u levelScaleDCU, Ipp32u levelScaleDCV);

```

### Parameters

<i>ppSrcDstCoeff</i>	Pointer to the order of 4x4 blocks of residual coefficients for this macroblock, which are taken as a result of Huffman decoding (2x2 DC <i>U</i> -block, 2x2 DC <i>V</i> -block, 4x4 AC <i>U</i> -blocks, 4x4 AC <i>V</i> -blocks if the block is not zero-filled) in the same order as is shown in Figure 8-7(b) of [ITUH264]. Pointer is updated by the function and points to the blocks for the next macroblock.
<i>pSrcDstUPlane</i>	Pointer to macroblock that is reconstructed in current <i>U</i> plane. This macroblock must contain intra prediction samples.
<i>pSrcDstVPlane</i>	Pointer to macroblock that is reconstructed in current <i>V</i> plane. This macroblock must contain intra prediction samples.
<i>srcDstUVStep</i>	Plane step.
<i>intraChromaMode</i>	Prediction mode of the Intra_chroma prediction process for chroma samples. This mode is defined in the same way as in <a href="#">PredictIntraChroma8x8_H264</a> function.
<i>cbp4x4</i>	Coded block pattern. If <i>cbp4x4</i> & (1<<(IPPVC_CBP_1ST_CHROMA_DC_BITPOS)) is not equal to 0, 2x2 DC <i>U</i> -block is not zero-filled and exists in <i>ppSrcDstCoeff</i> . If <i>cbp4x4</i> & (1<<(IPPVC_CBP_1ST_CHROMA_DC_BITPOS+ 1)) is not equal to 0, 2x2 DC <i>V</i> -block is not zero-filled and exists in <i>ppSrcDstCoeff</i> . If <i>cbp4x4</i> & (1<<(IPPVC_CBP_1ST_CHROMA_AC_BITPOS+ <i>i</i> )) is not equal to 0, (0 ≤ <i>i</i> < 8), <i>i</i> -th 4x4 AC <i>U</i> -block is not zero-filled and exists in <i>ppSrcDstCoeff</i> . If <i>cbp4x4</i> &

	$(1 < (IPPVC\_CBP\_1ST\_CHROMA\_AC\_BITPOS + i + 8))$ is not equal to 0, $(0 \leq i < 8)$ , $i$ -th 4x4 AC $U$ -block is not zero-filled and exists in $ppSrcDstCoeff$ .
<i>ChromaQPU</i>	Chroma quantizer for $U$ plane ( $QP_C$ in [ITUH264]). It must be within the range [0;39].
<i>ChromaQPV</i>	Chroma quantizer for $V$ plane ( $QP_C$ in [ITUH264]). It must be within the range [0;39].
<i>levelScaleDCU</i>	Chroma quantizer for $U$ plane ( $LevelScale(QP_C, DC \% 6, 0, 0)$ in [ITUH264]) for chroma DC coefficients.
<i>levelScaleDCV</i>	Chroma quantizer for $V$ plane ( $LevelScale(QP_C, DC \% 6, 0, 0)$ in [ITUH264]) for chroma DC coefficients.
<i>edgeType</i>	Flag that specifies the availability of the macroblocks used for prediction. If the upper macroblock is not available for intra prediction, $edgeType \& IPPVC\_TOP\_EDGE$ must be non-zero. If the left macroblock is not available for intra prediction, $edgeType \& IPPVC\_LEFT\_EDGE$ must be non-zero. If the upper-left macroblock is not available for intra prediction, $edgeType \& IPPVC\_TOP\_LEFT\_EDGE$ must be non-zero.
<i>pQuantTableU</i>	Pointer to the quantization table for $U$ plane ( $LevelScale(qP \% 6, i, j)$ in [ITUH264]).
<i>pQuantTableV</i>	Pointer to the quantization table for $V$ plane ( $LevelScale(qP \% 6, i, j)$ in [ITUH264]).
<i>bypassFlag</i>	Flag enabling lossless coding.
<i>pReconstructInfo[2]</i>	Pointer to the <code>IppiReconstructHighMB_32s16u</code> and <code>IppiReconstructHighMB_16s8u</code> structures. The first pointer $pReconstructInfo[0]$ points to the structure with parameters for restructuring chroma macroblock of plane $U$ , and the second pointer $pReconstructInfo[1]$ points to the structure with parameters for restructuring Chroms macroblock of plane $V$ .

## Description

The functions `ippiReconstructChroma422Intra4x4_H264High_32s16u_IP2R` and `ippiReconstructChroma422Intra4x4_H264High_16s8u_IP2R` are declared in the `ippvc.h` file. These functions reconstruct eight 4x4 intra chroma macroblocks (16x8  $U$  macroblock and 16x8  $V$  macroblock) for chroma format 4:2:2 ( $chroma\_format\_idc$  is equal to 2):

- Perform integer inverse transformation and dequantization for 2x2  $U$  DC coefficients and for 2x2  $V$  DC coefficients in accordance with 8.5.7 of [ITUH264] in the same way as `TransformDequantChromaDC_H264` function does.
- Perform scaling, integer inverse transformation, and shift for 4x4 AC  $U$ -blocks and shift for 4x4 AC  $V$ -blocks in accordance with 8.5.8 of [ITUH264] in the same way as `DequantTransformResidual_H264` function does. This process is performed on all 4x4 chroma blocks in the same order as is shown in Figure 8-7(b) of [ITUH264].
- Perform adding of 16x8 intra prediction blocks and 16x8 residual blocks in accordance with 8-306 of [ITUH264].

These functions are used in the H.264 decoder included into Intel IPP Samples. See [introduction](#) to the H.264 section.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if at least one of the specified pointers is NULL.
<code>ippStsOutOfRangeErr</code>	<i>ChromaQPU</i> or <i>ChromaQPV</i> is less than 0 or greater than 39.

## ReconstructLumaInterMB\_H264

*Reconstructs inter luma macroblock.*

### Syntax

```
IppStatus ippReconstructLumaInterMB_H264_16s8u_C1R(Ipp16s** ppSrcCoeff, Ipp8u*
pSrcDstYPlane, Ipp32u srcDstYStep, Ipp32u cbp4x4, Ipp32s QP);
```

### Parameters

<i>ppSrcCoeff</i>	Pointer to the order of 4x4 blocks of residual coefficients for this macroblock, which are taken as a result of Huffman decoding (4x4 luma blocks, if the block is not zero-filled) in the same order as is shown in Figure 6-6 of [JVTG050]. Pointer is updated by the function and points to the blocks for the next macroblock.
<i>pSrcDstYPlane</i>	Pointer to the current macroblock that is reconstructed in current Y-plane. This macroblock must contain inter prediction samples.
<i>srcDstStep</i>	Plane step.
<i>cbp4x4</i>	Coded block pattern. If $cbp4x4 \ \& \ (1 << (1+i))$ is not equal to 0 ( $0 \leq i < 16$ ), <i>i</i> -th 4x4 AC luma block is not zero-filled and it exists in <i>ppSrcCoeff</i> .
<i>QP</i>	Quantization parameter ( $QP_Y$ in [JVTG050]). It must be within the range [0;51].

### Description

The function `ippReconstructLumaInterMB_H264_16s8u_C1R` is declared in the `ippvc.h` file. This function reconstructs inter luma macroblock:

- Performs scaling, integer inverse transformation, and shift for each 4x4 block in the same order as is shown in Figure 6-6 of [JVTG050] in accordance with 8.5.8 of [JVTG050] in the same way as [DequantTransformResidual\\_H264](#) function does.
- Performs adding of a 16x16 inter prediction block and a 16x16 residual block in accordance with 8-247 of [JVTG050].

This function is used in the H.264 decoder included into Intel IPP Samples. See [introduction](#) to the H.264 section.

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
--------------------------	---------------------

<code>ippStsNullPtrErr</code>	Indicates an error condition if at least one of the specified pointers is NULL.
<code>ippStsOutOfRangeErr</code>	$QP$ is less than 0 or greater than 51.

## ReconstructLumaIntraHalfMB\_H264

*Reconstructs half of intra luma macroblock.*

### Syntax

```
ippStatus ippiReconstructLumaIntraHalfMB_H264_16s8u_C1R(Ipp16s** ppSrcCoeff, Ipp8u*
pSrcDstYPlane, Ipp32s srcDstYStep, IppIntra4x4PredMode_H264* pMBIntraTypes, Ipp32u cbp4x2,
Ipp32u QP, Ipp8u edgeType);
```

### Parameters

<code>ppSrcCoeff</code>	Pointer to the order of 4x4 blocks of residual coefficients for this macroblock, which are taken as a result of Huffman decoding (4x4 luma blocks, if the block is not zero-filled) in the same order as is shown in Figure 6-6 of [JVTG050]. Pointer is updated by the function and points to the blocks for the next macroblock.
<code>pSrcDstYPlane</code>	Pointer to the current macroblock that is reconstructed in Y-plane.
<code>srcDstYStep</code>	Y-Plane step.
<code>pMBIntraTypes</code>	Array of Intra_4x4 luma prediction modes for 8 subblocks. <code>pMBIntraTypes[ i ]</code> is defined in the same way as in <a href="#">PredictIntra_4x4_H264</a> function ( $0 \leq i < 8$ ).
<code>cbp4x2</code>	Coded block pattern. If <code>cbp4x2 &amp; (1 &lt;&lt; (1+i))</code> is not equal to 0 ( $0 \leq i < 8$ ), $i$ -th 4x4 AC luma block is not zero-filled and it exists in <code>ppSrcCoeff</code> .
<code>QP</code>	Quantization parameter ( $QP_Y$ in [JVTG050]). It must be within the range [0;51].
<code>edgeType</code>	Flag that specifies the availability of the macroblocks used for prediction. If the upper macroblock is not available for 4x4 intra prediction, <code>edgeType&amp;IPPVC_TOP_EDGE</code> must be non-zero. If the left macroblock is not available for 4x4 intra prediction, <code>edgeType&amp;IPPVC_LEFT_EDGE</code> must be non-zero. If the upper-left macroblock is not available for 4x4 intra prediction, <code>edgeType&amp;IPPVC_TOP_LEFT_EDGE</code> must be non-zero. If the upper-right macroblock is not available for 4x4 intra prediction, <code>edgeType&amp;IPPVC_TOP_RIGHT_EDGE</code> must be non-zero.

### Description

The function `ippiReconstructLumaIntraHalfMB_H264_16s8u_C1R` is declared in the `ippvc.h` file. This function reconstructs a half of intra luma macroblock, that is, eight 4x4 subblocks. The process for the eight 4x4 blocks (from 0 to 7 or from 8 to 15) in the same order as is shown in Figure 6-6 of [JVTG050] is described below:



- Performs scaling, integer inverse transformation, and shift for a 4x4 block in accordance with 8.5.8 of [JVTG050] in the same way as [DequantTransformResidual\\_H264](#) function does.
- Performs intra prediction for a 4x4 luma component in accordance with 8.3.1.2 of [JVTG050] in the same way as [PredictIntra\\_4x4\\_H264](#) function does.
- Performs adding of 16x8 prediction block and 16x8 residual block in accordance with 8-247 of [JVTG050].

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if at least one of the specified pointers is NULL.
<code>ippStsOutOfRangeErr</code>	$QP$ is less than 0 or greater than 51.

## ReconstructLumaIntraMB\_H264

*Reconstructs intra luma macroblock.*

### Syntax

```
IppStatus ippiReconstructLumaIntraMB_H264_16s8u_C1R(Ipp16s** ppSrcCoeff, Ipp8u*
pSrcDstYPlane, Ipp32s srcDstYStep, const IppIntra4x4PredMode_H264* pMBIntraTypes, const
Ipp32u cbp4x4, const Ipp32u QP, const Ipp8u edgeType);
```

### Parameters

<code>ppSrcCoeff</code>	Pointer to the order of 4x4 blocks of residual coefficients for this macroblock, which are taken as a result of Huffman decoding (4x4 luma blocks, if the block is not zero-filled) in the same order as is shown in Figure 6-6 of [JVTG050]. Pointer is updated by the function and points to the blocks for the next macroblock.
<code>pSrcDstYPlane</code>	Pointer to the current macroblock that is reconstructed in $Y$ -plane.
<code>srcDstYStep</code>	$Y$ -Plane step.
<code>pMBIntraTypes</code>	Array of Intra_4x4 luma prediction modes for each subblock. <code>pMBIntraTypes[ <math>i</math> ]</code> is defined in the same way as in <a href="#">PredictIntra_4x4_H264</a> function ( $0 \leq i < 16$ ).
<code>cbp4x4</code>	Coded block pattern. If <code>cbp4x4 &amp; (1 &lt;&lt; (1+i))</code> is not equal to 0 ( $0 \leq i < 16$ ), $i$ -th 4x4 AC luma block is not zero-filled and it exists in <code>ppSrcCoeff</code> .
<code>QP</code>	Quantization parameter ( $QP_Y$ in [JVTG050] ). It must be within the range [0;51].
<code>edgeType</code>	Flag that specifies the availability of the macroblocks used for prediction. If the upper macroblock is not available for 4x4 intra prediction, <code>edgeType&amp;IPPVC_TOP_EDGE</code> must be non-zero. If the left macroblock is not available for 4x4 intra prediction, <code>edgeType&amp;IPPVC_LEFT_EDGE</code> must be non-zero. If the upper-left macroblock is not available for 4x4 intra prediction, <code>edgeType&amp;IPPVC_TOP_LEFT_EDGE</code> must be non-zero.

If the upper-right macroblock is not available for 4x4 intra prediction, `edgeType&IPPVC_TOP_RIGHT_EDGE` must be non-zero.

## Description

The function `ippiReconstructLumaIntraMB_H264_16s8u_C1R` is declared in the `ippvc.h` file. This function reconstructs intra luma macroblock. The process for each 4x4 block in the same order as is shown in Figure 6-6 of [JVTG050] is described below:

- Performs scaling, integer inverse transformation, and shift for a 4x4 block in accordance with 8.5.8 of [JVTG050] in the same way as `DequantTransformResidual_H264` function does.
- Performs intra prediction for a 4x4 luma component in accordance with 8.3.1.2 of [JVTG050] in the same way as `PredictIntra_4x4_H264` function does.
- Performs adding of 4x4 prediction block and 4x4 residual block in accordance with 8-247 of [JVTG050].

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if at least one of the specified pointers is NULL.
<code>ippStsOutOfRangeErr</code>	<code>QP</code> is less than 0 or greater than 51.

## ReconstructLumaInter4x4MB\_H264, ReconstructLumaInter4x4\_H264High

Reconstruct 4X4 inter luma macroblock for high profile.

## Syntax

```
IppStatus ippiReconstructLumaInter4x4MB_H264_16s8u_C1R(Ipp16s** ppSrcDstCoeff, Ipp8u*
pSrcDstYPlane, Ipp32u srcDstYStep, Ipp32u cbp4x4, Ipp32s QP, const Ipp16s* pQuantTable,
Ipp8u bypassFlag);
```

```
IppStatus ippiReconstructLumaInter4x4_H264High_32s16u_IP1R(const
IppiReconstructHighMB_32s16u* pReconstructInfo);
```

## Parameters

<code>ppSrcDstCoeff</code>	Pointer to the order of 4x4 blocks of residual coefficients for this macroblock, which are taken as a result of Huffman decoding (4x4 luma blocks, if the block is not zero-filled) in the same order as is shown in Figure 6-10 of [ITUH264]. Pointer is updated by the function and points to the blocks for the next macroblock.
<code>pSrcDstYPlane</code>	Pointer to the current macroblock that is reconstructed in current Y-plane. This macroblock must contain inter prediction samples.
<code>srcDstStep</code>	Plane step.
<code>cbp4x4</code>	Coded block pattern. If <code>cbp4x4 &amp; (1&lt;&lt;(1+i))</code> is not equal to 0 ( $0 \leq i < 16$ ), <i>i</i> -th 4x4 AC luma block is not zero-filled and it exists in <code>ppSrcDstCoeff</code> .

<i>QP</i>	Quantization parameter ( <i>QP<sub>Y</sub></i> in [ITUH264]). It must be within the range [0;51].
<i>pQuantTable</i>	Pointer to the quantization table ( <i>LevelScale(qP%6, i, j)</i> in [ITUH264]).
<i>bypassFlag</i>	Flag enabling lossless coding.
<i>pReconstructInfo</i>	Pointer to the <code>IppiReconstructHighMB_32s16u</code> structure.

## Description

The functions `ippiReconstructLumaInter4x4MB_H264_16s8u_C1R` and `ippiReconstructLumaInter4x4_H264High_32s16u_IP1R` are declared in the `ippvc.h` file. These functions reconstruct a 4x4 inter luma macroblock for high profile:

- Perform scaling, integer inverse transformation, and shift for each 4x4 block in the same order as is shown in Figure 6-10 of [ITUH264] in accordance with 8.5.10 of [ITUH264].
- Perform adding of sixteen 4x4 inter prediction blocks and 4x4 residual blocks in accordance with 8-295 of [ITUH264].

These functions are used in the H.264 decoder included into Intel IPP Samples. See [introduction](#) to the H.264 section.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if at least one of the specified pointers is NULL.
<code>ippStsOutOfRangeErr</code>	<i>QP</i> is less than 0 or greater than 51.

## Example

The code example below shows how to use `ippiReconstructLumaInter4x4MB_H264_16s8u_C1R` function.

```
{
    Ipp8u  planeY[16*16] =
    { 0x9c, 0x92, 0x7f, 0x8b, 0x94, 0x22, 0x8a, 0xbc, 0xbb, 0x23, 0x60, 0xbb, 0x9a, 0xa9, 0x54, 0x48,
      0x13, 0xa3, 0x29, 0x8a, 0xba, 0x73, 0x65, 0x14, 0x9a, 0xc6, 0x34, 0xae, 0x2e, 0x5e, 0xbb, 0x38,
      0x70, 0x48, 0x5e, 0x68, 0xaf, 0x16, 0x22, 0x9a, 0x38, 0x6c, 0x54, 0x8f, 0x3e, 0x7f, 0x6a, 0x69,
      0x3e, 0x5e, 0x11, 0x7e, 0x08, 0x01, 0xb9, 0x8c, 0x76, 0x7b, 0x16, 0x87, 0xa5, 0xa4, 0xc6, 0x78,
      0xc3, 0x0a, 0x8a, 0x9d, 0x47, 0x0f, 0x9f, 0xa1, 0x90, 0x5f, 0x1d, 0x0b, 0x79, 0x36, 0x47, 0x79,
      0x5a, 0x45, 0x28, 0x87, 0x28, 0x0f, 0x4a, 0xb3, 0x24, 0x83, 0x14, 0xa4, 0x56, 0x5c, 0x81, 0x0b,
      0x29, 0x61, 0xc1, 0x4b, 0x96, 0x33, 0x65, 0xa3, 0x9f, 0x3b, 0x08, 0x25, 0xa9, 0x5c, 0x44, 0x97,
      0x50, 0x2b, 0x77, 0x64, 0x59, 0x2c, 0xb6, 0x30, 0x24, 0x7b, 0x47, 0x6b, 0xaa, 0x85, 0xa3, 0xa2,
      0x58, 0x66, 0x15, 0x31, 0x78, 0x7e, 0x3b, 0x20, 0x72, 0x25, 0xb3, 0x9a, 0x82, 0x13, 0x48, 0x53,
      0x61, 0x4f, 0x45, 0x0c, 0x69, 0xa2, 0xa1, 0x01, 0x62, 0x0f, 0x0a, 0x94, 0x17, 0x4b, 0x77, 0x41,
      0xa4, 0x59, 0x16, 0x17, 0x8e, 0x32, 0x61, 0x94, 0xbe, 0x1d, 0x22, 0x05, 0x11, 0x16, 0xb0, 0x4f,
      0x31, 0x97, 0x3e, 0x61, 0x92, 0x36, 0x13, 0x9c, 0x07, 0x27, 0x1a, 0x05, 0x5d, 0x82, 0x9b, 0x4e,
      0x94, 0x6f, 0x57, 0xc7, 0xb7, 0x99, 0x1e, 0x48, 0x33, 0x3f, 0x15, 0x9a, 0x30, 0x75, 0x03, 0x69,
      0x09, 0x86, 0x86, 0xb6, 0x53, 0x9d, 0x9b, 0x03, 0x9c, 0x80, 0xbf, 0x3a, 0x81, 0x0b, 0x59, 0xb5,
      0x6f, 0x81, 0x65, 0x29, 0xb7, 0x93, 0xb4, 0x2a, 0xbf, 0x05, 0x11, 0x53, 0x49, 0x92, 0x97, 0x3c,
      0x0d, 0xac, 0x8f, 0x5d, 0x88, 0x25, 0x99, 0x9b, 0x5e, 0x53, 0x5c, 0x07, 0x0f, 0x0c, 0x41, 0xc2
    };

    Ipp32s  stepY = 16;
    Ipp16s  quantTable[16] = {22,12,34,50,34,33,46,21,18,19,34,28,28,31,15,3};
    Ipp16s  coef[16*16] = {
```

```

12,-4, 0, 0, 0,-7, 2, 0, 0, 1, 0, 0, 0, 0, 0, 0,
0, 4, -5, 1, 0,-4, 3, 0, 0, 2, 0, 0, 0, 0, 0, 0,
0, 0, 10, -2,-1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 7, -13, 2, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1,
0, 0, -19, 6, 0, 0, 8,-4, 0, 0, 0, 1, 0, 0, 0, 0,
0, 0, -1, -1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 25, -3,-2, 0, 0, 2,-1, 0,-2, 1, 0, 0, 0, 0,
0, 0, -3, 0, 1,-2,-1, 0, 0, 0,-1, 0, 0, 0, 0, 0,
0, 0, -1, -1, 1, 1, 2, 1,-2,-1, 0, 1, 0, 1, 0, 0,
0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0,-1,-1,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,-1, 1, 0,
0, 0, 0, 0,-1, 0, 0, 0, 0, 0, 0, 0, 0,-1, 1, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 }

Ipp32u cbp4x4 = 0x0001fe00;
Ipp32s QP = 22;
Ipp8u bypass_flag = 0;
IppStatus result;
Ipp16s* pCoef;

pCoef = coef;

result = ippiReconstructLumaInter4x4MB_H264_16s8u_C1R(&pCoef,
                                                    planeY,
                                                    stepY,
                                                    cbp4x4,
                                                    QP,
                                                    quantTable,
                                                    bypass_flag);
}

```

## ReconstructLumaIntraHalf4x4MB\_H264, ReconstructLumaIntraHalf4x4\_H264High

*Reconstruct half of 4X4 intra luma macroblock for high profile.*

### Syntax

```

IppStatus ippiReconstructLumaIntraHalf4x4MB_H264_16s8u_C1R(Ipp16s** ppSrcDstCoeff, Ipp8u*
pSrcDstYPlane, Ipp32s srcDstYStep, IppIntra4x4PredMode_H264* pMBIntraTypes, Ipp32u cbp4x2,
Ipp32s QP, Ipp8u edgeType, const Ipp16s* pQuantTable, Ipp8u bypassFlag);

IppStatus ippiReconstructLumaIntraHalf4x4_H264High_32s16u_IP1R(const
IppiReconstructHighMB_32s16u* pReconstructInfo, const IppIntra4x4PredMode_H264*
pMBIntraTypes, Ipp32u edgeType);

```

### Parameters

<i>ppSrcDstCoeff</i>	Pointer to the order of 4x4 blocks of residual coefficients for this macroblock, which are taken as a result of Huffman decoding (4x4 luma blocks, if the block is not zero-filled) in the same order as is shown in Figure 6-10 of [ITUH264]. Pointer is updated by the function and points to the blocks for the next macroblock.
<i>pSrcDstYPlane</i>	Pointer to the current macroblock that is reconstructed in Y-plane.

<i>srcDstYStep</i>	Y-plane step.
<i>pMBIntraTypes</i>	Array of Intra_4x4 luma prediction modes for eight subblocks. <i>pMBIntraTypes[ i ]</i> is defined in the same way as in <a href="#">PredictIntra_4x4_H264</a> function ( $0 \leq i < 8$ ).
<i>cbp4x2</i>	Coded block pattern. If <i>cbp4x2</i> & (1<<(1+i)) is not equal to 0 ( $0 \leq i < 8$ ), <i>i</i> -th 4x4 AC luma block is not zero-filled and it exists in <i>ppSrcDstCoeff</i> .
<i>QP</i>	Quantization parameter ( <i>QP<sub>Y</sub></i> in <a href="#">ITUH264</a> ). It must be within the range [0;51].
<i>edgeType</i>	Flag that specifies the availability of the macroblocks used for prediction. If the upper macroblock is not available for 4x4 intra prediction, <i>edgeType</i> &IPPVC_TOP_EDGE must be non-zero. If the left macroblock is not available for 4x4 intra prediction, <i>edgeType</i> &IPPVC_LEFT_EDGE must be non-zero. If the upper-left macroblock is not available for 4x4 intra prediction, <i>edgeType</i> &IPPVC_TOP_LEFT_EDGE must be non-zero. If the upper-right macroblock is not available for 4x4 intra prediction, <i>edgeType</i> &IPPVC_TOP_RIGHT_EDGE must be non-zero.
<i>pQuantTable</i>	Pointer to the quantization table ( <i>LevelScale(qP%6, i, j)</i> in <a href="#">ITUH264</a> ).
<i>bypassFlag</i>	Flag enabling lossless coding.
<i>pReconstructInfo</i>	Pointer to the <i>IppiReconstructHighMB_32s16u</i> structure.

## Description

The functions *ippiReconstructLumaIntraHalf4x4MB\_H264\_16s8u\_C1R* and *ippiReconstructLumaIntraHalf4x4\_H264High\_32s16u\_IP1R* are declared in the *ippvc.h* file. These functions reconstruct half of an intra luma macroblock for high profile. The process for eight 4x4 blocks (from 0 to 7 or from 8 to 15) in the same order as is shown in Figure 6-10 of [ITUH264](#) is described below:

- Performs scaling, integer inverse transformation, and shift for 4x4 blocks in accordance with 8.5.10 of [ITUH264](#).
- Performs intra prediction for 4x4 luma components in accordance with 8.3.1.2 of [ITUH264](#).
- Performs adding of a 16x8 prediction block (eight 4x4 blocks) and a 16x8 residual block in accordance with 8-295 of [ITUH264](#).

## Return Values

<i>ippStsNoErr</i>	Indicates no error.
<i>ippStsNullPtrErr</i>	Indicates an error condition if at least one of the specified pointers is NULL.
<i>ippStsOutOfRangeErr</i>	<i>QP</i> is less than 0 or greater than 51.

## ReconstructLumaIntra4x4MB\_H264, ReconstructLumaIntra4x4\_H264High

*Reconstruct 4X4 intra luma macroblock for high profile.*

### Syntax

```
IppStatus ippiReconstructLumaIntra4x4MB_H264_16s8u_C1R(Ipp16s** ppSrcDstCoeff, Ipp8u*
pSrcDstYPlane, Ipp32s srcDstYStep, IppIntra4x4PredMode_H264* pMBIntraTypes, Ipp32u cbp4x4,
Ipp32s QP, Ipp8u edgeType, const Ipp16s* pQuantTable, Ipp8u bypassFlag);
```

```
IppStatus ippiReconstructLumaIntra4x4_H264High_32s16u_IP1R(const
IppiReconstructHighMB_32s16u* pReconstructInfo, const IppIntra4x4PredMode_H264*
pMBIntraTypes, Ipp32s edgeType);
```

### Parameters

<i>ppSrcDstCoeff</i>	Pointer to the order of 4x4 blocks of residual coefficients for this macroblock, which are taken as a result of Huffman decoding (4x4 luma blocks, if the block is not zero-filled) in the same order as is shown in Figure 6-10 of [ITUH264]. Pointer is updated by the function and points to the blocks for the next macroblock.
<i>pSrcDstYPlane</i>	Pointer to the current macroblock that is reconstructed in current Y-plane.
<i>srcDstYStep</i>	Y-plane step.
<i>pMBIntraTypes</i>	Array of Intra_4x4 luma prediction modes for each subblock. <i>pMBIntraTypes[ i ]</i> is defined in the same way as in <a href="#">PredictIntra_4x4_H264</a> function ( $0 \leq i < 16$ ).
<i>cbp4x4</i>	Coded block pattern. If <i>cbp4x4</i> & $(1 < (1+i))$ is not equal to 0 ( $0 \leq i < 16$ ), <i>i</i> -th 4x4 AC luma block is not zero-filled and it exists in <i>ppSrcDstCoeff</i> .
<i>QP</i>	Quantization parameter ( <i>QP<sub>Y</sub></i> in [ITUH264]). It must be within the range [0;51].
<i>edgeType</i>	Flag that specifies the availability of the macroblocks used for prediction. If the upper macroblock is not available for 4x4 intra prediction, <i>edgeType</i> &IPPVC_TOP_EDGE must be non-zero. If the left macroblock is not available for 4x4 intra prediction, <i>edgeType</i> &IPPVC_LEFT_EDGE must be non-zero. If the upper-left macroblock is not available for 4x4 intra prediction, <i>edgeType</i> &IPPVC_TOP_LEFT_EDGE must be non-zero. If the upper-right macroblock is not available for 4x4 intra prediction, <i>edgeType</i> &IPPVC_TOP_RIGHT_EDGE must be non-zero.
<i>pQuantTable</i>	Pointer to the quantization table ( <i>LevelScale</i> ( <i>QP%6</i> , <i>i</i> , <i>j</i> ) in [ITUH264]).
<i>bypassFlag</i>	Flag enabling lossless coding.
<i>pReconstructInfo</i>	Pointer to the <i>IppiReconstructHighMB_32s16u</i> structure.

## Description

The functions `ippiReconstructLumaIntra4x4MB_H264_16s8u_C1R` and `ippiReconstructLumaIntra4x4_H264High_32s16u_IP1R` are declared in the `ippvc.h` file. These functions reconstruct inter luma macroblock for high profile. The process for each 4x4 block in the same order as is shown in Figure 6-10 of [ITUH264] is described below:

- Perform scaling, integer inverse transformation, and shift for each 4x4 block in accordance with 8.5.10 of [ITUH264].
- Perform intra prediction for a 4x4 luma component in accordance with 8.3.1.2 of [ITUH264].
- Perform adding of sixteen 4x4 prediction blocks and sixteen 4x4 residual blocks in accordance with 8-295 of [ITUH264].

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if at least one of the specified pointers is NULL.
<code>ippStsOutOfRangeErr</code>	$QP$ is less than 0 or greater than 51.

## ReconstructLumaInter8x8MB\_H264, ReconstructLumaInter8x8\_H264High

*Reconstructs 8X8 inter luma macroblock for high profile.*

## Syntax

```
ippStatus ippiReconstructLumaInter8x8MB_H264_16s8u_C1R(Ipp16s** ppSrcDstCoeff, Ipp8u* pSrcDstYPlane, Ipp32u srcDstYStep, Ipp32u cbp8x8, Ipp32s QP, const Ipp16s* pQuantTable, Ipp8u bypassFlag);
```

```
ippStatus ippiReconstructLumaInter8x8_H264High_32s16u_IP1R(const IppiReconstructHighMB_32s16u* pReconstructInfo);
```

## Parameters

<code>ppSrcDstCoeff</code>	Pointer to the order of 8x8 blocks of residual coefficients for this macroblock, which are taken as a result of Huffman decoding (8x8 luma blocks, if the block is not zero-filled) in the same order as is shown in Figure 6-11 of [ITUH264]. Pointer is updated by the function and points to the blocks for the next macroblock.
<code>pSrcDstYPlane</code>	Pointer to the current macroblock that is reconstructed in current Y-plane. This macroblock must contain inter prediction samples.
<code>srcDstStep</code>	Plane step.
<code>cbp8x8</code>	Coded block pattern. If $cbp8x8 \ \& \ (1 << (1+i))$ is not equal to 0 ( $0 \leq i < 4$ ), $i$ -th 8x8 AC luma block is not zero-filled and it exists in <code>ppSrcDstCoeff</code> .
<code>QP</code>	Quantization parameter ( $QP_Y$ in [ITUH264]). It must be within the range [0;51].

<i>pQuantTable</i>	Pointer to the quantization table ( <i>LevelScale(qP%6, i, j)</i> in [ITUH264]).
<i>bypassFlag</i>	Flag enabling lossless coding.
<i>pReconstructInfo</i>	Pointer to the <code>IppiReconstructHighMB_32s16u</code> structure.

### Description

The function `ippiReconstructLumaInter8x8MB_H264_16s8u_C1R` is declared in the `ippvc.h` file. This function reconstructs 8x8 inter luma macroblock for high profile:

- Performs scaling, integer inverse transformation, and shift for each 8x8 block in the same order as is shown in Figure 6-11 of [ITUH264] in accordance with 8.5.11 of [ITUH264].
- Performs adding of a 16x16 inter prediction block and a 16x16 residual block in accordance with 8-298 of [ITUH264].

This function is used in the H.264 decoder included into Intel IPP Samples. See [introduction](#) to the H.264 section.

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if at least one of the specified pointers is NULL.
<code>ippStsOutOfRangeErr</code>	<i>QP</i> is less than 0 or greater than 51.

## ReconstructLumaIntraHalf8x8MB\_H264, ReconstructLumaIntraHalf8x8\_H264High

*Reconstruct half of 8X8 intra luma macroblock for high profile.*

---

### Syntax

```
IppStatus ippiReconstructLumaIntraHalf8x8MB_H264_16s8u_C1R(Ipp16s** ppSrcDstCoeff, Ipp8u* pSrcDstYPlane, Ipp32s srcDstYStep, IppIntra8x8PredMode_H264* pMBOIntraTypes, Ipp32u cbp8x2, Ipp32s QP, Ipp8u edgeType, const Ipp16s* pQuantTable, Ipp8u bypassFlag);
```

```
IppStatus ippiReconstructLumaIntraHalf8x8_H264High_32s16u_IP1R(const IppiReconstructHighMB_32s16u* pReconstructInfo, IppIntra8x8PredMode_H264* pMBIntraTypes, Ipp32u edgeType);
```

### Parameters

<i>ppSrcDstCoeff</i>	Pointer to the order of 8x8 blocks of residual coefficients for this macroblock, which are taken as a result of Huffman decoding (8x8 luma blocks, if the block is not zero-filled) in the same order as is shown in Figure 6-11 of [ITUH264]. Pointer is updated by the function and points to the blocks for the next macroblock.
<i>pSrcDstYPlane</i>	Pointer to the current macroblock that is reconstructed in <i>Y</i> -plane.
<i>srcDstStep</i>	<i>Y</i> -plane step.



<i>pMBIntraTypes</i>	Array of Intra_8x8 luma prediction modes for each subblock. <i>pMBIntraTypes[i]</i> is defined for subblocks ( $0 \leq i < 2$ ) as described in 8.3.2.1 of [ITUH264].
<i>cbp8x2</i>	Coded block pattern. If <i>cbp8x2</i> & (1<<(1+i)) is not equal to 0 ( $0 \leq i < 2$ ), <i>i</i> -th 8x8 AC luma block is not zero-filled and it exists in <i>ppSrcDstCoeff</i> .
<i>QP</i>	Quantization parameter ( $QP_Y$ in [ITUH264]). It must be within the range [0;51].
<i>edgeType</i>	Flag that specifies the availability of the macroblocks used for prediction. If the upper macroblock is not available for 8x8 intra prediction, <i>edgeType</i> &IPPVC_TOP_EDGE must be non-zero. If the left macroblock is not available for 8x8 intra prediction, <i>edgeType</i> &IPPVC_LEFT_EDGE must be non-zero. If the upper-left macroblock is not available for 8x8 intra prediction, <i>edgeType</i> &IPPVC_TOP_LEFT_EDGE must be non-zero. If the upper-right macroblock is not available for 8x8 intra prediction, <i>edgeType</i> &IPPVC_TOP_RIGHT_EDGE must be non-zero.
<i>pQuantTable</i>	Pointer to the quantization table ( <i>LevelScale</i> ( <i>qP</i> %6, <i>i</i> , <i>j</i> ) in [ITUH264]).
<i>bypassFlag</i>	Flag enabling lossless coding.
<i>pReconstructInfo</i>	Pointer to the <i>IppiReconstructHighMB_32s16u</i> structure.

## Description

The functions *ippiReconstructLumaIntraHalf8x8MB\_H264\_16s8u\_C1R* and *ippiReconstructLumaIntraHalf8x8\_H264High\_32s16u\_IP1R* are declared in the *ippvc.h* file. These functions reconstruct 8x8 intra luma macroblock for high profile. The process for each 8x8 block in the same order as shown in Figure 6-11 of [ITUH264] is described below:

- Performs scaling, integer inverse transformation, and shift for a 8x8 block in accordance with 8.5.11 of [ITUH264].
- Performs intra prediction for a 8x8 luma component in accordance with 8.3.2 of [ITUH264].
- Performs adding of a 16x16 inter prediction block and a 16x16 residual block in accordance with 8-298 of [ITUH264].

## Return Values

<i>ippStsNoErr</i>	Indicates no error.
<i>ippStsNullPtrErr</i>	Indicates an error condition if at least one of the specified pointers is NULL.
<i>ippStsOutOfRangeErr</i>	<i>QP</i> is less than 0 or greater than 51.

## ReconstructLumaIntra8x8MB\_H264, ReconstructLumaIntra8x8\_H264High

Reconstruct 8X8 intra luma macroblock for high profile.

### Syntax

```
IppStatus ippiReconstructLumaIntra8x8MB_H264_16s8u_C1R(Ipp16s** ppSrcDstCoeff, Ipp8u*
pSrcDstYPlane, Ipp32s srcDstYStep, IppIntra8x8PredMode_H264* pMB0IntraTypes, Ipp32u cbp8x8,
Ipp32s QP, Ipp8u edgeType, const Ipp16s* pQuantTable, Ipp8u bypassFlag);
```

```
IppStatus ippiReconstructLumaIntra8x8_H264High_32s16u_IP1R(const
IppiReconstructHighMB_32s16u* pReconstructInfo, IppIntra8x8PredMode_H264* pMBIntraTypes,
Ipp32u edgeType);
```

### Parameters

<i>ppSrcDstCoeff</i>	Pointer to the order of 8x8 blocks of residual coefficients for this macroblock, which are taken as a result of Huffman decoding (8x8 luma blocks, if the block is not zero-filled) in the same order as is shown in Figure 6-11 of [ITUH264]. Pointer is updated by the function and points to the blocks for the next macroblock.
<i>pSrcDstYPlane</i>	Pointer to the current macroblock that is reconstructed in current Y-plane.
<i>srcDstStep</i>	Y-plane step.
<i>pMBIntraTypes</i>	Array of Intra_8x8 luma prediction modes for each subblock. <i>pMBIntraTypes[i]</i> is defined for subblocks ( $0 \leq i < 4$ ) as described in 8.3.2.1 of [ITUH264].
<i>cbp8x8</i>	Coded block pattern. If <i>cbp8x8</i> & ( $1 < (1+i)$ ) is not equal to 0 ( $0 \leq i < 4$ ), <i>i</i> -th 8x8 AC luma block is not zero-filled and it exists in <i>ppSrcDstCoeff</i> .
<i>QP</i>	Quantization parameter ( $QP_Y$ in [ITUH264]). It must be within the range [0;51].
<i>edgeType</i>	Flag that specifies the availability of the macroblocks used for prediction. If the upper macroblock is not available for 8x8 intra prediction, <i>edgeType</i> &IPPVC_TOP_EDGE must be non-zero. If the left macroblock is not available for 8x8 intra prediction, <i>edgeType</i> &IPPVC_LEFT_EDGE must be non-zero. If the upper-left macroblock is not available for 8x8 intra prediction, <i>edgeType</i> &IPPVC_TOP_LEFT_EDGE must be non-zero. If the upper-right macroblock is not available for 8x8 intra prediction, <i>edgeType</i> &IPPVC_TOP_RIGHT_EDGE must be non-zero.
<i>pQuantTable</i>	Pointer to the quantization table ( <i>LevelScale</i> ( <i>qP</i> %6, <i>i</i> , <i>j</i> ) in [ITUH264]).
<i>bypassFlag</i>	Flag enabling lossless coding.
<i>pReconstructInfo</i>	Pointer to the <i>IppiReconstructHighMB_32s16u</i> structure.

## Description

The functions `ippiReconstructLumaIntra8x8MB_H264_16s8u_C1R` and `ippiReconstructLumaIntra8x8_H264High_32s16u_IP1R` are declared in the `ippvc.h` file. These functions reconstruct 8x8 intra luma macroblock for high profile. The process for each 8x8 block in the same order as shown in Figure 6-11 of [ITUH264] is described below:

- Performs scaling, integer inverse transformation, and shift for a 8x8 block in accordance with 8.5.11 of [ITUH264].
- Performs intra prediction for a 8x8 luma component in accordance with 8.3.2 of [ITUH264].
- Performs adding of a 16x16 inter prediction block and a 16x16 residual block in accordance with 8-298 of [ITUH264].

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if at least one of the specified pointers is NULL.
<code>ippStsOutOfRangeErr</code>	<i>QP</i> is less than 0 or greater than 51.

## ReconstructLumaIntra16x16MB\_H264

*Reconstructs 16X16 intra luma macroblock.*

## Syntax

```
IppStatus ippiReconstructLumaIntra16x16MB_H264_16s8u_C1R(Ipp16s** ppSrcCoeff, Ipp8u*
pSrcDstYPlane, Ipp32u srcDstYStep, const IppIntra16x16PredMode_H264 intraLumaMode, const
Ipp32u cbp4x4, const Ipp32u QP, const Ipp8u edgeType);
```

## Parameters

<i>ppSrcCoeff</i>	Pointer to the order of 4x4 blocks of residual coefficients for this macroblock, which are taken as a result of Huffman decoding (4x4 DC luma blocks, 4x4 AC luma blocks, if the block is not zero-filled) in the same order as is shown in Figure 6-6 of [JVTG050]. Pointer is updated by the function and points to the blocks for the next macroblock.
<i>pSrcDstYPlane</i>	Pointer to the current macroblock that is reconstructed in <i>Y</i> -plane.
<i>srcDstYStep</i>	<i>Y</i> -Plane step.
<i>intraLumaMode</i>	Prediction mode of the Intra_16x16 prediction process for luma samples. It is defined in the same way as in <a href="#">PredictIntra_16x16_H264</a> function.
<i>cbp4x4</i>	Coded block pattern. If <i>cbp4x4</i> & IPPVC_CBP_LUMA_DC is not equal to 0, 4x4 DC luma block is not zero-filled and it exists in <i>ppSrcCoeff</i> . If <i>cbp4x4</i> & (1 << (IPPVC_CBP_1ST_LUMA_AC_BITPOS + <i>i</i> )) is not equal to 0 (0 ≤ <i>i</i> < 16), <i>i</i> -th 4x4 AC luma block is not zero-filled and it exists in <i>ppSrcCoeff</i> .

<i>QP</i>	Quantization parameter ( <i>QP<sub>Y</sub></i> in [JVTG050]). It must be within the range [0;51].
<i>edgeType</i>	Flag that specifies the availability of the macroblocks used for prediction. If the upper macroblock is not available for 16x16 intra prediction, <i>edgeType</i> &IPPVC_TOP_EDGE must be non-zero. If the left macroblock is not available for 16x16 intra prediction, <i>edgeType</i> &IPPVC_LEFT_EDGE must be non-zero. If the upper-left macroblock is not available for 16x16 intra prediction, <i>edgeType</i> &IPPVC_TOP_LEFT_EDGE must be non-zero.

## Description

The function `ippiReconstructLumaIntra16x16MB_H264_16s8u_C1R` is declared in the `ippvc.h` file. This function reconstructs 16x16 intra luma macroblock:

- Performs integer inverse transformation and dequantization for 4x4 luma DC coefficients in accordance with 8.5.6 of [JVTG050] in the same way as `TransformDequantLumaDC_H264` function does.
- Performs scaling, integer inverse transformation, and shift for 4x4 AC blocks in accordance with 8.5.8 of [JVTG050] in the same way as `DequantTransformResidual_H264` function does. This process is performed on all 4x4 AC blocks in the same order as Figure 6-6 of [JVTG050] shows.
- Performs intra prediction for a 16x16 luma component in accordance with 8.3.2 of [JVTG050] in the same way as `PredictIntra_16x16_H264` function does.
- Performs adding of 16x16 prediction blocks and 16x16 residual blocks in accordance with 8-247 of [JVTG050].

This function is used in the H.264 decoder included into Intel IPP Samples. See [introduction](#) to the H.264 section.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if at least one of the specified pointers is NULL.
<code>ippStsOutOfRangeErr</code>	<i>QP</i> is less than 0 or greater than 51.

## ReconstructLumaIntra\_16x16MB\_H264, ReconstructLumaIntra16x16\_H264High

Reconstruct 16X16 intra luma macroblock for high profile.

## Syntax

```

IppStatus ippiReconstructLumaIntra_16x16MB_H264_16s8u_C1R(Ipp16s** ppSrcDstCoeff, Ipp8u*
pSrcDstYPlane, Ipp32u srcDstYStep, IppIntra16x16PredMode_H264 intraLumaMode, Ipp32u cbp4x4,
Ipp32u QP, Ipp8u edgeType, const Ipp16s* pQuantTable, Ipp8u bypassFlag);

IppStatus ippiReconstructLumaIntra16x16_H264High_32s16u_IP1R(const
IppiReconstructHighMB_32s16u* pReconstructInfo, IppIntra16x16PredMode_H264 intraLumaMode,
Ipp32u edgeType);

```

## Parameters

<i>ppSrcDstCoeff</i>	Pointer to the order of 4x4 blocks of residual coefficients for this macroblock, which are taken as a result of Huffman decoding (4x4 luma blocks, if the block is not zero-filled) in the same order as is shown in Figure 6-10 of [JVTG050]. Pointer is updated by the function and points to the blocks for the next macroblock.
<i>pSrcDstYPlane</i>	Pointer to the current macroblock that is reconstructed in Y-plane.
<i>srcDstYStep</i>	Y-Plane step.
<i>intraLumaMode</i>	Prediction mode of the Intra_16x16 prediction process for luma samples. It is defined in the same way as in <a href="#">PredictIntra_16x16_H264</a> function.
<i>cbp4x4</i>	Coded block pattern. If <i>cbp4x4</i> & IPPVC_CBP_LUMA_DC is not equal to 0, 4x4 DC luma block is not zero-filled and it exists in <i>ppSrcDstCoeff</i> . If <i>cbp4x4</i> & (1 << (IPPVC_CBP_1ST_LUMA_AC_BITPOS + <i>i</i> )) is not equal to 0 ( $0 \leq i < 16$ ), <i>i</i> -th 4x4 AC luma block is not zero-filled and it exists in <i>ppSrcDstCoeff</i> .
<i>QP</i>	Quantization parameter ( $QP_Y$ in [JVTG050]). It must be within the range [0;51].
<i>edgeType</i>	Flag that specifies the availability of the macroblocks used for prediction. If the upper macroblock is not available for 16x16 intra prediction, <i>edgeType</i> &IPPVC_TOP_EDGE must be non-zero. If the left macroblock is not available for 16x16 intra prediction, <i>edgeType</i> &IPPVC_LEFT_EDGE must be non-zero. If the upper-left macroblock is not available for 16x16 intra prediction, <i>edgeType</i> &IPPVC_TOP_LEFT_EDGE must be non-zero.
<i>pQuantTable</i>	Pointer to the quantization table ( <i>LevelScale</i> ( <i>qp%6</i> , <i>i</i> , <i>j</i> ) in [JVTG050]).
<i>bypassFlag</i>	Flag enabling lossless coding.
<i>pReconstructInfo</i>	Pointer to the <i>IppiReconstructHighMB_32s16u</i> structure.

## Description

The functions *ippiReconstructLumaIntra\_16x16MB\_H264\_16s8u\_C1R* and *ippiReconstructLumaIntra16x16\_H264High\_32s16u\_IP1R* are declared in the *ippvc.h* file. These functions reconstruct 16x16 intra luma macroblock for high profile:

- Perform integer inverse transformation and dequantization for 4x4 luma DC coefficients in accordance with 8.5.8 of [JVTG050].
- Perform scaling, integer inverse transformation, and shift for 4x4 AC blocks in accordance with 8.5.10 of [ITUH264]. This process is performed on all 4x4 AC blocks in the same order as Figure 6-10 of [ITUH264] shows.
- Perform intra prediction for a 16x16 luma component in accordance with 8.3.3 of [ITUH264] in the same way as [PredictIntra\\_16x16\\_H264](#) function does.
- Perform adding of 16x16 prediction blocks and 16x16 residual blocks in accordance with 8-297 of [ITUH264].

These functions are used in the H.264 decoder included into Intel IPP Samples. See [introduction](#) to the H.264 section.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if at least one of the specified pointers is NULL.
<code>ippStsOutOfRangeErr</code>	<i>QP</i> is less than 0 or greater than 51.

## Deblocking Filtering

Note that the functions with `_IP1R` descriptor use the following structures:

```
typedef struct _IppiFilterDeblock_16u
```

```
{
    Ipp16u*   pSrcDstPlane;
    Ipp32s    srcDstStep;
    Ipp16u*   pAlpha;
    Ipp16u*   pBeta;
    Ipp16u*   pThresholds;
    Ipp8u*    pBs;
    Ipp32s    bitDepth;
} IppiFilterDeblock_16u;
```

```
typedef struct _IppiFilterDeblock_8u
```

```
{
    Ipp8u*    pSrcDstPlane;
    Ipp32s    srcDstStep;
    Ipp8u*    pAlpha;
    Ipp8u*    pBeta;
    Ipp8u*    pThresholds;
    Ipp8u*    pBs;
} IppiFilterDeblock_8u;
```

with parameters:

<i>pSrcDstPlane</i>	Pointer to the upper left pixel of the macroblock and resultant samples.
<i>srcDstStep</i>	Distance in items between starts of the consecutive lines in the source/destination plane.
<i>pAlpha</i>	Alpha Thresholds.
<i>pBeta</i>	Beta Thresholds.

<i>pThresholds</i>	Thresholds ( $T_{C0}$ ).
<i>pBs</i>	BS parameters.
<i>bitDepth</i>	Number of bits of the plane sample with range [8.. 14].

Number of elements in the arrays the *pAlpha*, *pBeta*, *pThresholds*, and *pBs* pointers point to may vary from function to function.

## FilterDeblockingLuma\_VerEdge\_H264

*Performs deblocking filtering on the vertical edges of the 16x16 luma macroblock.*

### Syntax

```

IppStatus ippiFilterDeblockingLuma_VerEdge_H264_8u_C1IR(Ipp8u* pSrcDst, Ipp32s srcdstStep,
const Ipp8u* pAlpha, const Ipp8u* pBeta, const Ipp8u* pThresholds, const Ipp8u* pBs);

IppStatus ippiFilterDeblockingLumaVerEdge_H264_16u_C1IR(const IppiFilterDeblock_16u*
pDeblockInfo);

```

### Parameters

<i>pSrcDst</i>	Pointer to the initial and resultant coefficients.
<i>srcdstStep</i>	Distance in items between starts of the consecutive lines in the array.
<i>pAlpha</i>	Array of size 2 of Alpha Thresholds (values for external and internal vertical edge).
<i>pBeta</i>	Array of size 2 of Beta Thresholds (values for external and internal vertical edge).
<i>pThresholds</i>	Array of size 16 of Thresholds ( $T_{C0}$ ) (values for the left edge of each 4x4 block).
<i>pBs</i>	Array of size 16 of BS parameters (values for the left edge of each 4x4 block).
<i>pDeblockInfo</i>	Pointer to the IppiFilterDeblock_16u structure.

### Description

The functions `ippiFilterDeblockingLuma_VerEdge_H264_8u_C1IR` and `ippiFilterDeblockingLumaVerEdge_H264_16u_C1IR` are declared in the `ippvc.h` file. These functions perform Deblocking Filtering on the vertical edges of the 16x16 luma macroblock in accordance with 8.7.2 of [JVTG050].

The function uses arrays *pAlpha*, *pBeta*, *pBs*, *pThresholds* as input arguments, where *pAlpha*[0], *pBeta*[0] are values for the external vertical edge, and *pAlpha*[1], *pBeta*[1] are values for the internal vertical edge. See [Figure "Arrangement of pThresholds Array Elements into a macroblock"](#) for the arrangement of *pThresholds* and *pBs* array elements.

Values of the arrays are calculated as follows:

- *pBs* values are calculated as per 8.7.2.1 of [JVTG050] and may take the following values: 0 - if no edge is filtered; [1,3] - if filtering is weak; 4 - if filtering is strong.
- *pAlpha* values are calculated from the formulas 8-326, 8-327 and Table 8-14 of [JVTG050].

- *pBeta* values are calculated from the formulas 8-326, 8-328 and Table 8-14 of [JVTG050].
- *pThresholds[i]* values are calculated from the formulas 8-326, 8-327, values of *pBs* array and Table 8-15 of [JVTG050].

Arrangement of *pThresholds* Array Elements into a macroblock

0	4	8	12
1	5	9	13
2	6	10	14
3	7	11	15

This function is used in the H.264 decoder and encoder included into Intel IPP Samples. See [introduction](#) to the H.264 section.

Return Values

<i>ippStsNoErr</i>	Indicates no error.
<i>ippStsNullPtrErr</i>	Indicates an error condition if at least one of the specified pointers is NULL.

FilterDeblockingLuma\_VerEdge\_MBAFF\_H264

*Performs deblocking filtering on the external vertical edges of half of 16X16 luma macroblock.*

Syntax

```
IppStatus ippiFilterDeblockingLuma_VerEdge_MBAFF_H264_8u_C1IR(Ipp8u* pSrcDst, Ipp32s srcdstStep, Ipp32u nAlpha, Ipp32u nBeta, const Ipp8u* pThresholds, const Ipp8u* pBs);  
IppStatus ippiFilterDeblockingLumaVerEdgeMBAFF_H264_16u_C1IR(const IppiFilterDeblock_16u* pDeblockInfo);
```

Parameters

<i>pSrcDst</i>	Pointer to the initial and resultant coefficients.
<i>srcdstStep</i>	Distance in items between starts of the consecutive lines in the array.
<i>pAlpha</i>	Alpha Threshold.
<i>pBeta</i>	Beta Threshold.
<i>pThresholds</i>	Array of size 8 of Thresholds ( $T_{C0}$ ) (two values for the left edge of each 4x4 block).



<i>pBs</i>	Array of size 8 of BS parameters (two values for the left edge of each 4x4 block).
<i>pDeblockInfo</i>	Pointer to the <i>IppiFilterDeblock_16u</i> structure.

## Description

The functions `ippiFilterDeblockingLuma_VerEdge_MBAFF_H264_8u_C1IR` and `ippiFilterDeblockingLumaVerEdgeMBAFF_H264_16u_C1IR` are declared in the `ippvc.h` file. These functions perform Deblocking Filtering on the vertical edges of the 16x16 luma macroblock in accordance with 8.7.2 of [JVTG050].

In process of filling *pThresholds* and *pBs* parameters for deblocking of each 4x4 block, a number of parameters of the neighboring blocks are taken into account. In MBAFF mode, the edge blocks may border on two others when the left and the current macroblocks are coded differently: one as a field, the other as a frame, and vice versa. As a result, two values of *pThresholds* and *pBs* parameters are taken for each block. For higher flexibility, the function processes only half of the macroblock to fit both types of the macroblocks, that is, coded as a field and coded as a frame.

The functions use arrays *nAlpha*, *nBeta*, *pBs*, *pThresholds* as input arguments. See [Figure "Arrangement of pThresholds Array Elements into a macroblock"](#) for the arrangement of *pThresholds* and *pBs* array elements.

Values of the arrays are calculated as follows:

- *pBs* values are calculated as per 8.7.2.1 of [JVTG050] and may take the following values: 0 - if no edge is filtered; [1,3] - if filtering is weak; 4 - if filtering is strong.
- *nAlpha* values are calculated from the formulas 8-326, 8-327 and Table 8-14 of [JVTG050].
- *nBeta* values are calculated from the formulas 8-326, 8-328 and Table 8-14 of [JVTG050].
- *pThresholds[i]* values are calculated from the formulas 8-326, 8-327, values of *pBs* array and Table 8-15 of [JVTG050].

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if at least one of the specified pointers is NULL.

## FilterDeblockingLuma\_HorEdge\_H264

*Performs deblocking filtering on the horizontal edges of the 16X16 luma macroblock.*

## Syntax

```
IppStatus ippiFilterDeblockingLuma_HorEdge_H264_8u_C1IR(Ipp8u* pSrcDst, Ipp32s srcdstStep,
const Ipp8u* pAlpha, const Ipp8u* pBeta, const Ipp8u* pThresholds, const Ipp8u* pBs);
IppStatus ippiFilterDeblockingLumaHorEdge_H264_16u_C1IR(const IppiFilterDeblock_16u*
pDeblockInfo);
```

## Parameters

<i>pSrcDst</i>	Pointer to the initial and resultant coefficients.
<i>srcdstStep</i>	Distance in items between starts of the consecutive lines in the array.

<i>pAlpha</i>	Array of size 2 of Alpha Thresholds (values for external and internal horizontal edge).
<i>pBeta</i>	Array of size 2 of Beta Thresholds (values for external and internal horizontal edge).
<i>pThresholds</i>	Array of size 16 of Thresholds ( $T_{C0}$ ) (values for upper edge of each 4x4 block).
<i>pBs</i>	Array of size 16 of BS parameters (values for upper edge of each 4x4 block).
<i>pDeblockInfo</i>	Pointer to the <i>IppiFilterDeblock_16u</i> structure.

### Description

The functions `ippiFilterDeblockingLuma_HorEdge_H264_8u_C1IR` and `ippiFilterDeblockingLuma_HorEdge_H264_16u_C1IR` are declared in the `ippvc.h` file. These functions perform Deblocking Filtering on the horizontal edges of the 16x16 luma macroblock in accordance with 8.7.2 of [JVTG050].

The functions use arrays *pAlpha*, *pBeta*, *pBs*, *pThresholds* as input arguments, where *pAlpha*[0], *pBeta*[0] are values for the external horizontal edge, and *pAlpha*[1], *pBeta*[1] are values for the internal horizontal edge. See [Figure "Arrangement of \*pThresholds\* Array Elements into a Macroblock"](#) for the arrangement of *pThresholds* and *pBs* array elements.

Values of the arrays are calculated as follows:

- *pBs* values are calculated as per 8.7.2.1 of [JVTG050] and may take the following values: 0 - if no edge is filtered; [1,3] - if filtering is weak; 4 - if filtering is strong.
- *pAlpha* values are calculated from the formulas 8-326, 8-327 and Table 8-14 of [JVTG050].
- *pBeta* values are calculated from the formulas 8-326, 8-328 and Table 8-14 of [JVTG050].
- *pThresholds*[*i*] values are calculated from the formulas 8-326, 8-327, values of *pBs* array and Table 8-15 of [JVTG050].

### Arrangement of *pThresholds* Array Elements into a Macroblock

---

0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15

---

This function is used in the H.264 decoder and encoder included into Intel IPP Samples. See [introduction](#) to the H.264 section.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if at least one of the specified pointers is NULL.

## FilterDeblockingChroma\_VerEdge\_H264, FilterDeblockingChroma422VerEdge\_H264

*Performs deblocking filtering on the vertical edges of a chroma macroblock.*

## Syntax

```

IppStatus ippiFilterDeblockingChroma_VerEdge_H264_8u_C1IR(Ipp8u* pSrcDst, Ipp32s srcdstStep,
const Ipp8u* pAlpha, const Ipp8u* pBeta, const Ipp8u* pThresholds, const Ipp8u* pBs);

IppStatus ippiFilterDeblockingChroma_VerEdge_H264_8u_C2IR(Ipp8u* pSrcDst, Ipp32s srcdstStep,
const Ipp8u* pAlpha, const Ipp8u* pBeta, const Ipp8u* pThresholds, const Ipp8u* pBs);

IppStatus ippiFilterDeblockingChroma_VerEdge_H264_8u_C2I(const IppiFilterDeblock_8u*
pDeblockInfo);

IppStatus ippiFilterDeblockingChroma_VerEdge_H264_16u_C1IR(const IppiFilterDeblock_16u*
pDeblockInfo);

IppStatus ippiFilterDeblockingChroma422VerEdge_H264_8u_C1IR(const IppiFilterDeblock_8u*
pDeblockInfo);

IppStatus ippiFilterDeblockingChroma422VerEdge_H264_16u_C1IR(const IppiFilterDeblock_16u*
pDeblockInfo);

```

## Parameters

<code>pSrcDst</code>	<p>Pointer to the initial and resultant coefficients.</p> <p>For <code>FilterDeblockingChroma_VerEdge_H264_8u_C2IR</code> and <code>FilterDeblockingChroma_VerEdge_H264_8u_C2I</code> the coefficients are in the NV12 format:</p> <pre> 0 UVUVUVUVUVUVUVUV 1 UVUVUVUVUVUVUVUV ... 7 UVUVUVUVUVUVUVUV </pre>
<code>srcdstStep</code>	Distance in items between starts of the consecutive lines in the array.
<code>pAlpha</code>	<p>Array of size 2 of Alpha Thresholds (values for external and internal vertical edge). For <code>FilterDeblockingChroma_VerEdge_H264_8u_C2I</code>, array of size 4 of Alpha Thresholds (values for external and internal vertical edge: 2 for U plane - Alpha [0], Alpha [1] and 2 to V plane - Alpha [2], Alpha [3]).</p>

<i>pBeta</i>	Array of size 2 of Beta Thresholds (values for external and internal vertical edge). For <code>FilterDeblockingChroma_VerEdge_H264_8u_C2I</code> , array of size 4 of Beta Thresholds (values for external and internal vertical edge: 2 for U plane - <code>Beta [0]</code> , <code>Beta[1]</code> and 2 to V plane - <code>Beta[2]</code> , <code>Beta[3]</code> ).
<i>pThresholds</i>	Array of size 8 of Thresholds ( $T_{CO}$ ) (values for the left edge of each 2x2 block). For <code>FilterDeblockingChroma_VerEdge_H264_8u_C2I</code> , array of size 16 of Thresholds ( $T_{CO}$ ) (values for the left edge of each 8x2 - <code>[0,7]</code> for U plane and <code>[8,15]</code> for V plane).
<i>pBs</i>	Array of size 16 of BS parameters (values for the left edge of each 2x2 block). For <code>FilterDeblockingChroma_VerEdge_H264_8u_C2I</code> , array of size 16 of BS parameters (values for the left edge of each 8x2 block - the same for both U and V planes).
<i>pDeblockInfo</i>	Pointer to the <code>IppiFilterDeblock_8u</code> or <code>IppiFilterDeblock_16u</code> structure.

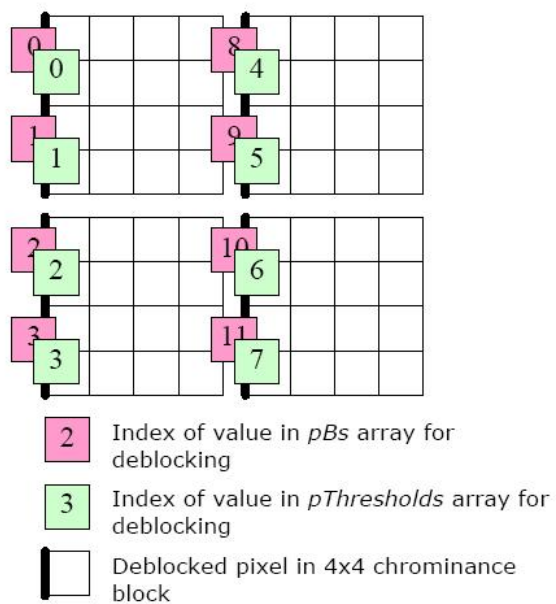
## Description

The functions are declared in the `ippvc.h` file. These functions perform Deblocking Filtering on the vertical edge of an 8x8 (`_C1IR`) or 16x8 (`_C2IR`, `_C2I`) chroma macroblock in accordance with 8.7.2 of [JVTG050]. `FilterDeblockingChroma_VerEdge_H264_8u_C2IR` and `FilterDeblockingChroma_VerEdge_H264_8u_C2I` work with NV12 chroma macroblocks.

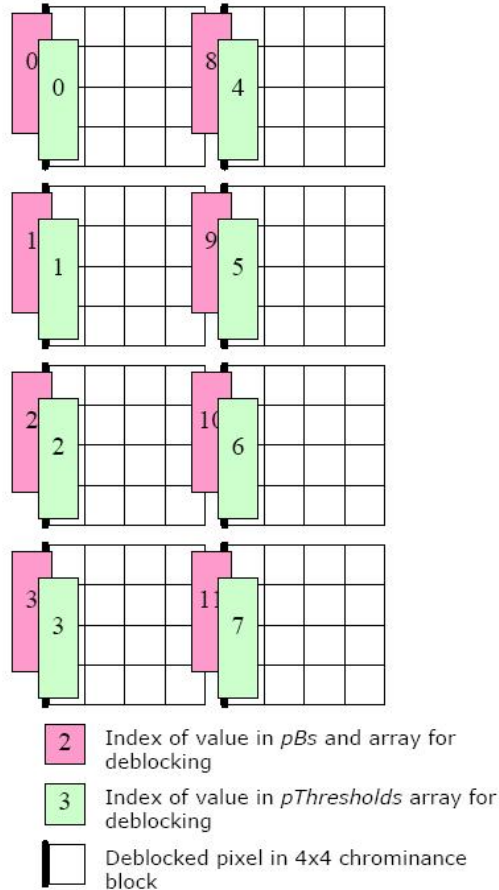
The functions use arrays *pAlpha*, *pBeta*, *pBs*, *pThresholds* as input arguments. *pAlpha*, *pBeta*, and *pBs* are the same arrays as in `FilterDeblockingLuma_VerEdge_H264` function. *pAlpha*[0], *pBeta*[0] are values for the external vertical edge, and *pAlpha*[1], *pBeta*[1] are values for the internal vertical edge.

For the `FilterDeblockingChroma_VerEdge_H264_8u_C2I` function, *pAlpha*[0], *pAlpha*[2], *pBeta*[0], *pBeta*[2] are values for the external vertical edge, and *pAlpha*[1], *pAlpha*[3], *pBeta*[1], *pBeta*[3] are values for the internal vertical edge.

See [Figure "Arrangement of \*pBs\* and \*pThresholds\* Array Elements for 4:2:0 format"](#) for the arrangement of *pBs* and *pThresholds* array elements into an 8x8 chroma block of 4:2:0 format and [Figure "Arrangement of \*pBs\* and \*pThresholds\* Array Elements for 4:2:2 format"](#) for their arrangement into an 8x16 block of 4:2:2 format.

**Arrangement of  $pBs$  and  $pThresholds$  Array Elements for 4:2:0 format**

### Arrangement of *pBs* and *pThresholds* Array Elements for 4:2:2 format



As two vertical edges are filtered for the chroma component, the function uses *pBs* array elements with indices [0,3] for the external edge and [8,11] for the internal edge.

Values of the arrays are calculated as follows:

- *pBs* values are calculated as per 8.7.2.1 of [JVTG050] and may take the following values: 0 - if no edge is filtered; [1,3] - if filtering is weak; 4 - if filtering is strong.
- *pAlpha* values are calculated from the formulas 8-326, 8-327 and Table 8-14 of [JVTG050].
- *pBeta* values are calculated from the formulas 8-326, 8-328 and Table 8-14 of [JVTG050].
- *pThresholds*[*i*] values are calculated from the formulas 8-326, 8-327, values of *pBs* array and Table 8-15 of [JVTG050].

## Example

The code example below shows how to use `ippiFilterDeblockingChroma_VerEdge_H264_8u_C1IR` function.

```
{
    Ipp8u    pSrcDst[8*8] =
    {
        0x78, 0x77, 0x76, 0x74, 0x73, 0x73, 0x74, 0x75,
        0x78, 0x77, 0x76, 0x74, 0x73, 0x73, 0x75, 0x79,
        0x76, 0x76, 0x76, 0x75, 0x75, 0x75, 0x7c, 0x7e,
        0x75, 0x75, 0x75, 0x76, 0x76, 0x76, 0x7c, 0x7e,
        0x78, 0x79, 0x7a, 0x7c, 0x7d, 0x7e, 0x81, 0x82,
        0x78, 0x79, 0x7a, 0x7c, 0x7d, 0x7e, 0x81, 0x82,
        0x7a, 0x7b, 0x7c, 0x7e, 0x7e, 0x7f, 0x81, 0x82,
        0x7d, 0x7e, 0x7f, 0x81, 0x81, 0x82, 0x82, 0x82
    };
    Ipp8u    pAlpha[2]      = {25,25};
    Ipp8u    pBeta[2]       = {8,8};
    Ipp8u    pThreshold[16] = {255,255,255,255,255,255,1,1,204,204,204,204,204,204,204,204};
    Ipp8u    pBs[16]        = {0,0,0,0,0,0,0,2,0,0,1,2,0,0,0,0};

    ippiFilterDeblockingChroma_VerEdge_H264_8u_C1IR (pSrcDst,
                                                    8,
                                                    pAlpha,
                                                    pBeta,
                                                    pThreshold,
                                                    pBs);
}
```

These functions are used in the H.264 decoder and encoder included into Intel IPP Samples. See [introduction](#) to the H.264 section.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if at least one of the specified pointers is NULL.

## FilterDeblockingChroma\_VerEdge\_MBAFF\_H264, FilterDeblockingChroma422VerEdgeMBAFF\_H264

*Performs deblocking filtering on the vertical edges of half of 8X8 chroma macroblock.*

## Syntax

```
IppStatus ippiFilterDeblockingChroma_VerEdge_MBAFF_H264_8u_C1IR(Ipp8u* pSrcDst, Ipp32s
srcdstStep, Ipp32u nAlpha, Ipp32u nBeta, const Ipp8u* pThresholds, const Ipp8u* pBs);

IppStatus ippiFilterDeblockingChromaVerEdgeMBAFF_H264_16u_C1IR(const IppiFilterDeblock_16u*
pDeblockInfo);

IppStatus ippiFilterDeblockingChroma422VerEdgeMBAFF_H264_8u_C1IR(const IppiFilterDeblock_8u*
pDeblockInfo);

IppStatus ippiFilterDeblockingChroma422VerEdgeMBAFF_H264_16u_C1IR(const
IppiFilterDeblock_16u* pDeblockInfo);
```

## Parameters

<i>pSrcDst</i>	Pointer to the initial and resultant coefficients.
<i>srcdstStep</i>	Distance in items between starts of the consecutive lines in the array.
<i>pAlpha</i>	Alpha Threshold.
<i>pBeta</i>	Beta Threshold.
<i>pThresholds</i>	Array of size 4 of Thresholds ( $T_{C0}$ ) (two values for the left edge of each 2x2 block).
<i>pBs</i>	Array of size 4 of BS parameters (two values for the left edge of each 2x2 block).
<i>pDeblockInfo</i>	Pointer to the <code>IppiFilterDeblock_16u</code> structure.

## Description

The functions `ippiFilterDeblockingChroma_VerEdge_MBAFF_H264_8u_C1IR`, `ippiFilterDeblockingChromaVerEdgeMBAFF_H264_16u_C1IR`, `ippiFilterDeblockingChroma422VerEdgeMBAFF_H264_8u_C1IR`, and `ippiFilterDeblockingChroma422VerEdgeMBAFF_H264_16u_C1IR` are declared in the `ippvc.h` file. These functions perform Deblocking Filtering on the vertical edge of the 8x8 chroma macroblock in accordance with 8.7.2 of [JVTG050].

In process of filling *pThresholds* and *pBs* parameters for deblocking of each 2x2 block, a number of parameters of the neighboring blocks are taken into account. In MBAFF mode, the edge blocks may border on two others when the left and the current macroblocks are coded differently: one as a field, the other as a frame, and vice versa. As a result, two values of *pThresholds* and *pBs* parameters are taken for each block. For higher flexibility, the functions process only half of the macroblock to fit both types of the macroblocks, that is, coded as a field and coded as a frame.

The functions use arrays *nAlpha*, *nBeta*, *pBs*, *pThresholds* as input arguments. *nAlpha*, *nBeta*, and *pBs* are the same arrays as in `FilterDeblockingLuma_VerEdge_H264` function. See [Figure "Arrangement of \*pBs\* and \*pThresholds\* Array Elements for 4:2:0 format"](#) for the arrangement of *pBs* and *pThresholds* array elements into an 8x8 chroma block of 4:2:0 format and [Figure "Arrangement of \*pBs\* and \*pThresholds\* Array Elements for 4:2:2 format"](#) for their arrangement into an 8x16 block of 4:2:2 format.

Values of the arrays are calculated as follows:

- *pBs* values are calculated as per 8.7.2.1 of [JVTG050] and may take the following values: 0 - if no edge is filtered; [1,3] - if filtering is weak; 4 - if filtering is strong.
- *nAlpha* values are calculated from the formulas 8-326, 8-327 and Table 8-14 of [JVTG050].
- *nBeta* values are calculated from the formulas 8-326, 8-328 and Table 8-14 of [JVTG050].
- *pThresholds[i]* values are calculated from the formulas 8-326, 8-327, values of *pBs* array and Table 8-15 of [JVTG050].

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if at least one of the specified pointers is NULL.



## FilterDeblockingChroma\_HorEdge\_H264, FilterDeblockingChroma422HorEdge\_H264

*Performs deblocking filtering on the horizontal edges of a chroma macroblock.*

### Syntax

```
IppStatus ippiFilterDeblockingChroma_HorEdge_H264_8u_C1IR(Ipp8u* pSrcDst, Ipp32s srcdstStep,
const Ipp8u* pAlpha, const Ipp8u* pBeta, const Ipp8u* pThresholds, const Ipp8u* pBs);

IppStatus ippiFilterDeblockingChroma_HorEdge_H264_8u_C2IR(Ipp8u* pSrcDst, Ipp32s srcdstStep,
const Ipp8u* pAlpha, const Ipp8u* pBeta, const Ipp8u* pThresholds, const Ipp8u* pBs);

IppStatus ippiFilterDeblockingChroma_HorEdge_H264_8u_C2I(const IppiFilterDeblock_8u*
pDeblockInfo);

IppStatus ippiFilterDeblockingChroma_HorEdge_H264_16u_C1IR(const IppiFilterDeblock_16u*
pDeblockInfo);

IppStatus ippiFilterDeblockingChroma422HorEdge_H264_8u_C1IR(const IppiFilterDeblock_8u*
pDeblockInfo);

IppStatus ippiFilterDeblockingChroma422HorEdge_H264_16u_C1IR(const IppiFilterDeblock_16u*
pDeblockInfo);
```

### Parameters

<i>pSrcDst</i>	<p>Pointer to the initial and resultant coefficients.</p> <p>For <code>FilterDeblockingChroma_HorEdge_H264_8u_C2IR</code> and <code>FilterDeblockingChroma_HorEdge_H264_8u_C2I</code> the coefficients are in the NV12 format:</p> <pre>0 UVUVUVUVUVUVUVUV 1 UVUVUVUVUVUVUVUV ... 7 UVUVUVUVUVUVUVUV</pre>
<i>srcdstStep</i>	Distance in items between starts of the consecutive lines in the array.
<i>pAlpha</i>	<p>Array of size 2 of Alpha Thresholds (values for external and internal horizontal edge). For <code>FilterDeblockingChroma_HorEdge_H264_8u_C2I</code>, array of size 4 of Alpha Thresholds (values for external and internal vertical edge: 2 for U plane - Alpha [0], Alpha [1] and 2 to V plane - Alpha [2], Alpha [3]).</p>
<i>pBeta</i>	<p>Array of size 2 of Beta Thresholds (values for external and internal horizontal edge). For <code>FilterDeblockingChroma_HorEdge_H264_8u_C2I</code>, array of size 4 of Beta Thresholds (values for external and internal vertical edge: 2 for U plane - Beta [0], Beta[1] and 2 to V plane - Beta[2], Beta[3]).</p>

<i>pThresholds</i>	Array of size 8 of Thresholds ( $T_{c0}$ ) (values for upper edge of each 2x2 block). For <code>FilterDeblockingChroma_HorEdge_H264_8u_C2I</code> , an array of size 16 of Thresholds ( $T_{c0}$ ) (values for the upper edge of each 4x4 - [0,7] for U plane and [8,15] for V plane).
<i>pBs</i>	Array of size 16 of BS parameters (values for upper edge of each 2x2 block). For <code>FilterDeblockingChroma_HorEdge_H264_8u_C2I</code> , an array of size 16 of BS parameters (values for upper edge of each 4x4 block - the same for both U and V planes).
<i>pDeblockInfo</i>	Pointer to the <code>IppiFilterDeblock_8u</code> or <code>IppiFilterDeblock_16u</code> structure.

## Description

The functions are declared in the `ippvc.h` file. These functions perform Deblocking Filtering on the horizontal edge of an 8x8 (`_C1IR`) or 16x8 (`_C2IR`, `_C2I`) chroma macroblock in accordance with 8.7.2 of [JVTG050]. `FilterDeblockingChroma_HorEdge_H264_8u_C2IR` and `FilterDeblockingChroma_HorEdge_H264_8u_C2I` work with NV12 chroma macroblocks.

The functions use arrays *pAlpha*, *pBeta*, *pBs*, *pThresholds* as input arguments. *pAlpha*, *pBeta*, and *pBs* are the same arrays as in `FilterDeblockingLuma_HorEdge_H264` function. *pAlpha*[0], *pBeta*[0] are values for the external horizontal edge, and *pAlpha*[1], *pBeta*[1] are values for the internal horizontal edge.

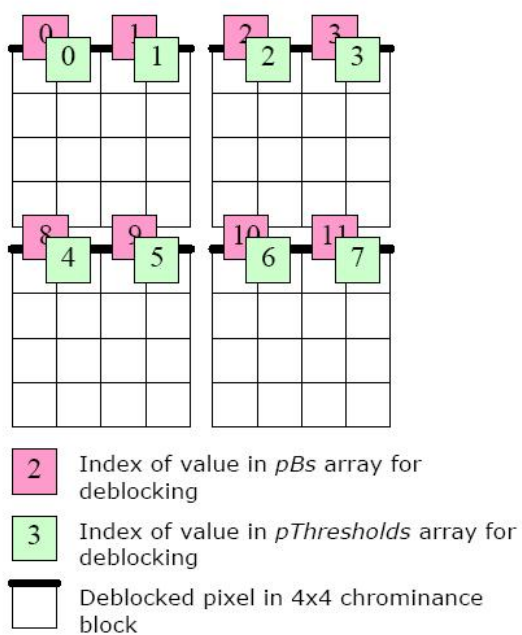
For the `FilterDeblockingChroma_HorEdge_H264_8u_C2I` function, *pAlpha*[0], *pAlpha*[2], *pBeta*[0], *pBeta*[2] are values for the external vertical edge, and *pAlpha*[1], *pAlpha*[3], *pBeta*[1], *pBeta*[3] are values for the internal vertical edge.

See [Figure "Arrangement of \*pBs\* and \*pThresholds\* Array Elements for 4:2:0 format"](#) for the arrangement of *pBs* and *pThresholds* array elements into an 8x8 chroma block of 4:2:0 format and [Figure "Arrangement of \*pBs\* and \*pThresholds\* Array Elements for 4:2:2 format"](#) for their arrangement into an 8x16 block of 4:2:2 format.

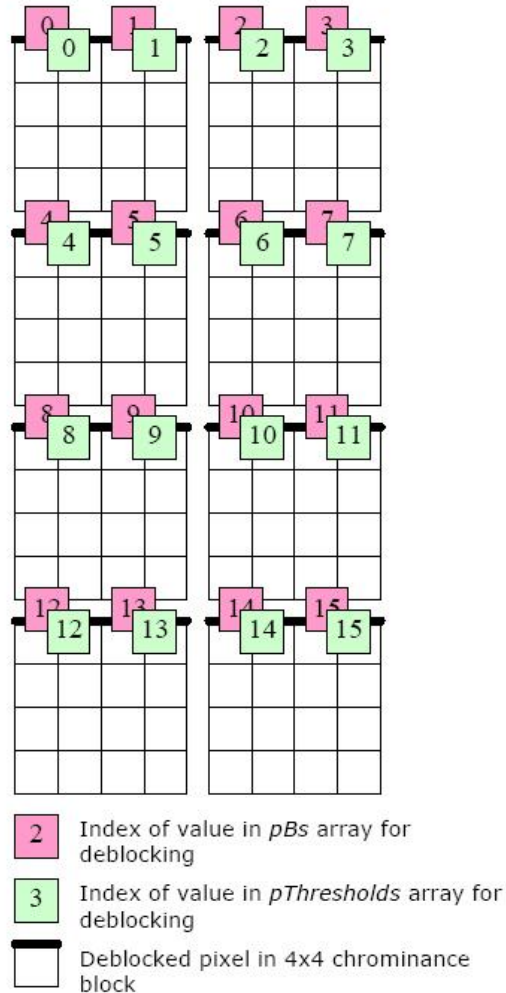
---

**Arrangement of  $pBs$  and  $pThresholds$  Array Elements for 4:2:0 format**

---



### Arrangement of *pBs* and *pThresholds* Array Elements for 4:2:2 format



As two horizontal edges are filtered for the chroma component, the function uses *pBs* array elements with indices [0,3] for the external edge and [8,11] for the internal edge.

Values of the arrays are calculated as follows:

- *pBs* values are calculated as per 8.7.2.1 of [JVTG050] and may take the following values: 0 - if no edge is filtered; [1,3] - if filtering is weak; 4 - if filtering is strong.
- *pAlpha* values are calculated from the formulas 8-326, 8-327 and Table 8-14 of [JVTG050].
- *pBeta* values are calculated from the formulas 8-326, 8-328 and Table 8-14 of [JVTG050].
- *pThresholds*[*i*] values are calculated from the formulas 8-326, 8-327, values of *pBs* array and Table 8-15 of [JVTG050].

These functions are used in the H.264 decoder and encoder included into Intel IPP Samples. See [introduction](#) to the H.264 section.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if at least one of the specified pointers is NULL.

## Example

The code example below shows how to use `FilterDeblockingChroma_HorEdge_H264_8u_C1IR` function.

```
{
    Ipp8u    pSrcDst[8*8] =
    {
        0x78, 0x77, 0x76, 0x74, 0x73, 0x73, 0x74, 0x75,
        0x78, 0x77, 0x76, 0x74, 0x73, 0x73, 0x75, 0x79,
        0x76, 0x76, 0x76, 0x75, 0x75, 0x75, 0x7c, 0x7e,
        0x75, 0x75, 0x75, 0x76, 0x76, 0x76, 0x7c, 0x7e,
        0x78, 0x79, 0x7a, 0x7c, 0x7d, 0x7e, 0x81, 0x82,
        0x78, 0x79, 0x7a, 0x7c, 0x7d, 0x7e, 0x81, 0x82,
        0x7a, 0x7b, 0x7c, 0x7e, 0x7e, 0x7f, 0x81, 0x82,
        0x7d, 0x7e, 0x7f, 0x81, 0x81, 0x82, 0x82, 0x82
    };
    Ipp8u    pAlpha[2]      = {25,25};
    Ipp8u    pBeta[2]       = {8,8};
    Ipp8u    pThreshold[16] = {255,255,255,255,255,255,1,1,204,204,204,204,204,204,204,204};
    Ipp8u    pBs[16]        = {0,0,0,0,0,0,0,2,0,0,1,2,0,0,0,0};

    ippStaticInitCpu(ippCpuSSE2);

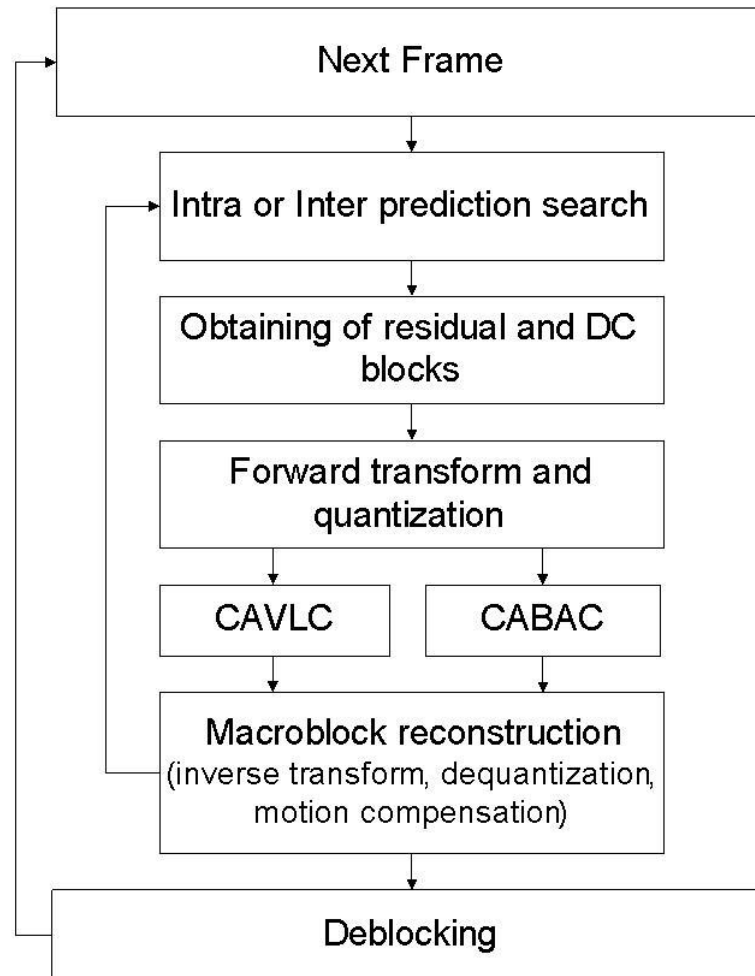
    ippiFilterDeblockingChroma_HorEdge_H264_8u_C1IR(pSrcDst,
                                                    8,
                                                    pAlpha,
                                                    pBeta,
                                                    pThreshold,
                                                    pBs);
}
```

## H.264 Encoder Functions

This section describes the functions for H.264 Encoder in accordance with JVT-G050 ([JVTG050]) standard.

### H.264 Encoder Structure

---



The edges detection function ([EdgesDetect16x16](#)) is taken from the [General Functions](#) category. This function helps to choose a mode of intra macroblock coding: INTRA 16X16 or INTRA 4X4. If edges are detected inside the macroblock, both modes should be estimated. Otherwise, INTRA 16X16 mode should be selected.

Other general functions used by H.264 Encoder for predicted blocks estimation and for obtaining residual and DC blocks are [SAD Functions](#), [Sum of Differences Evaluation functions](#), and the [GetDiff4x4](#) function.

H.264 Encoder uses the H.264 Decoder functions for calculation of [inter](#) and [intra predicted blocks](#), [Macroblock Reconstruction](#), and [Deblocking Filtering](#). Forward transform and quantization, and CAVLC coding are performed by Encoder proper functions, which are described below.

## Forward Transform and Quantization

### TransformQuantChromaDC\_H264

*Performs forward transform and quantization for 2X2 DC chroma blocks.*

---

#### Syntax

```
IppStatus ippiTransformQuantChromaDC_H264_16s_C1I(Ipp16s* pSrcDst, Ipp16s* pTBlock, Ipp32s QPChroma, Ipp8s* NumLevels, Ipp8u Intra, Ipp8u NeedTransform);
```

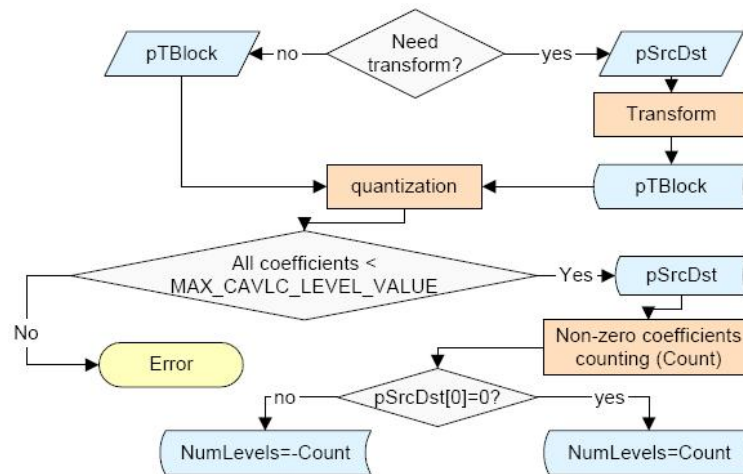
#### Parameters

<i>pSrcDst</i>	Pointer to a 2x2 chroma DC block - source and destination array of size 4.
<i>pTBlock</i>	Pointer to a 2x2 transformed chroma DC block - source or destination array of size 4.
<i>QPChroma</i>	Quantization parameter for chroma, in range [0, 39].
<i>NumLevels</i>	Pointer to a value that contains: <ul style="list-style-type: none"> <li>a negative value of a number of non-zero elements in block after quantization (when the first quantized element in block is not equal to zero),</li> <li>a number of non-zero elements in block after quantization (when the first quantized element in block is equal to zero).</li> </ul>
<i>Intra</i>	This value is calculated by the function. Flag equal to 1 in the case of intra macroblock, 0 otherwise.
<i>NeedTransform</i>	Flag; if equal to 1, the function is supposed to do DC coeffs transformation; if equal to 0, the DC coeffs passed to the function are already transformed.

## Description

The function `ippiTransformQuantChromaDC_H264_16s_C1I` is declared in the `ippvc.h` file. This function performs forward transform, if necessary, and quantization for a 2x2 DC chroma block as shown in the figure below:

### Forward Transform and Quantization for 2x2 Block



If *NeedTransform* is equal to 0, transformed DC block coefficients (*pTBlock*) are used for quantization.

If *NeedTransform* is equal to 1, at first a 2x2 chroma DC block *pSrcDst* is transformed and is saved in *pTBlock*. Then quantization process is performed on the transformed chroma DC block.

If all coefficients after quantization are not greater than `MAX_CAVLC_LEVEL_VALUE`, they are saved in *pSrcDst*. Otherwise, the function returns error.

This function is used in the H.264 encoder included into Intel IPP Samples. See [introduction](#) to the H.264 section.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if at least one of the specified pointers is NULL.
<code>ippStsOutOfRangeErr</code>	Indicates an error if <i>QPChroma</i> > 39 or <i>QPChroma</i> < 0.
<code>ippStsScaleRangeErr</code>	Indicates an error condition if any coefficient after quantization is greater than <code>MAX_CAVLC_LEVEL_VALUE</code> .



## TransformQuantLumaDC\_H264

*Performs forward transform and quantization for 4X4 DC luma blocks.*

### Syntax

```
IppStatus ippTransformQuantLumaDC_H264_16s_C1I(Ipp16s* pSrcDst, Ipp16s* pTBlock, Ipp32s
QP, Ipp8s* NumLevels, Ipp8u NeedTransform, const Ipp16s* pScanMatrix, Ipp8u* LastCoeff);
```

### Parameters

<i>pSrcDst</i>	Pointer to a 4x4 luma DC block - source and destination array of size 16.
<i>pTBlock</i>	Pointer to a 4x4 transformed luma DC block - source or destination array of size 16.
<i>QP</i>	Quantization parameter for luma, in range [0, 51].
<i>NumLevels</i>	Pointer to a value that contains: <ul style="list-style-type: none"> <li>a negative value of a number of non-zero elements in block after quantization (when the first quantized element in block is not equal to zero),</li> <li>a number of non-zero elements in block after quantization (when the first quantized element in block is equal to zero).</li> </ul> This value is calculated by the function.
<i>NeedTransform</i>	Flag; if equal to 1, the function is supposed to do DC coeffs transformation; if equal to 0, the DC coeffs passed to the function are already transformed.
<i>pScanMatrix</i>	Scan matrix for coefficients in block (array of size 16).
<i>LastCoeff</i>	Position of the last (in order of <i>pScanMatrix</i> ) non-zero coefficient in block after quantization. This value is calculated by the function.

### Description

The function `ippTransformQuantLumaDC_H264_16s_C1I` is declared in the `ippvc.h` file. This function performs forward transform, if necessary, and quantization for a 4x4 DC luma block as shown in [Figure "Forward Transform and Quantization for 2x2 Block"](#):

If *NeedTransform* is equal to 0, transformed DC block coefficients (*pTBlock*) are used for quantization.

If *NeedTransform* is equal to 1, at first a 4x4 luma DC block *pSrcDst* is transformed and is saved in *pTBlock*. Then quatization process is performed on the transformed luma DC block.

If all coefficients after quantization are not greater than `MAX_CAVLC_LEVEL_VALUE`, they are saved in *pSrcDst*. Otherwise, the function returns error.

This function is used in the H.264 encoder included into Intel IPP Samples. See [introduction](#) to the H.264 section.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if at least one of the specified pointers is NULL.
<code>ippStsOutOfRangeErr</code>	Indicates an error if $QP > 51$ or $QP < 0$ .
<code>ippStsScaleRangeErr</code>	Indicates an error condition if any coefficient after quantization is greater than <code>MAX_CAVLC_LEVEL_VALUE</code> .

## TransformResidual4x4Fwd\_H264

*Performs forward transform for 4X4 residual blocks.*

---

### Syntax

```

IppStatus ippiTransformResidual4x4Fwd_H264_16s_C1(const Ipp16s* pSrc, Ipp16s* pDst);
IppStatus ippiTransformResidual4x4Fwd_H264_32s_C1(const Ipp32s* pSrc, Ipp32s* pDst);

```

### Parameters

<code>pSrc</code>	Pointer to a 4x4 residual block - source array of size 16.
<code>pDst</code>	Pointer to a 4x4 residual block - destination array of size 16.

### Description

The function `ippiTransformResidual4x4Fwd_H264_16s_C1` and `ippiTransformResidual4x4Fwd_H264_32s_C1` are declared in the `ippvc.h` file. This function performs forward transform for a 4x4 residual block. A 4x4 residual block (`pSrc`) is transformed and saved in `pDst`.

This function is used in the H.264 encoder included into Intel IPP Samples. See [introduction](#) to the H.264 section.

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if at least one of the specified pointers is NULL.

## QuantizeResidual4x4Fwd\_H264

*Performs forward quantization for 4X4 residual blocks.*

---

### Syntax

```

IppStatus ippiQuantizeResidual4x4Fwd_H264_16s_C1(const Ipp16s* pSrc, Ipp16s* pDst, Ipp32u*
pNumNonZeros, Ipp32u* pLastNonZero, const Ipp16s* pQuantTable, const Ipp16s* pScanMatrix,
Ipp32s QP, Ipp32s roundMode);

IppStatus ippiQuantizeResidual4x4Fwd_H264_16s32s_C1(const Ipp16s* pSrc, Ipp16s* pDst,
Ipp32u* pNumNonZeros, Ipp32u* pLastNonZero, const Ipp32s* pQuantTable, const Ipp16s*
pScanMatrix, Ipp32s QP, Ipp32s roundMode);

```

```

IppStatus ippiQuantizeResidual4x4Fwd_H264_32s_C1(const Ipp32s* pSrc, Ipp32s* pDst, Ipp32u*
pNumNonZeros, Ipp32u* pLastNonZero, const Ipp32s* pQuantTable, const Ipp16s* pScanMatrix,
Ipp32s QP, Ipp32s roundMode);

```

## Parameters

<i>pSrc</i>	Pointer to a 4x4 residual block - source array of size 16.
<i>pDst</i>	Pointer to a 4x4 residual block - destination array of size 16.
<i>pNumNonZeros</i>	Pointer to a value that contains the number of non-zero elements in the block after quantization. This value is calculated by the function.
<i>pLastNonZero</i>	Position of the last (in order of <i>pScanMatrix</i> ) non-zero coefficient in the block after quantization. This value is calculated by the function.
<i>pQuantTable</i>	Pointer to a user's specific quantization table.
<i>pScanMatrix</i>	Scan matrix for coefficients in block (array of size 16).
<i>QP</i>	Quantization parameter for luma or chroma, in range [0,51] or [0,39].
<i>roundMode</i>	Rounding mode. 0 means inter frame is processed, otherwise - intra frame is processed.

## Description

The functions `ippiQuantizeResidual4x4Fwd_H264_16s_C1`, `ippiQuantizeResidual4x4Fwd_H264_16s32s_C1`, and `ippiQuantizeResidual4x4Fwd_H264_32s_C1` are declared in the `ippvc.h` file. These functions perform forward quantization for a 4x4 residual block. A 4x4 residual block (*pSrc*) is quantized. All values are rounded towards nearest (intra frame) or towards zero (inter frame). Then the number of non-zero elements and index of the last non-zero element are calculated and stored.

If *pQuantTable* is NULL, the default quantization table is used.

This function is used in the H.264 encoder included into Intel IPP Samples. See [introduction](#) to the H.264 section.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if at least one of the specified pointers is NULL.
<code>ippStsOutOfRangeErr</code>	Indicates an error if $QP > 51$ or $QP < 0$ .

## TransformQuantResidual\_H264

*Performs forward transform and quantization for 4X4 residual blocks.*

---

## Syntax

```

IppStatus ippiTransformQuantResidual_H264_16s_C1I(Ipp16s* pSrcDst, Ipp32s QP, Ipp8s*
NumLevels, Ipp8u Intra, const Ipp16s* pScanMatrix, Ipp8u* LastCoeff);

```

## Parameters

<i>pSrcDst</i>	Pointer to a 4x4 residual block - source and destination array of size 16.
<i>QP</i>	Quantization parameter for luma or for chroma, in range [0, 51] or [0, 39].
<i>NumLevels</i>	Pointer to a value that contains: <ul style="list-style-type: none"> <li>a negative value of a number of non-zero elements in block after quantization (when the first quantized element in block is not equal to zero),</li> <li>a number of non-zero elements in block after quantization (when the first quantized element in block is equal to zero).</li> </ul> <p>This value is calculated by the function.</p>
<i>Intra</i>	Flag equal to 1 in the case of intra slice, 0 otherwise.
<i>pScanMatrix</i>	Scan matrix for coefficients in block (array of size 16).
<i>LastCoeff</i>	Position of the last (in order of <i>pScanMatrix</i> ) non-zero coefficient in block after quantization. This value is calculated by the function.

## Description

The function `ippiTransformQuantResidual_H264_16s_C1I` is declared in the `ippvc.h` file. This function performs forward transform and quantization for a 4x4 residual block.

A 4x4 residual block (*pSrcDst*) is transformed. Then quatization process is performed on the transformed residual block.

If all coefficients after quantization are not greater than `MAX_CAVLC_LEVEL_VALUE`, they are saved in *pSrcDst*. Otherwise, the function returns error.

This function is used in the H.264 encoder included into Intel IPP Samples. See [introduction](#) to the H.264 section.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if at least one of the specified pointers is NULL.
<code>ippStsOutOfRangeErr</code>	Indicates an error if $QP > 51$ or $QP < 0$ .
<code>ippStsScaleRangeErr</code>	Indicates an error condition if any coefficient after quantization is greater than <code>MAX_CAVLC_LEVEL_VALUE</code> .

## TransformLuma8x8Fwd\_H264

*Performs forward 8X8 transform for 8X8 luma blocks without normalization.*

---

## Syntax

```
IppStatus ippiTransformLuma8x8Fwd_H264_16s_C1I(Ipp16s* pSrcDst);
```

## Parameters

*pSrcDst* Pointer to a 8x8 luma block - source and destination array of size 64.

## Description

The function `ippiTransformLuma8x8Fwd_H264_16s_C1I` is declared in the `ippvc.h` file. This function performs forward transform for a 8x8 luma block without final normalization (see `normAdjust8x8` matrix described in [ITUH264]):

## Return Values

`ippStsNoErr` Indicates no error.  
`ippStsNullPtrErr` Indicates an error condition if the specified pointer is NULL.

## QuantLuma8x8\_H264

*Performs quantization for 8X8 luma block coefficients including 8X8 transform normalization.*

## Syntax

```
ippStatus ippiQuantLuma8x8_H264_16s_C1(const Ipp16s* pSrc, Ipp16s* pDst, int Qp6, int Intra, const Ipp16s* pScanMatrix, const Ipp16s* pScaleLevels, int* pNumLevels, int* pLastCoeff);
```

## Parameters

*pSrc* Pointer to source luma block coefficients - array of size 64.  
*pDst* Pointer to the destination quantized block - array of size 64.  
*QP6* Quantization parameter divided by 6.  
*Intra* Flag that is equal to 1 if the slice is intra and 0 otherwise.  
*pScanMatrix* Pointer to a scan matrix for the coefficients in the block (array of size 64).  
*pScaleLevels* Pointer to a scale level matrix.  
*pNumLevels* Pointer to a value that contains:

- a negative value of a number of non-zero elements in block after quantization (when the first quantized element in block is not equal to zero),
- a number of non-zero elements in block after quantization (when the first quantized element in block is equal to zero).

*pLastCoeff* Position of the last (in order of *pScanMatrix*) non-zero coefficient in block after quantization. This value is calculated by the function.

## Description

The function `ippiQuantLuma8x8_H264_16s_C1` is declared in the `ippvc.h` file. This function performs quantization of the coefficients of a luma block after 8x8 transform including transform normalization (see `normAdjust8x8` matrix described in [ITUH264]).

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if at least one of the specified pointers is NULL.
<code>ippStsQPErr</code>	Indicates an error if $Qp6 > 8$ or $Qp6 < 0$ .

## GenScaleLevel8x8\_H264

*Generates ScaleLevel matrices for forward and inverse quantization including normalization for 8X8 forward and inverse transform.*

---

## Syntax

```
ippStatus ippiGenScaleLevel8x8_H264_8u16s_D2(const Ipp8u* pSrcInvScaleMatrix, int SrcStep,
Ipp16s* pDstInvScaleMatrix, Ipp16s* pDstScaleMatrix, int QpRem);
```

## Parameters

<code>pSrcInvScaleMatrix</code>	Pointer to the source inverse scaling matrix for 8x8 transform.
<code>SrcStep</code>	Step of <code>pSrcInvScaleMatrix</code> , in bytes.
<code>pDstInvScaleMatrix</code>	Pointer to the destination inverse scaling matrix - array of size 64.
<code>pDstScaleMatrix</code>	Pointer to the destination forward scaling matrix - array of size 64.
<code>QpRem</code>	Reminder from integer division of the quantization parameter by 6.

## Description

The function `ippiGenScaleLevel8x8_H264_8u16s_D2` is declared in the `ippvc.h` file. This function generates scaling matrices for forward and inverse quantization, taking into account normalization for forward and inverse 8x8 transform accordingly as defined.




---

**NOTE.** According to the [ITUH264] standard an original inverse scaling matrix `pSrcIncScaleMatrix` cannot contain zeroes.

---

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if at least one of the specified pointers is NULL.
<code>ippStsQPErr</code>	Indicates an error if $QpRem > 5$ or $QpRem < 0$ .

## CAVLC Functions

These functions calculate block characteristics for CAVLC encoding:

- number of `trailing_ones` transform coefficient levels
- code that describes signs of `trailing_ones`
- number of non-zero coefficients in block
- number of zero coefficients in block
- array of levels
- array of runs.

`EncodeChromaDcCoeffsCAVLC_H264` is used for a 2x2 DC chroma block. For other block types `EncodeCoeffsCAVLC_H264` is used.

## EncodeCoeffsCAVLC\_H264

*Calculates characteristics of 4X4 block for CAVLC encoding.*

### Syntax

```
IppStatus ippiEncodeCoeffsCAVLC_H264_16s(const Ipp16s* pSrc, Ipp8u AC, const Ipp32u*
pScanMatrix, Ipp8u Count, Ipp8u* Trailing_One, Ipp8u* Trailing_One_Signs, Ipp8u* NumOutCoeffs,
Ipp8u* TotalZeros, Ipp16s* pLevels, Ipp8u* pRuns);
```

### Parameters

<i>pSrc</i>	Pointer to 4x4 block - array of size 16.
<i>AC</i>	Flag that is equal to zero in the cases when zero coefficient should be encoded, and is equal to one otherwise.
<i>pScanMatrix</i>	Scan matrix for the coefficients in block (array of size 16).
<i>Count</i>	Position of the last non-zero block coefficient in the order of scan matrix. It should be in range <code>[AC; 15]</code> .
<i>Trailing_One</i>	The number of "trailing ones" transform coefficient levels in a range <code>[0;3]</code> . This value is calculated by the function.
<i>Trailing_One_Signs</i>	Code that describes signs of "trailing ones". ( <i>Trailing_One</i> - 1 - <i>i</i> )-bit in this code corresponds to a sign of <i>i</i> -"trailing one" in the current block. In this code 1 indicates negative value, 0 - positive value. This value is calculated by the function.
<i>NumOutCoeffs</i>	The number of non-zero coefficients in block (including "trailing ones"). This value is calculated by the function.
<i>TotalZeros</i>	The number of zero coefficients in block (except "trailing zeros"). This value is calculated by the function.
<i>pLevels</i>	Pointer to an array of size 16 that contains non-zero quantized coefficients of the current block (except "trailing ones") in reverse scan matrix order. Elements of this array are calculated by the function.

*pRuns*

Pointer to an array of size 16 that contains runs before non-zero quantized coefficients (including “trailing ones”) of the current block in reverse scan matrix order (except run before the first non-zero coefficient in block, which can be calculated using *TotalZeros*). Elements of this array are calculated by the function.

### Description

The function `ippiEncodeCoeffsCAVLC_H264_16s` is declared in the `ippvc.h` file. This function calculates some characteristics (*Trailing\_One*, *Trailing\_One\_Signs*, *NumOutCoeffs*, *TotalZeros*, *pLevels*, *pRuns*) of a 4x4 block for CAVLC encoding. See [Figure "EncodeCoeffsCAVLC\\_H264 Operation"](#) for an example of the function operation:

### EncodeCoeffsCAVLC\_H264 Operation

---

Intra 16x16 AC Block

0	0	-3	6
0	0	5	0
-1	0	-1	1
0	1	0	0

Scan Matrix

0	1	4	8
5	2	3	6
9	12	13	10
7	11	14	15

---

*Count* = 13.

### Arrangement of Block Coefficients After Scanning

---

coef	0	0	5	-3	0	0	0	6	-1	1	1	0	-1	0	0
pos	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

---



*Trailing\_One* = 3 (-1, 1, 1) *Trailing\_One\_Signs* = 00000100 *NumOutCoeffs* = 7 *TotalZeros* = 6

### Array of Runs Before Non-Zero Coefficients (Except First) in Reverse Order

pRuns	1	0	0	0	3	0
coefficient	-1	1	1	-1	6	-3

### Array of Non-Zero Coefficients (Except "Trailing Ones") in Reverse Order

pLevel	-1	6	-3	5
--------	----	---	----	---

This function is used in the H.264 encoder included into Intel IPP Samples. See [introduction](#) to the H.264 section.

#### Optimization Notice

Intel's compilers may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include SSE2, SSE3, and SSSE3 instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on microprocessors not manufactured by Intel. Microprocessor-dependent optimizations in this product are intended for use with Intel microprocessors. Certain optimizations not specific to Intel microarchitecture are reserved for Intel microprocessors. Please refer to the applicable product User and Reference Guides for more information regarding the specific instruction sets covered by this notice.

Notice revision #20110804

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if at least one of the specified pointers is NULL.
<code>ippStsOutOfRangeErr</code>	Indicates an error when <i>Count</i> is out of range [ <i>AC</i> ; 15].

## EncodeChromaDcCoeffsCAVLC\_H264

*Calculates characteristics of 2X2 chroma DC block for CAVLC encoding.*

### Syntax

```
IppStatus ippiEncodeChromaDcCoeffsCAVLC_H264_16s(const Ipp16s* pSrc, Ipp8u* Trailing_One,
Ipp8u* Trailing_One_Signs, Ipp8u* NumOutCoeffs, Ipp8u* TotalZeros, Ipp16s* pLevels, Ipp8u*
pRuns);
```

### Parameters

<i>pSrc</i>	Pointer to 2x2 block - array of size 4.
<i>Trailing_One</i>	The number of "trailing ones" transform coefficient levels in a range [0;3]. This value is calculated by the function.
<i>Trailing_One_Signs</i>	Code that describes signs of "trailing ones". ( <i>Trailing_One</i> - 1 - <i>i</i> )-bit in this code corresponds to a sign of <i>i</i> -"trailing one" in the current block. In this code 1 indicates negative value, 0 - positive value. This value is calculated by the function.
<i>NumOutCoeffs</i>	The number of non-zero coefficients in block (including "trailing ones"). This value is calculated by the function.
<i>TotalZeros</i>	The number of zero coefficients in block (except "trailing zeros"). This value is calculated by the function.
<i>pLevels</i>	Pointer to an array of size 4 that contains non-zero quantized coefficients of the current block (except "trailing ones") in reverse scan matrix order. Elements of this array are calculated by the function.
<i>pRuns</i>	Pointer to an array of size 4 that contains runs before non-zero quantized coefficients (including "trailing ones") of the current block in reverse scan matrix order (except run before the first non-zero coefficient in block, which can be calculated using <i>TotalZeros</i> ). Elements of this array are calculated by the function.

### Description

The function `ippiEncodeChromaDcCoeffsCAVLC_H264_16s` is declared in the `ippvc.h` file. This function calculates some characteristics (*Trailing\_One*, *Trailing\_One\_Signs*, *NumOutCoeffs*, *TotalZeros*, *pLevels*, *pRuns*) of a 2x2 chroma DC block for CAVLC encoding.

This function is used in the H.264 encoder included into Intel IPP Samples. See [introduction](#) to the H.264 section.

#### Optimization Notice

Intel's compilers may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include SSE2, SSE3, and SSSE3 instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on microprocessors not manufactured by Intel. Microprocessor-dependent optimizations in this

**Optimization Notice**

product are intended for use with Intel microprocessors. Certain optimizations not specific to Intel microarchitecture are reserved for Intel microprocessors. Please refer to the applicable product User and Reference Guides for more information regarding the specific instruction sets covered by this notice.

Notice revision #20110804

**Return Values**

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if at least one of the specified pointers is NULL.

**CABAC Functions**

CABAC encoding starts after encoding slice headers with [CAVLC](#).

- Call [CABACInit\\_H264](#) or [CABACInitAlloc\\_H264](#) to start CABAC encoding. The `pBitStream` parameter of these functions should point to the same bit stream where slice headers were written. After calling these functions, do not write to the bit stream directly until the encoding with CABAC is finished.
- Use the low-level [CABACEncodeBin\\_H264](#) or [CABACEncodeBinBypass\\_H264](#) functions for encoding all data elements but residual blocks.
- Use the high-level [CABACEncodeResidualBlock\\_H264](#) function for encoding residual blocks to minimize overhead on a function call.
- Use [CABACTerminateSlice\\_H264](#) to finish encoding of the slice. After calling this function, the bit stream contains all the CABAC-encoded data.
- If any function fails with the `ippStsH264BufferFullErr` error, allocate a larger piece of memory for the bit stream, copy all the content of a bit stream to the place, call [CABACSetStream\\_H264](#), and then call the failed function again.
- The [CABACGetStreamSize\\_H264](#) function provides the number of bits that CABAC produces for this bit stream (for example, for the purpose of rate-control), however it is not guaranteed that these bits are written to the bit stream until [CABACTerminateSlice\\_H264](#) is called.
- The [CABACGetContexts\\_H264](#) function retrieves a specified range of CABAC contexts. The full range contains 460 contexts (see Table 9-11 of H.264 standard ([[ITUH264](#)])). You can use a retrieved context to estimate the number of bits needed for coding a single macroblock by a specific coding algorithm. This function does not determine the estimating method and does not specify the coding algorithm.

**CABACGetSize\_H264**

*Returns the size required for the CABAC state.*

**Syntax**

```
IppStatus ippiCABACGetSize_H264(Ipp32u* pSize);
```

**Parameters**

*pSize*                      Pointer to a variable to receive size.

## Description

The function `ippiCABACGetSize_H264` is declared in the `ippvc.h` file. This function returns a size of required for the CABAC state.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when the pointer <code>pSize</code> is NULL.

## CABACGetContexts\_H264

*Copies contexts from CABAC state structure.*

---

## Syntax

```
ippStatus ippiCABACGetContexts_H264(const IppvcCABACState* pCabacState, Ipp32u offset,
Ipp8u* pContexts, Ipp32u nContexts);
```

## Parameters

<code>pCabacState</code>	Pointer to CABAC state initialized with <code>ippiCABACInit_H264</code> or <code>ippiCABACInitAlloc_H264</code> .
<code>offset</code>	Offset from the initial element of the contexts array inside the <code>pCABACState</code> structure.
<code>pContexts</code>	Pointer to the destination array.
<code>nContexts</code>	Number of the contexts to copy from the source to the destination array.

## Description

The function `ippiCABACGetContexts_H264` is declared in the `ippvc.h` file. This function copies `nContexts` contexts, starting from the `offset` number of contexts, from the `IppvcCABACState` structure to the destination array `pContexts`.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when at least one of the specified pointers is NULL.
<code>ippStsOutOfRangeErr</code>	Indicates an error if the sum of <code>offset</code> and <code>nContexts</code> is greater than 460.

## CABACInit\_H264

*Initializes CABAC state and aligns bitstream to byte boundary by writing 1s.*

---

### Syntax

```
IppStatus ippICABACInit_H264(IppvcCABACState* pCabacState, Ipp8u* pBitStream, Ipp32u
nBitStreamOffsetBits, Ipp32u nBitStreamSize, Ipp32s SliceQPy, Ipp32s cabacInitIdc);
```

### Parameters

<i>pCabacState</i>	Pointer to CABAC state to be initialized. Size of this state can be retrieved by <code>ippICABACGetSize_H264</code> function.
<i>pBitStream</i>	Pointer to bitstream to write to. Must be aligned by 4.
<i>nBitStreamOffsetBits</i>	Number of bits from <i>pBitStream</i> to the first unwritten bit in a bitstream.
<i>nBitStreamSize</i>	Size of the allocated memory for the bitstream in bytes.
<i>SliceQPy</i>	<i>SliceQPy</i> parameter. See equation 7-28 from H.264 standard <a href="#">[[ITUH264_07]</a> . This parameter will be clipped to range [1; 51] (see clause 9.3.1.1 of the above standard).
<i>cabacInitIdc</i>	Index (in range [0; 2]) for determining the initialisation table used in the initialisation process for context variables (for inter slices) or -1 (for intra slices). See variable <code>cabac_init_idc</code> from the H.264 standard.

### Description

The function `ippICABACInit_H264` is declared in the `ippvc.h` file. This function initializes CABAC state and aligns the bitstream to byte boundary by writing 1s.

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when any of the pointers is NULL.
<code>ippStsSizeErr</code>	Indicates no error if <i>nBitStreamSize</i> is 0.
<code>ippStsOutOfRangeErr</code>	Indicates an error when the <code>cabac_init_idc</code> parameter is out of range [-1; 2].
<code>ippStsMisalignedBuf</code>	Indicates no error if <i>pBitStream</i> is not 4-byte aligned.
<code>ippStsH264BufferFullErr</code>	Not enough free space in the bitstream. This error code can be returned if less than 8 free bytes remain in the bitstream.

## CABACInitAlloc\_H264

*Allocates and initializes CABAC state and aligns bitstream to byte boundary by writing 1s.*

---

### Syntax

```
IppStatus ippicABACInitAlloc_H264(IppvcCABACState** ppCabacState, Ipp8u* pBitStream, Ipp32u nBitStreamOffsetBits, Ipp32u nBitStreamSize, Ipp32s SliceQPy, Ipp32s cabacInitIdc);
```

### Parameters

<i>ppCabacState</i>	Pointer to CABAC state allocated by the function. On completing encoding, the CABAC state must be released by a call to <a href="#">ippiCABACFree_H264</a> . In case of an error, the pointer pointed by <i>ppCabacState</i> remains unchanged.
<i>pBitStream</i>	Pointer to bitstream to write to. Must be aligned by 4.
<i>nBitStreamOffsetBits</i>	Number of bits from <i>pBitStream</i> to the first unwritten bit in a bitstream.
<i>nBitStreamSize</i>	Size of the allocated memory for the bitstream in bytes.
<i>SliceQPy</i>	SliceQPy parameter. See equation 7-28 from H.264 standard [ITUH264_07]. This parameter will be clipped to range [1; 51] (see clause 9.3.1.1 of the above standard).
<i>cabacInitIdc</i>	Index (in range [0; 2]) for determining the initialisation table used in the initialisation process for context variables (for inter slices) or -1 (for intra slices). See variable <code>cabac_init_idc</code> from the H.264 standard.

### Description

The function `ippicABACInitAlloc_H264` is declared in the `ippvc.h` file. This function allocates and initializes CABAC state and aligns the bitstream to byte boundary by writing 1s.

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when any of the pointers is NULL.
<code>ippStsSizeErr</code>	Indicates no error if <i>nBitStreamSize</i> is 0.
<code>ippStsOutOfRangeErr</code>	Indicates an error when the <code>cabac_init_idc</code> parameter is out of range [-1; 2].
<code>ippStsNoMemErr</code>	Not enough memory to allocate space for CABAC state.
<code>ippStsMisalignedBuf</code>	Indicates no error if <i>pBitStream</i> is not 4-byte aligned.
<code>ippStsH264BufferFullErr</code>	Not enough free space in the bitstream. This error code can be returned if less than 8 free bytes remain in the bitstream.

## CABACFree\_H264

*Releases CABAC state.*

---

### Syntax

```
IppStatus ippicABACFree_H264(IppvcCABACState* pCabacState);
```

### Parameters

*pCabacState* Pointer to CABAC state allocated by the [ippicABACInitAlloc\\_H264](#) function.

### Description

The function `ippicABACFree_H264` is declared in the `ippvc.h` file. This function releases CABAC state allocated by the [ippicABACInitAlloc\\_H264](#) function .

### Return Values

`ippStsNoErr` Indicates no error.  
`ippStsNullPtrErr` Indicates an error when the pointer *pSize* is NULL.

## CABACSetStream\_H264

*Sets new pointer for output stream.*

---

### Syntax

```
IppStatus ippicABACSetStream_H264(Ipp8u* pBitStream, Ipp32u nBitStreamSize, IppvcCABACState* pCabacState);
```

### Parameters

*pBitStream* Pointer to a new location of the bitstream. This pointer must be aligned by 4.  
*nBitStreamSize* Size of the allocated memory for the bitstream in bytes; should exceed the previous size of the stream.  
*pCabacState* Pointer to CABAC state initialized with [ippicABACInit\\_H264](#) or [ippicABACInitAlloc\\_H264](#).

### Description

The function `ippicABACSetStream_H264` is declared in the `ippvc.h` file. This function sets a new pointer for the output stream. Before calling this function, copy the content of the bitstream to the new location.

### Return Values

`ippStsNoErr` Indicates no error.  
`ippStsNullPtrErr` Indicates an error when any of the pointers is NULL.

<code>ippStsSizeErr</code>	Indicates no error if <i>nBitStreamSize</i> does not exceed the previous size of the stream.
<code>ippStsMisalignedBuf</code>	Indicates no error if <i>pBitStream</i> is not 4-byte aligned.
<code>ippStsH264BufferFullErr</code>	Not enough free space in the bitstream. This error code can be returned if less than 8 free bytes remain in the bitstream.

## CABACGetStreamSize\_H264

*Returns number of bits in bitstream.*

---

### Syntax

```
IppStatus ippICABACGetStreamSize_H264(Ipp32u* pBitStreamBits, IppvcCABACState* pCabacState);
```

### Parameters

<i>pBitStreamBits</i>	Pointer to a variable to receive the number of bits written to the stream by CABAC.
<i>pCabacState</i>	Pointer to CABAC state initialized with <a href="#">ippiCABACInit_H264</a> or <a href="#">ippiCABACInitAlloc_H264</a> .

### Description

The function `ippiCABACGetStreamSize_H264` is declared in the `ippvc.h` file. This function returns the number of bits in the bitstream.

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when any of the pointers is NULL.

## CABACEncodeBin\_H264

*Encodes one bin with CABAC and writes it to the stream.*

---

### Syntax

```
IppStatus ippICABACEncodeBin_H264(Ipp32u ctxIdx, Ipp32u code, IppvcCABACState* pCabacState);
```

### Parameters

<i>ctxIdx</i>	Index of CABAC context to encode this bin; must be in range 0 to 459. See clause 9.3.3.1 of H.264 standard [ <a href="#">ITUH264_07</a> ].
<i>code</i>	Value of bin to be encoded; must be either 0 or 1.
<i>pCabacState</i>	Pointer to CABAC state initialized with <a href="#">ippiCABACInit_H264</a> or <a href="#">ippiCABACInitAlloc_H264</a> .



## Description

The function `ippiCABACEncodeBin_H264` is declared in the `ippvc.h` file. This function encodes one bin with CABAC and writes it to the stream.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when <code>pCabacState</code> is NULL.
<code>ippStsOutOfRangeErr</code>	The <code>ctxIdx</code> parameter is not within the range of [0; 459] or <code>code</code> is out of range [0; 1].
<code>ippStsH264BufferFullErr</code>	Not enough free space in the bitstream. This error code can be returned if less than 8 free bytes remain in the bitstream.

## CABACEncodeBinBypass\_H264

*Encodes bin with CABAC using bypass encoding process and writes it to the stream.*

---

## Syntax

```
IppStatus ippiCABACEncodeBinBypass_H264(Ipp32u code, IppvcCABACState* pCabacState);
```

## Parameters

<code>code</code>	Value of bin to be encoded; must be either 0 or 1.
<code>pCabacState</code>	Pointer to CABAC state initialized with <code>ippiCABACInit_H264</code> or <code>ippiCABACInitAlloc_H264</code> .

## Description

The function `ippiCABACEncodeBinBypass_H264` is declared in the `ippvc.h` file. This function encodes CABAC using bypass encoding process and writes it to the stream.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when <code>pCabacState</code> is NULL.
<code>ippStsOutOfRangeErr</code>	The <code>code</code> parameter is out of range [0; 1].
<code>ippStsH264BufferFullErr</code>	Not enough free space in the bitstream. This error code can be returned if less than 8 free bytes remain in the bitstream.

## CABACTerminateSlice\_H264

*Terminates encoding slice with CABAC.*

---

## Syntax

```
IppStatus ippiCABACTerminateSlice_H264(Ipp32u* pBitStreamBytes, IppvcCABACState* pCabacState);
```

## Parameters

<i>pBitStreamBytes</i>	Pointer to a variable to receive the number of bytes in a bitstream.
<i>pCabacState</i>	Pointer to CABAC state initialized with <a href="#">ippiCABACInit_H264</a> or <a href="#">ippiCABACInitAlloc_H264</a> .

## Description

The function `ippiCABACTerminateSlice_H264` is declared in the `ippvc.h` file. This function terminates encoding slice with CABAC.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when any of the pointers is NULL.
<code>ippStsH264BufferFullErr</code>	Not enough free space in the bitstream. This error code can be returned if less than 8 free bytes remain in the bitstream.

## CABACEncodeResidualBlock\_H264

*Encodes block of residual coefficients with CABAC.*

---

## Syntax

```
ippStatus ippiCABACEncodeResidualBlock_H264_16s(const Ipp16* pResidualCoeffs, Ipp32u
nLastNonZeroCoeff, Ipp32u ctxBloackCat, Ipp32u log2NumC8x8, Ipp32u bFrameBlock,
IppvcCABACState* pCabacState);
```

```
ippStatus ippiCABACEncodeResidualBlock_H264_32s(const Ipp32* pResidualCoeffs, Ipp32u
nLastNonZeroCoeff, Ipp32u ctxBloackCat, Ipp32u log2NumC8x8, Ipp32u bFrameBlock,
IppvcCABACState* pCabacState);
```

## Parameters

<i>pResidualCoeffs</i>	Pointer to an array of residual coefficients to encode. Size of an array must correspond <i>ctxBloackCat</i> and <i>NumC8x8</i> parameters (see Table 9-42 of <a href="#">ITUH264_07</a> ).
<i>nLastNonZeroCoeff</i>	Index of the last non-zero coefficient in the array, pointed by <i>pResidualCoeffs</i> .
<i>ctxBloackCat</i>	Variable <i>ctxBloackCat</i> from the H.264 standard (for details, see Table 9-42 of <a href="#">ITUH264_07</a> ).
<i>log2NumC8x8</i>	For <i>ctxBloackCat</i> = 3, base 2 logarithm of <i>NumC8x8</i> variable from H.264 standard; 0 for other values of <i>ctxBloackCat</i> .
<i>bFrameBlock</i>	Boolean value that specifies whether the frame ( <i>bFrameBlock</i> = 1) or field ( <i>bFrameBlock</i> = 0) is being encoded.
<i>pCabacState</i>	Pointer to CABAC state initialized with <code>ippiCABACInit_H264</code> or <code>ippiCABACInitAlloc_H264</code> function.

## Description

The function `ippiCABACEncodeResidualBlock_H264_16s` and `ippiCABACEncodeResidualBlock_H264_32s` are declared in the `ippvc.h` file. These functions encode a block of residual coefficients with CABAC.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when any of the pointers is NULL.
<code>ippStsOutOfRangeErr</code>	<code>ctxBlockCat</code> is not in range [0; 5] or <code>bFrameBlock</code> is not in range [0; 1].
<code>ippStsBadArgErr</code>	<code>log2NumC8x8</code> is not zero, while <code>ctxBlockCat</code> is not 3.
<code>ippStsH264BufferFullErr</code>	Not enough free space in the bitstream. This error code can be returned if less than 8 free bytes remain in the bitstream.

## Inverse Quantization and Transform

### QuantLuma8x8Inv\_H264

*Performs inverse quantization for 8X8 luma block coefficients including normalization of the subsequent inverse 8X8 transform.*

## Syntax

```
ippStatus ippiQuantLuma8x8Inv_H264_16s_C1I(Ipp16s* pSrcDst, int Qp6, const Ipp16s* pInvLevelScale);
```

## Parameters

<code>pSrcDst</code>	Pointer to luma block coefficients - source and destination array of size 64.
<code>Qp6</code>	Quantization parameter divided by 6.
<code>pInvLevelScale</code>	Pointer to an inverse scale level matrix.

## Description

The function `ippiQuantLuma8x8Inv_H264_16s_C1I` is declared in the `ippvc.h` file. This function performs inverse quantization and normalization for 8x8 inverse transform of luma block coefficients according to 8.5.13 of [ITUH264].

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if at least one of the specified pointers is NULL.
<code>ippStsQPErr</code>	Indicates an error condition if <code>Qp6</code> is less than 0 or greater than 8.

## TransformLuma8x8InvAddPred\_H264

*Performs inverse 8X8 transform of 8X8 luma block coefficients with subsequent intra prediction or motion compensation.*

---

### Syntax

```
IppStatus ippiTransformLuma8x8InvAddPred_H264_16s8u_C1R(const Ipp8u* pPred, int PredStep,
Ipp16s* pSrc, Ipp8u* pDst, int DstStep);
```

### Parameters

<i>pPred</i>	Pointer to a reference 8x8 block, which is used either for intra prediction or motion compensation.
<i>PredStep</i>	Distance between starts of the consecutive lines in the reference frame.
<i>pSrc</i>	Pointer to 8x8 luma block coefficients - source and buffer array of size 64.
<i>pDst</i>	Pointer to the destination 8x8 block.
<i>DstStep</i>	Distance between starts of the consecutive lines in the destination frame.

### Description

The function `ippiTransformLuma8x8InvAddPred_H264_16s8u_C1R` is declared in the `ippvc.h` file. This function performs inverse transform of a 8x8 luma block described in 8.5.13 of [ITUH264], with subsequent intra prediction or motion compensation. Input coefficients are assumed to be inverse quantized and normalized. The *pSrc* array is used as a buffer during computations.

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if at least one of the specified pointers is NULL.

## AVS

This section describes video coding functions for encoding, decoding, processing and presenting of digital audio-video in accordance with AVS ([AVS]) standard. These functions apply to high and standard resolution of digital broadcast, laser digital media storage, broadband network streaming and multimedia communication etc.

The following enumeration is used to indicate prediction modes of the Intra\_8x8 prediction process for luma samples (9.8.2 of [AVS]):

```
typedef enum {
    IPP_8x8_VERT      = 0,
    IPP_8x8_HOR       = 1,
    IPP_8x8_DC        = 2,
    IPP_8x8_DIAG_DL   = 3,
    IPP_8x8_DIAG_DR   = 4,
} IppIntra8x8PredMode_AVS;
```

The table below shows the correspondence between prediction modes and constants of `enum IppIntra8x8PredMode_AVS`.

Name of Constant	Prediction Mode	Chapter in AVS
IPP_8x8_VERT	Intra_8x8_Vertical	9.8
IPP_8x8_HOR	Intra_8x8_Horizontal	9.8
IPP_8x8_DC	Intra_8x8_DC	9.8
IPP_8x8_DIAG_DL	Intra_8x8_Diagonal_Down_Left	9.8
IPP_8x8_DIAG_DR	Intra_8x8_Diagonal_Down_Right	9.8

The following enumeration is used to indicate prediction modes of the intra prediction process for chroma samples (9.8.3 of AVS):

```
typedef enum {
    IPP_CHROMA_DC      = 0,
    IPP_CHROMA_HOR     = 1,
    IPP_CHROMA_VERT    = 2,
    IPP_CHROMA_PLANE   = 3,
} IppIntraChromaPredMode_AVS;
```

The table below shows the correspondence between prediction modes and constants of `enum IppIntraChromaPredMode_AVS`.

Name of Constant	Prediction Mode	Chapter in AVS
IPP_CHROMA_DC	Intra_Chroma_DC	9.8
IPP_CHROMA_HOR	Intra_Chroma_Horizontal	9.8
IPP_CHROMA_VERT	Intra_Chroma_Vertical	9.8
IPP_CHROMA_PLANE	Intra_Chroma_Plane	9.8

## AVS Decoder Functions

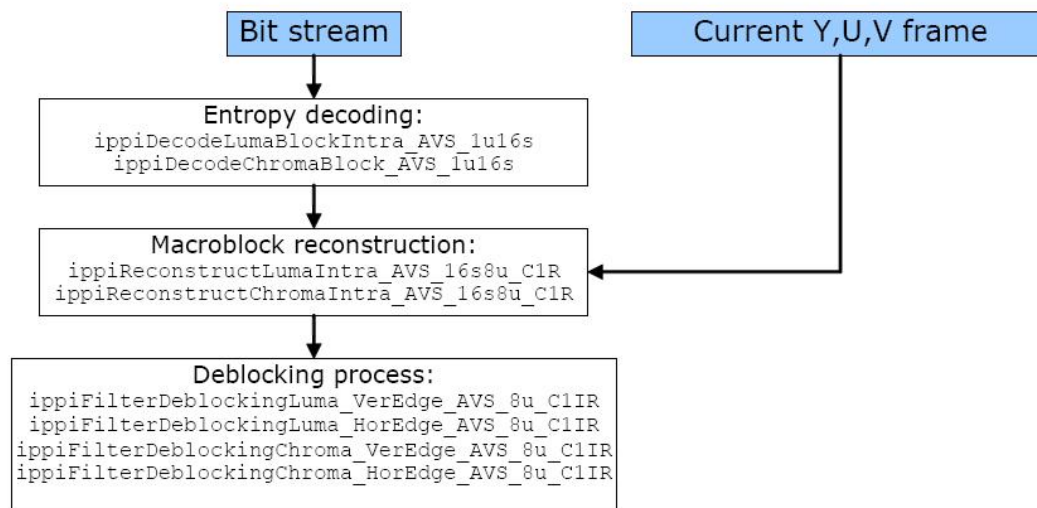
Function Short Name	Description
<b>Entropy Decoding</b>	
<a href="#">DecodeLumaBlockIntra_AVS</a>	Decodes luminance coefficients of an intra block.
<a href="#">DecodeLumaBlockInter_AVS</a>	Decodes luminance coefficients of an inter block.
<a href="#">DecodeChromaBlock_AVS</a>	Decodes chrominance coefficients of a block.
<b>Inter Prediction</b>	
<a href="#">InterpolateLumaBlock_AVS</a>	Performs interpolation for motion estimation of the luma component.
<a href="#">WeightPrediction_AVS</a>	Applies weighting process to a predicted block.
<b>Macroblock Reconstruction</b>	
<a href="#">ReconstructLumaIntra_AVS</a>	Reconstructs intra luma macroblock.
<a href="#">ReconstructLumaInter_AVS</a>	Reconstructs inter luma macroblock.
<a href="#">ReconstructChromaIntra_AVS</a>	Reconstructs intra chroma macroblock.

Function Short Name	Description
<a href="#">ReconstructChromaInter_AVS</a>	Reconstructs inter chroma macroblock.
<b>Deblocking Filtering</b>	
<a href="#">FilterDeblockingLuma_VerEdge_AVS</a>	Performs deblocking filtering on the vertical edges of the luma 16x16 macroblock.
<a href="#">FilterDeblockingLuma_HorEdge_AVS</a>	Performs deblocking filtering on the horizontal edges of the luma 16x16 macroblock.
<a href="#">FilterDeblockingChroma_VerEdge_AVS</a>	Performs deblocking filtering on the vertical edges of the chroma 8x8 macroblock.
<a href="#">FilterDeblockingChroma_HorEdge_AVS</a>	Performs deblocking filtering on the horizontal edges of the chroma 8x8 macroblock.
<b>AVS Encoder Functions</b>	
Function Short Name	Description
<b>Forward Transform and Quantization</b>	
<a href="#">TransformQuant8x8Fwd_AVS</a>	Performs forward transform and quantization for 8X8 residual blocks.
<b>Macroblock Disassembling</b>	
<a href="#">DisassembleLumaIntra_AVS</a>	Disassembles intra luma macroblocks.
<a href="#">DisassembleChroma420Intra_AVS</a>	Disassembles intra chroma macroblocks.

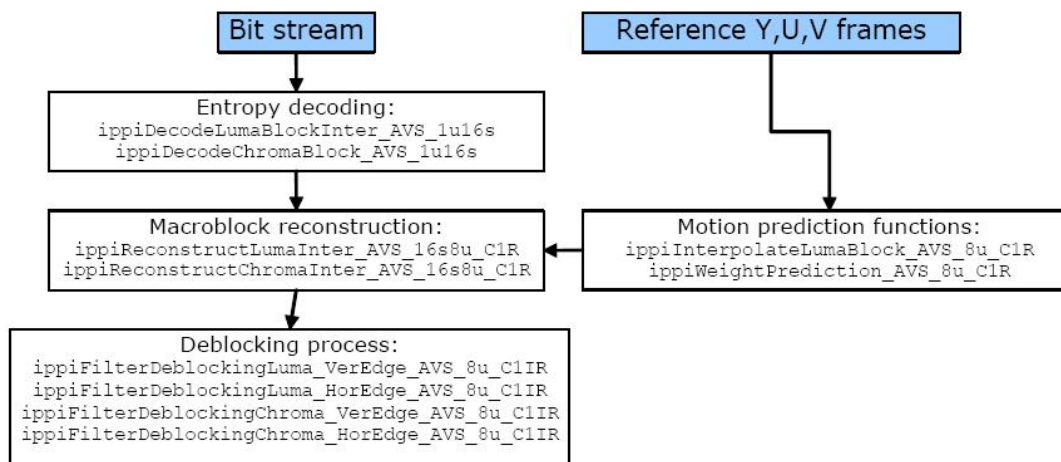
## AVS Decoder Functions

This subsection describes functions for decoding of digital audio-video in accordance with AVS ([AVS]) standard.

### AVS Intra Macroblock Decoding Process



### AVS Inter Macroblock Decoding Process



## Entropy Decoding

### DecodeLumaBlockIntra\_AVS

*Decodes luminance coefficients of an intra block.*

---

#### Syntax

```
IppStatus ippiDecodeLumaBlockIntra_AVS_1u16s(Ipp32u** ppBitStream, Ipp32s* pBitOffset,
Ipp32s* pNumCoeff, Ipp16s* pDstCoeffs, Ipp32u scanType);
```

#### Parameters

<i>ppBitStream</i>	Double pointer to the current position in the bitstream. The pointer is updated by the function.
<i>pBitOffset</i>	Pointer to offset between the bit that <i>ppBitStream</i> points to and the start of the code. The pointer is updated by the function.
<i>pNumCoeff</i>	Pointer to the output number of non-zero coefficients.
<i>pDstCoeffs</i>	Pointer to an 8x8 block of coefficients. These coefficients are calculated by the function.
<i>scanType</i>	Type of inverse scan method. 0 means the type described in Figure 9-5(a) of [AVS]. 1 means the type described in Figure 9-5(b) of [AVS].

#### Description

This function is declared in the `ippvc.h` header file. The function `ippiDecodeLumaBlockIntra_AVS_1u16s` decodes luminance coefficients of intra blocks in accordance with 9.5.1 of [AVS].

After decoding all coefficients of a block, the function performs inverse scanning, using the given type of scanning. The function returns the last coefficient index in classical scan order through the *pNumCoeff* variable as shown in Figure 9-5(a) of [AVS].

#### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when at least one input pointer is NULL.

### DecodeLumaBlockInter\_AVS

*Decodes luminance coefficients of an inter block.*

---

#### Syntax

```
IppStatus ippiDecodeLumaBlockInter_AVS_1u16s(Ipp32u** ppBitStream, Ipp32s* pBitOffset,
Ipp32s* pNumCoeff, Ipp16s* pDstCoeffs, Ipp32u scanType);
```

#### Parameters

<i>ppBitStream</i>	Double pointer to the current position in the bitstream. The pointer is updated by the function.
--------------------	--



<i>pBitOffset</i>	Pointer to offset between the bit that <i>ppBitStream</i> points to and the start of the code. The pointer is updated by the function.
<i>pNumCoeff</i>	Pointer to the output number of non-zero coefficients.
<i>pDstCoeffs</i>	Pointer to an 8x8 block of coefficients. These coefficients are calculated by the function.
<i>scanType</i>	Type of inverse scan method. 0 means the type described in Figure 9-5(a) of [AVS]. 1 means the type described in Figure 9-5(b) of [AVS].

## Description

This function is declared in the `ippvc.h` header file. The function `ippiDecodeLumaBlockInter_AVS_1u16s` decodes luminance coefficients of inter blocks in accordance with 9.5.1 of [AVS].

After decoding all coefficients of a block, the function performs inverse scanning, using the given type of scanning. The function returns the last coefficient index in classical scan order through the *pNumCoeff* variable as shown in Figure 9-5(a) of [AVS].

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when at least one input pointer is <code>NULL</code> .

## DecodeChromaBlock\_AVS

*Decodes chrominance coefficients of a block.*

## Syntax

```
ippStatus ippiDecodeChromaBlock_AVS_1u16s(Ipp32u**ppBitStream, Ipp32s* pBitOffset, Ipp32s* pNumCoeff, Ipp16s* pDstCoeffs, Ipp32u scanType);
```

## Parameters

<i>ppBitStream</i>	Double pointer to the current position in the bitstream. The pointer is updated by the function.
<i>pBitOffset</i>	Pointer to offset between the bit that <i>ppBitStream</i> points to and the start of the code. The pointer is updated by the function.
<i>pNumCoeff</i>	Pointer to the output number of non-zero coefficients.
<i>pDstCoeffs</i>	Pointer to an 8x8 block of coefficients. These coefficients are calculated by the function.
<i>scanType</i>	Type of inverse scan method. 0 means the type described in Figure 9-5(a) of [AVS]. 1 means the type described in Figure 9-5(b) of [AVS].

## Description

This function is declared in the `ippvc.h` header file. The function `ippiDecodeChromaBlock_AVS_1u16s` decodes chrominance coefficients of blocks in accordance with 9.5.1 of [AVS].

After decoding all coefficients of a block, the function performs inverse scanning, using the given type of scanning. The function returns the last coefficient index in classical scan order through the *pNumCoeff* variable as shown in Figure 9-5(a) of [AVS].

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when at least one input pointer is NULL.

### Inter Prediction

## InterpolateLumaBlock\_AVS

*Performs interpolation for motion estimation of the luma component.*

---

### Syntax

```
IppStatus ippiInterpolateLumaBlock_AVS_8u_PlR(const IppVCInterpolateBlock_8u*  
interpolateInfo);
```

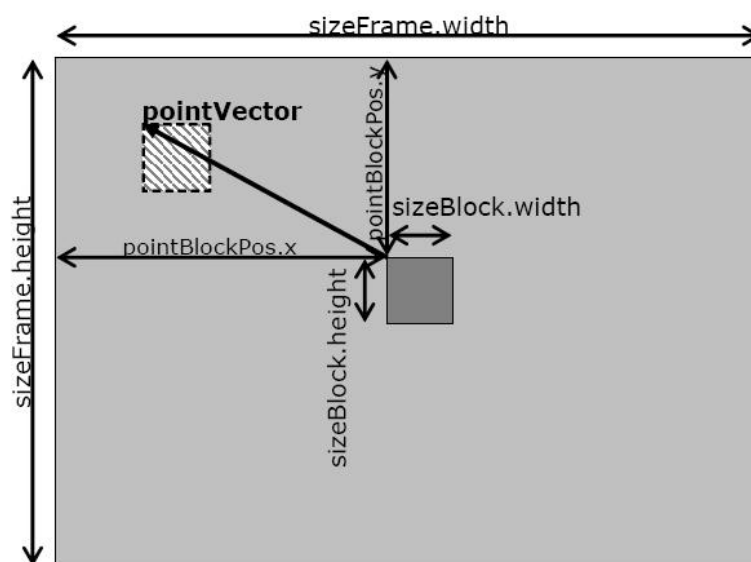
### Parameters

<i>interpolateInfo</i>	Pointer to an instance of the structure holding interpolation parameters. See <a href="#">IppVCInterpolateBlock_8u</a> structure in Inter Prediction subsection.
------------------------	--

## Description

The function `ippiInterpolateLumaBlock_AVS_8u_P1R` is declared in the `ippvc.h` file. This function performs interpolation (convolution with 6x6 kernel) for motion estimation of the luminance component in accordance with 9.9 of [AVS]. The function uses whole motion vectors, calculating fractional and integer part of vectors. The function also handles overlapping cases, when the source block pointed by the motion vector lies out of the source frame. Non-existing samples are cloned from the nearest existing samples.

## Interpolation of Luminance Component Block for Motion Estimation



## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if the specified pointer is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <code>roi.width</code> or <code>roi.height</code> take values other than {16, 8}.

## WeightPrediction\_AVS

*Applies weighting process to a predicted block.*

## Syntax

```
ippStatus ippiWeightPrediction_AVS_8u_C1R(const Ipp8u* pSrc, Ipp32s srcStep, Ipp8u* pDst,
Ipp32s dstStep, Ipp32u scale, Ipp32s shift, IppiSize sizeBlock);
```

## Parameters

<i>pSrc</i>	Pointer to the source data.
<i>srcStep</i>	Distance in bytes between starts of the consecutive lines in the source data.
<i>pDst</i>	Pointer to the destination buffer.
<i>dstStep</i>	Distance in bytes between starts of the consecutive lines in the destination buffer.
<i>scale</i>	Scale value.
<i>shift</i>	Shift.
<i>sizeBlock</i>	Size of the pointed data block. It could be 16 or 8 in each dimension.

## Description

The function `ippiWeightPrediction_AVS_8u_C1R` is declared in the `ippvc.h` file. This function applies the weighting process to predicted blocks according to 9.9.3 of [AVS] when `slice_weighting_flag` is equal to 1.

The function uses the following formula:

$$pDst[x, y] = \text{Clip1}((pSrc[x, y] \cdot scale) + 16) \gg 5 + shift$$

where  $x \in [0, roi.width-1]$ ,  $y \in [0, roi.height-1]$

$$\text{Clip1}(z) = \begin{cases} 0 & ; z < 0 \\ 255 & ; z > 255 \\ z & ; \text{otherwise} \end{cases}$$

Values of *scale* and *shift* are parsed from the bitstream.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if at least one of the specified pointers is NULL.

## Macroblock Reconstruction

### ReconstructLumaIntra\_AVS

*Reconstructs intra luma macroblock.*

#### Syntax

```
IppStatus ippiReconstructLumaIntra_AVS_16s8u_C1R(Ipp16s** ppSrcCoeff, Ipp8u* pSrcDstYPlane,
Ipp32s srcDstYStep, const IppIntra8x8PredMode_AVS*pMBIntraTypes, const Ipp32s*
pSrcNumCoeffs, Ipp32u cbp8x8, Ipp32u QP, Ipp32u edgeType);
```

#### Parameters

<i>ppSrcCoeff</i>	Pointer to the order of 8x8 blocks of residual coefficients for this macroblock, which are taken as a result of entropy decoding (8x8 luminance intra blocks, if the block is not zero-filled) in the same order as is shown in Figure 6-7 of [AVS]. The pointer is updated by the function and points to the blocks for the next macroblock.
<i>pSrcDstYPlane</i>	Pointer to the current macroblock that is reconstructed in Y-plane.
<i>srcDstYStep</i>	Y-Plane step.
<i>pMBIntraTypes</i>	Array of Intra_8x8 luma prediction modes for each subblock. <i>pMBIntraTypes[i]</i> is defined in the same order as the coded subblocks go.
<i>pSrcNumCoeffs</i>	Array of indices of the last coefficient in each subblock. <i>pSrcNumCoeffs[i]</i> is defined in the same order as the coded subblocks go.
<i>cbp8x8</i>	Coded block pattern. If <i>cbp8x8 &amp; (1&lt;&lt;(1+i))</i> is not equal to 0 ( $0 \leq i < 4$ ), <i>i</i> -th 8x8 AC luma block is not zero-filled and it exists in <i>ppSrcCoeff</i> .
<i>QP</i>	Quantization parameter ( <i>CurrentQP</i> in [AVS]). It must be within the range [0;63].
<i>edgeType</i>	Flag that specifies the availability of the macroblocks used for prediction. If the upper macroblock is not available for 8x8 intra prediction, <i>edgeType&amp;IPPVC_TOP_EDGE</i> must be non-zero. If the left macroblock is not available for 8x8 intra prediction, <i>edgeType&amp;IPPVC_LEFT_EDGE</i> must be non-zero. If the upper-left macroblock is not available for 8x8 intra prediction, <i>edgeType&amp;IPPVC_TOP_LEFT_EDGE</i> must be non-zero. If the upper-right macroblock is not available for 8x8 intra prediction, <i>edgeType&amp;IPPVC_TOP_RIGHT_EDGE</i> must be non-zero.

#### Description

The function `ippiReconstructLumaIntra_AVS_16s8u_C1R` is declared in the `ippvc.h` file. This function reconstructs intra luma macroblocks. The process for each 8x8 block in the same order as is shown in Figure 6-7 of [AVS] is described below:

- Performs scaling, integer inverse transformation, and shift for an 8x8 block in accordance with 9.5 of [AVS].
- Performs intra prediction for an 8x8 luma component in accordance with 9.4.2 of [AVS].
- Performs adding of 8x8 prediction block and 8x8 residual block in accordance with 9.10 of [AVS].

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if at least one of the specified pointers is NULL.
<code>ippStsOutOfRangeErr</code>	<i>QP</i> is less than 0 or greater than 63.

## ReconstructLumaInter\_AVS

*Reconstructs inter luma macroblock.*

### Syntax

```
IppStatus ippReconstructLumaInter_AVS_16s8u_C1R(Ipp16s** ppSrcCoeff, Ipp8u* pSrcDstYPlane,
Ipp32s srcDstYStep, const Ipp32s* pSrcNumCoeffs, Ipp32u cbp8x8, Ipp32u QP);
```

### Parameters

<i>ppSrcCoeff</i>	Pointer to the order of 8x8 blocks of residual coefficients for this macroblock, which are taken as a result of entropy decoding (8x8 luminance inter blocks, if the block is not zero-filled) in the same order as is shown in Figure 6-7 of [AVS]. The pointer is updated by the function and points to the blocks for the next macroblock.
<i>pSrcDstYPlane</i>	Pointer to the current macroblock that is reconstructed in Y-plane. This macroblock must contain inter prediction samples.
<i>srcDstYStep</i>	Y-Plane step.
<i>pSrcNumCoeffs</i>	Array of indices of the last coefficient in each subblock. <i>pSrcNumCoeffs[i]</i> is defined in the same order as the coded subblocks go.
<i>cbp8x8</i>	Coded block pattern. If <i>cbp8x8</i> & (1<<(1+i)) is not equal to 0 (0 ≤ <i>i</i> < 4), <i>i</i> -th 8x8 AC luma block is not zero-filled and it exists in <i>ppSrcCoeff</i> .
<i>QP</i>	Quantization parameter ( <i>CurrentQP</i> in [AVS]). It must be within the range [0;63].

### Description

The function `ippReconstructLumaInter_AVS_16s8u_C1R` is declared in the `ippvc.h` file. This function reconstructs inter luma macroblocks:

- Performs scaling, integer inverse transformation, and shift for each 8x8 block in the same order as is shown in Figure 6-7 of [AVS] in accordance with 9.5 of [AVS].
- Performs adding of a 16x16 inter prediction block and a 16x16 residual block in accordance with 9.10 of [AVS].

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if at least one of the specified pointers is NULL.
<code>ippStsOutOfRangeErr</code>	$QP$ is less than 0 or greater than 63.

## ReconstructChromaIntra\_AVS

*Reconstructs intra chroma macroblock.*

### Syntax

```
ippStatus ippiReconstructChromaIntra_AVS_16s8u_C1R(Ipp16s** ppSrcCoeff, Ipp8u*
pSrcDstUPlane, Ipp8u* pSrcDstVPlane, Ipp32s srcDstUVStep, const IppIntraChromaPredMode_AVS
predMode, const Ipp32s* pSrcNumCoeffs, Ipp32u cbp8x8, Ipp32u chromaQP, Ipp32u edgeType);
```

### Parameters

<code>ppSrcCoeff</code>	Pointer to the order of 8x8 blocks of residual coefficients for this macroblock, which are taken as a result of entropy decoding (8x8 chrominance intra blocks, if the block is not zero-filled) in the same order as is shown in Figure 6-7 of [AVS]. The pointer is updated by the function and points to the blocks for the next macroblock.
<code>pSrcDstUPlane</code>	Pointer to the current macroblock that is reconstructed in $U$ -plane.
<code>pSrcDstVPlane</code>	Pointer to the current macroblock that is reconstructed in $V$ -plane.
<code>srcDstUVStep</code>	Chrominance planes step.
<code>predMode</code>	Chrominance prediction mode for both subblock.
<code>pSrcNumCoeffs</code>	Array of indices of the last coefficient in each subblock. <code>pSrcNumCoeffs[i]</code> is defined in the same order as the coded subblocks go.
<code>cbp8x8</code>	Coded block pattern. If <code>cbp8x8 &amp; (1&lt;&lt;(1+i))</code> is not equal to 0 ( $0 \leq i < 2$ ), $i$ -th 8x8 AC chroma block is not zero-filled and it exists in <code>ppSrcCoeff</code> .
<code>chromaQP</code>	Quantization parameter ( $QP$ from Table 9-6 in [AVS]). It must be within the range [0;51].
<code>edgeType</code>	Flag that specifies the availability of the macroblocks used for prediction. If the upper macroblock is not available for 8x8 intra prediction, <code>edgeType&amp;IPPVC_TOP_EDGE</code> must be non-zero. If the left macroblock is not available for 8x8 intra prediction, <code>edgeType&amp;IPPVC_LEFT_EDGE</code> must be non-zero. If the upper-left macroblock is not available for 8x8 intra prediction, <code>edgeType&amp;IPPVC_TOP_LEFT_EDGE</code> must be non-zero. If the upper-right macroblock is not available for 8x8 intra prediction, <code>edgeType&amp;IPPVC_TOP_RIGHT_EDGE</code> must be non-zero.

## Description

The function `ippiReconstructChromaIntra_AVS_16s8u_C1R` is declared in the `ippvc.h` file. This function reconstructs intra chroma macroblocks. The process for each 8x8 block in the same order as is shown in Figure 6-7 of [AVS] is described below:

- Performs scaling, integer inverse transformation, and shift for an 8x8 block in accordance with 9.5 of [AVS].
- Performs intra prediction for an 8x8 chroma component in accordance with 9.4.3 of [AVS].
- Performs adding of 8x8 prediction block and 8x8 residual block in accordance with 9.10 of [AVS].

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if at least one of the specified pointers is NULL.
<code>ippStsOutOfRangeErr</code>	<code>chromaQP</code> is less than 0 or greater than 51.

## ReconstructChromaInter\_AVS

*Reconstructs inter chroma macroblock.*

---

## Syntax

```
IppStatus ippiReconstructChromaInter_AVS_16s8u_C1R(Ipp16s** ppSrcCoeff, Ipp8u*
pSrcDstUPlane, Ipp8u* pSrcDstVPlane, Ipp32s srcDstUVStep, const Ipp32s* pSrcNumCoeffs,
Ipp32u cbp8x8, Ipp32u chromaQP);
```

## Parameters

<code>ppSrcCoeff</code>	Pointer to the order of 8x8 blocks of residual coefficients for this macroblock, which are taken as a result of entropy decoding (8x8 chrominance inter blocks, if the block is not zero-filled) in the same order as is shown in Figure 6-7 of [AVS]. The pointer is updated by the function and points to the blocks for the next macroblock.
<code>pSrcDstUPlane</code>	Pointer to the current macroblock that is reconstructed in <i>U</i> -plane.
<code>pSrcDstVPlane</code>	Pointer to the current macroblock that is reconstructed in <i>V</i> -plane.
<code>srcDstUVStep</code>	Chrominance planes step.
<code>pSrcNumCoeffs</code>	Array of indices of the last coefficient in each subblock. <code>pSrcNumCoeffs[i]</code> is defined in the same order as the coded subblocks go.
<code>cbp8x8</code>	Coded block pattern. If <code>cbp8x8 &amp; (1&lt;&lt;(1+i))</code> is not equal to 0 ( $0 \leq i < 4$ ), <i>i</i> -th 8x8 AC chroma block is not zero-filled and it exists in <code>ppSrcCoeff</code> .
<code>chromaQP</code>	Quantization parameter ( <i>QP</i> from Table 9-6 in [AVS]). It must be within the range [0;51].



## Description

The function `ippiReconstructChromaInter_AVS_16s8u_C1R` is declared in the `ippvc.h` file. This function reconstructs inter chroma macroblocks:

- Performs scaling, integer inverse transformation, and shift for each 8x8 block in the same order as is shown in Figures 6-7, 6-8 of [AVS] in accordance with 9.5 of [AVS]. Order of the processed blocks depends on the passed `cbp8x8` value.
- Performs adding of an 8x8 or 8x16 inter prediction block and an 8x8 or 8x16 residual block in accordance with 9.10 of [AVS]. Size of the processed data depends on the passed `cbp8x8` value.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if at least one of the specified pointers is NULL.
<code>ippStsOutOfRangeErr</code>	<code>chromaQP</code> is less than 0 or greater than 51.

## Deblocking Filtering

### FilterDeblockingLuma\_VerEdge\_AVS

*Performs deblocking filtering on the vertical edges of the 16x16 luma macroblock.*

## Syntax

```
IppStatus ippiFilterDeblockingLuma_VerEdge_AVS_8u_C1IR(IppiFilterDeblock_8u* pDeblockInfo);
```

## Parameters

`pDeblockInfo`                      Pointer to deblocking parameters

## Description

The function `ippiFilterDeblockingLuma_VerEdge_AVS_8u_C1IR` is declared in the `ippvc.h` file. This function performs Deblocking Filtering on the vertical edges of the 16x16 luma macroblock in accordance with 9.11 of [AVS].

`IppiFilterDeblock_8u` structure contains the following fields:

<code>pSrcDstPlane</code>	Pointer to the initial and resultant coefficients.
<code>srcDstStep</code>	Distance between starts of the consecutive lines in the array.
<code>pAlpha</code>	Array of size 2 of Alpha Thresholds (values for external and internal vertical edge).
<code>pBeta</code>	Array of size 2 of Beta Thresholds (values for external and internal vertical edge).
<code>pThresholds</code>	Array of size 2 of Thresholds (filter reduction parameter <i>C</i> ) (values for external and internal vertical edge).

*pBs* Array of size 4 of BS parameters (values for the left edge of each 8x8 block).

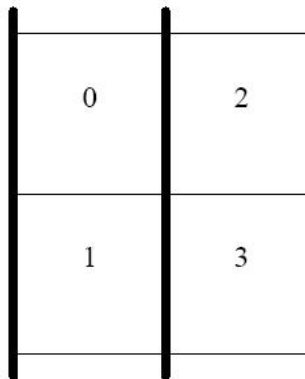
The function uses arrays *pAlpha*, *pBeta*, *pBs*, *pThresholds* as input arguments, where *pAlpha*[0], *pBeta*[0], *pThresholds* [0] are values for the external vertical edge, and *pAlpha*[1], *pBeta* [1], *pThresholds*[1] are values for the internal vertical edge. See [Figure 1](#) for the arrangement of *pBs* array elements.

Values of the arrays are calculated as follows:

- *pBs* values are calculated as per 9.11.1 of [AVS] and may take the following values: 0 - if no edge is filtered; 1 - if filtering is weak; 2 - if filtering is strong.
- *pAlpha* values are derived from Table 9-9 of [AVS].
- *pBeta* values are derived from Table 9-9 of [AVS].
- *pThresholds*[i] values are derived from Table 9-10 of [AVS].

## Arrangement of *pBs* Array Elements into a Macroblock

---



## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if at least one of the specified pointers is NULL.

## FilterDeblockingLuma\_HorEdge\_AVS

*Performs deblocking filtering on the horizontal edges of the 16X16 luma macroblock.*

---

## Syntax

```
IppStatus ippifilterDeblockingLuma_HorEdge_AVS_8u_C1IR(IppiFilterDeblock_8u* pDeblockInfo);
```

## Parameters

*pDeblockInfo* Pointer to the deblocking parameters

## Description

The function `ippiFilterDeblockingLuma_VerEdge_H264_8u_C1IR` is declared in the `ippvc.h` file. This function performs Deblocking Filtering on the horizontal edges of the 16x16 luma macroblock in accordance with 9.11 of [AVS].

`IppiFilterDeblock_8u` structure contains the following fields:

<i>pSrcDstPlane</i>	Pointer to the initial and resultant coefficients.
<i>srcDstStep</i>	Distance between starts of the consecutive lines in the array.
<i>pAlpha</i>	Array of size 2 of Alpha Thresholds (values for external and internal vertical edge).
<i>pBeta</i>	Array of size 2 of Beta Thresholds (values for external and internal vertical edge).
<i>pThresholds</i>	Array of size 2 of Thresholds (filter reduction parameter <i>C</i> ) (values for external and internal vertical edge).
<i>pBs</i>	Array of size 4 of BS parameters (values for the left edge of each 8x8 block).

The function uses arrays *pAlpha*, *pBeta*, *pBs*, *pThresholds* as input arguments, where *pAlpha*[0], *pBeta*[0], *pThresholds* [0] are values for the external horizontal edge, and *pAlpha*[1], *pBeta* [1], *pThresholds*[1] are values for the internal horizontal edge. See Figure 1 for the arrangement of *pBs* array elements.

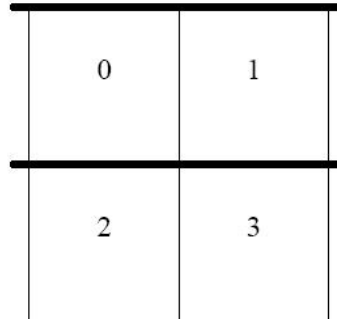
Values of the arrays are calculated as follows:

- *pBs* values are calculated as per 9.11.1 of [AVS] and may take the following values: 0 - if no edge is filtered; 1 - if filtering is weak; 2 - if filtering is strong.
- *pAlpha* values are derived from Table 9-9 of [AVS].
- *pBeta* values are derived from Table 9-9 of [AVS].

- `pThresholds[i]` values are derived from Table 9-10 of [AVS].

## Arrangement of `pBs` Array Elements into a Macroblock

---




---

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if at least one of the specified pointers is NULL.

## FilterDeblockingChroma\_VerEdge\_AVS

*Performs deblocking filtering on the vertical edges of 8X8 chroma macroblock.*

---

### Syntax

```
IppStatus ippFilterDeblockingChroma_VerEdge_AVS_8u_C1IR(IppFilterDeblock_8u*
pDeblockInfo);
```

### Parameters

`pDeblockInfo`                      Pointer to the deblocking parameters

### Description

The function `ippFilterDeblockingChroma_VerEdge_AVS_8u_C1IR` is declared in the `ippvc.h` file. This function performs Deblocking Filtering on the vertical edge of the 8x8 chroma macroblock in accordance with 9.11 of [AVS].

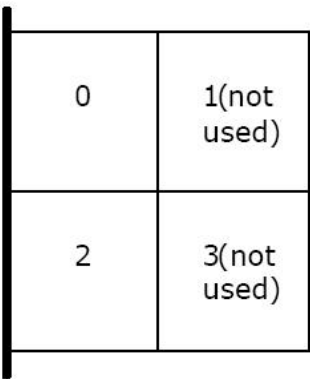
`IppFilterDeblock_8u` structure contains the following fields:

`pSrcDstPlane`                      Pointer to the initial and resultant coefficients.

<i>srcDstStep</i>	Distance between starts of the consecutive lines in the array.
<i>pAlpha</i>	Array of size 1 of Alpha Thresholds.
<i>pBeta</i>	Array of size 1 of Beta Thresholds.
<i>pThresholds</i>	Array of size 1 of Thresholds (filter reduction parameter <i>C</i> ).
<i>pBs</i>	Array of size 4 of BS parameters number (values for the left edge of an 8x4 block), but only two of the four elements are used: element 0 and element 2.

The function uses arrays *pAlpha*, *pBeta*, *pBs*, *pThresholds* as input arguments. *pAlpha*, *pBeta*, and *pBs* are the same arrays as in [FilterDeblockingLuma\\_VerEdge\\_AVS](#) function. See [Figure 1](#) for the arrangement of *pBs* array elements.

Arrangement of *pBs* Array Elements into an 8x8 Chroma Block



- Values of the arrays are calculated as follows:
- *pBs* values are calculated as per 9.11.1 of [\[AVS\]](#) and may take the following values: 0 - if no edge is filtered; 1 - if filtering is weak; 2 - if filtering is strong.
  - *pAlpha* values are derived from Table 9-9 of [\[AVS\]](#).
  - *pBeta* values are derived from Table 9-9 of [\[AVS\]](#).
  - *pThresholds*[*i*] values are derived from Table 9-10 of [\[AVS\]](#).

Return Values

<i>ippStsNoErr</i>	Indicates no error.
<i>ippStsNullPtrErr</i>	Indicates an error condition if at least one of the specified pointers is NULL.

## FilterDeblockingChroma\_HorEdge\_AVS

*Performs deblocking filtering on the horizontal edges of 8X8 chroma macroblock.*

---

### Syntax

```
IppStatus ippiFilterDeblockingChroma_HorEdge_AVS_8u_C1IR(IppiFilterDeblock_8u*  
pDeblockInfo);
```

### Parameters

*pDeblockInfo*                      Pointer to the deblocking parameters

### Description

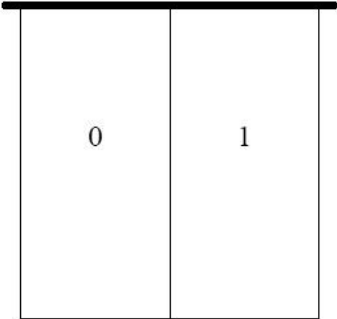
The function `ippiFilterDeblockingChroma_HorEdge_AVS_8u_C1IR` is declared in the `ippvc.h` file. This function performs Deblocking Filtering on the horizontal edge of the 8x8 chroma macroblock in accordance with 9.11 of [AVS].

`IppiFilterDeblock_8u` structure contains the following fields:

<i>pSrcDstPlane</i>	Pointer to the initial and resultant coefficients.
<i>srcDstStep</i>	Distance between starts of the consecutive lines in the array.
<i>pAlpha</i>	Array of size 1 of Alpha Thresholds.
<i>pBeta</i>	Array of size 1 of Beta Thresholds.
<i>pThresholds</i>	Array of size 1 of Thresholds (filter reduction parameter <i>C</i> ).
<i>pBs</i>	Array of size 2 of BS parameters (values for the upper edge of each 4x8 block).

The function uses arrays *pAlpha*, *pBeta*, *pBs*, *pThresholds* as input arguments. *pAlpha*, *pBeta*, and *pBs* are the same arrays as in [FilterDeblockingLuma\\_HorEdge\\_AVS](#) function. See [Figure 1](#) for the arrangement of *pBs* array elements.

Arrangement of *pBs* Array Elements into an 8x8 Chroma Block



Values of the arrays are calculated as follows:

- *pBs* values are calculated as per 9.11.1 of [\[AVS\]](#) and may take the following values: 0 - if no edge is filtered; 1 - if filtering is weak; 2 - if filtering is strong.
- *pAlpha* values are derived from Table 9-9 of [\[AVS\]](#).
- *pBeta* values are derived from Table 9-9 of [\[AVS\]](#).
- *pThresholds*[*i*] values are derived from Table 9-10 of [\[AVS\]](#).

Return Values

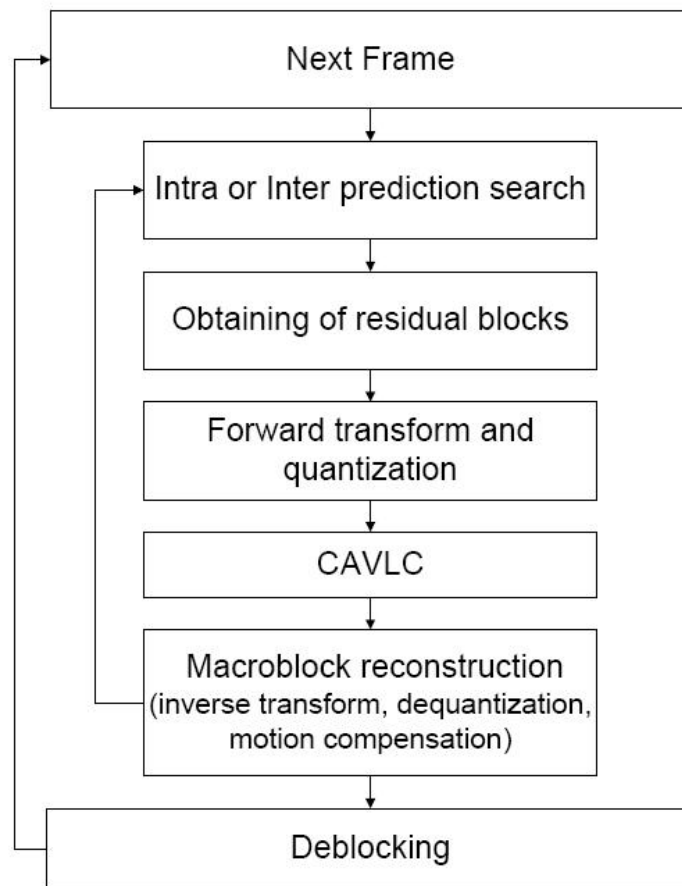
<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if at least one of the specified pointers is NULL.

## AVS Encoder Functions

This subsection describes functions for encoding of digital audio-video in accordance with the AVS ([[AVS](#)]) standard.

### AVS Encoder Structure

---



---

Other general functions used by the AVS Encoder for predicted blocks estimation and for obtaining residual blocks are [SAD Functions](#), [Sum of Differences Evaluation functions](#), and [GetDiff8x8](#) function.

The AVS Encoder uses the AVS Decoder functions for calculation of [inter](#) and [intra predicted blocks](#), [Macroblock Reconstruction](#), and [Deblocking Filtering](#). Forward transform and quantization, and CAVLC coding are performed by Encoder proper functions, which are described below.



## Forward Transform and Quantization

### TransformQuant8x8Fwd\_AVS

*Performs forward transform and quantization for 8X8 residual blocks.*

---

#### Syntax

```
IppStatus ippiTransformQuant8x8Fwd_AVS_16s_C1(const Ipp16s* pSrc, Ipp16s* pDst, Ipp32u*
pNumCoeffs, Ipp32u QP, Ipp32u roundMode);
```

#### Parameters

<i>pSrc</i>	Pointer to a 8x8 residual block - source array of size 64.
<i>pDst</i>	Pointer to a 8x8 residual block - destination array of size 64.
<i>pNumCoeffs</i>	Pointer to a value that contains a number of non-zero elements in the block after quantization. This value is calculated by the function.
<i>QP</i>	Quantization parameter ( <i>CurrentQP</i> in [AVS]) for luma or chroma, in the range [0;63] or [0, 51].
<i>roundMode</i>	Flag equal to 1 in the case of intra slice, 0 otherwise.

#### Description

The function `ippiTransformQuant8x8Fwd_AVS_16s_C1` is declared in the `ippvc.h` file. This function performs forward transform and quantization for an 8x8 residual block.

An 8x8 residual block (*pSrc*) is transformed. Quantization is performed on the transformed block. The result is saved to *pDst*.

This function is used in the AVS encoder included into Intel IPP Samples. See [introduction](#) to this chapter.

#### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if at least one of the specified pointers is NULL.
<code>ippStsOutOfRangeErr</code>	<i>QP</i> is less than 0 or greater than 63.

## Macroblock Disassembling

### DisassembleLumaIntra\_AVS

*Disassembles intra luma macroblocks.*

---

#### Syntax

```
IppStatus ippiDisassembleLumaIntra_AVS_16s8u_C1R(const Ipp8u* pSrcPlane, Ipp32s srcStep,
Ipp8u* pDstPlane, Ipp32s dstStep, Ipp16s** ppDstCoeff, const IppIntra8x8PredMode_AVS*
pPredModes, Ipp32u* pLumaCBP, Ipp32u QP, Ipp32u edgeType);
```

#### Parameters

<i>pSrcPlane</i>	Pointer to the source macroblock that is being coded in <i>Y</i> -plane.
<i>srcStep</i>	Source <i>Y</i> -plane step.
<i>pDstPlane</i>	Pointer to the current macroblock that is reconstructed in <i>Y</i> -plane.
<i>dstStep</i>	Destination <i>Y</i> -plane step.
<i>ppDstCoeff</i>	Pointer to the order of 8x8 blocks of residual coefficients for the macroblock. The pointer is updated by the function and points to free space for the blocks for the next macroblock.
<i>pPredModes</i>	Array of Intra_8x8 luma prediction modes for each subblock.
<i>pLumaCBP</i>	Pointer to the variable to store a coded block pattern. This value is calculated by the function.
<i>QP</i>	Quantization parameter ( <i>CurrentQP</i> in [AVS]), must be in the range [0;63].
<i>edgeType</i>	Flag that specifies the availability of the macroblocks used for prediction. If the upper macroblock is not available for 8x8 intra prediction, <i>edgeType&amp;IPPVC_TOP_EDGE</i> must be non-zero. If the left macroblock is not available for 8x8 intra prediction, <i>edgeType&amp;IPPVC_LEFT_EDGE</i> must be non-zero. If the upper-left macroblock is not available for 8x8Intra prediction, <i>edgeType&amp;IPPVC_TOP_LEFT_EDGE</i> must be non-zero. If the upper-right macroblock is not available for 8x8Intra prediction, <i>edgeType&amp;IPPVC_TOP_RIGHT_EDGE</i> must be non-zero.

#### Description

The function `ippiDisassembleLumaIntra_AVS_16s8u_C1R` is declared in the `ippvc.h` file. This function disassembles intra luma macroblocks. The residual coefficients are taken as a result of intra-prediction, forward transforming, and forward quantization (8x8 luminance intra blocks, if the block is not zero-filled) in the same order as is shown in Figure 6-7 of [AVS]. The process for the 8x8 blocks is described below:

- Perform intra prediction for an 8x8 luma component in accordance with 9.4.2 of [AVS].
- Perform subtraction of an 8x8 prediction block from an 8x8 source block.
- Perform scaling, integer transformation, and forward shift for an 8x8 block.

- Perform these steps for every available prediction modes for each block and save the best modes to a corresponding variable.

The function calculates the best prediction mode for each subblock, trying to save as much of visual information as possible. *pPredModes* [ *i* ] is defined in the same order as the coded subblocks go.

If *\*pLumaCBP* & (1 << (1 + *i*)) is not equal to 0 ( $0 \leq i < 4$ ), *i*-th 8x8 AC luma block is not zero-filled and it exists in *ppDstCoeff*.

## Return Values

<i>ippStsNoErr</i>	Indicates no error.
<i>ippStsNullPtrErr</i>	Indicates an error condition if at least one of the specified pointers is NULL.
<i>ippStsOutOfRangeErr</i>	<i>QP</i> is less than 0 or greater than 63.

## DisassembleChroma420Intra\_AVS

*Disassembles intra chroma macroblocks.*

### Syntax

```
IppStatus ippidisassembleChroma420Intra_AVS_16s8u_C1R(const Ipp8u* pSrcPlane[2], Ipp32s
srcStep, Ipp8u* pDstPlane[2], Ipp32s dstStep, Ipp16s** ppDstCoeff, const
IppIntraChromaPredMode_AVS* pPredModes, Ipp32u* pChromaCBP, Ipp32u QP, Ipp32u edgeType);
```

### Parameters

<i>pSrcPlane</i>	Array of pointers to the source macroblock that is coded in <i>U</i> -plane and <i>Y</i> -plane.
<i>srcStep</i>	Source chrominance plane step.
<i>pDstPlane</i>	Pointer to the current macroblock that is reconstructed in <i>U</i> -plane and <i>Y</i> -plane.
<i>dstStep</i>	Destination chrominance plane step.
<i>ppDstCoeff</i>	Double pointer to the order of 8x8 blocks of residual coefficients for this macroblock, which are taken as a result of intra-prediction, forward transforming, and forward quantization (8x8 chrominance intra blocks, if the block is not zero-filled) in the same order as is shown in Figure 6-7 of [AVS]. The pointer is updated by the function and points to free space for the blocks for the next macroblock.
<i>pPredModes</i>	Pointer to the variable to store chrominance prediction mode for both subblocks. The function calculates the best prediction mode for each subblock, trying to save as much of visual information as possible. <i>pPredModes</i> [ <i>i</i> ] is defined in the same order as the coded subblocks go.

<i>pChromaCBP</i>	Pointer to the variable to store a coded block pattern. This value is calculated by the function. If <i>*pChromaCBP</i> & (1 << (1 + <i>i</i> )) is not equal to 0 ( $0 \leq i < 4$ ), <i>i</i> -th 8x8 AC chroma block is not zero-filled and it exists in <i>ppDstCoeff</i> .
<i>QP</i>	Quantization parameter ( <i>CurrentQP</i> in [AVS]), must be within range [0;51].
<i>edgeType</i>	Flag that specifies the availability of the macroblocks used for prediction. If the upper macroblock is not available for 8x8 intra prediction, <i>edgeType</i> &IPPVC_TOP_EDGE must be non-zero. If the left macroblock is not available for 8x8 intra prediction, <i>edgeType</i> &IPPVC_LEFT_EDGE must be non-zero. If the upper-left macroblock is not available for 8x8Intra prediction, <i>edgeType</i> &IPPVC_TOP_LEFT_EDGE must be non-zero. If the upper-right macroblock is not available for 8x8Intra prediction, <i>edgeType</i> &IPPVC_TOP_RIGHT_EDGE must be non-zero.

## Description

The function `ippiDisassembleChroma420Intra_AVS_16s8u_C1R` is declared in the `ippvc.h` file. This function disassembles intra chroma macroblocks. The process for the 8x8 blocks in the same order as is shown in Figure 6-7 of [AVS] is described below:

- Perform intra prediction for an 8x8 chroma component in accordance with 9.4.2 of [AVS].
- Perform subtraction of an 8x8 prediction block from an 8x8 source block.
- Perform scaling, integer transformation, and forward shift for an 8x8 block.
- Perform these steps for every available prediction modes for each block and save the best modes to a corresponding variable.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if at least one of the specified pointers is NULL.
<code>ippStsOutOfRangeErr</code>	<i>QP</i> is less than 0 or greater than 51.

## VC-1

This section describes functions for decoding and encoding of video data in accordance with VC-1 standard ([SMPT421M]).

The following enumeration is used for deblocking and smoothing functions to indicate parts of luma blocks edges:

```
enum{
```

```
    IPPVC_EDGE_QUARTER_1          = 0x01,
    IPPVC_EDGE_QUARTER_2          = 0x02,
    IPPVC_EDGE_QUARTER_3          = 0x04,
    IPPVC_EDGE_QUARTER_4          = 0x08,
    IPPVC_EDGE_HALF_1             = IPPVC_EDGE_QUARTER_1 + IPPVC_EDGE_QUARTER_2,
    IPPVC_EDGE_HALF_2             = IPPVC_EDGE_QUARTER_3 + IPPVC_EDGE_QUARTER_4,
    IPPVC_EDGE_ALL                 = IPPVC_EDGE_HALF_1 + IPPVC_EDGE_HALF_2,
```

```
};
```

Note that the VC-1 functions with `_16u_` and `_8u_` data types use the following structures:

```
typedef struct _IppVCInterpolate_16u
```

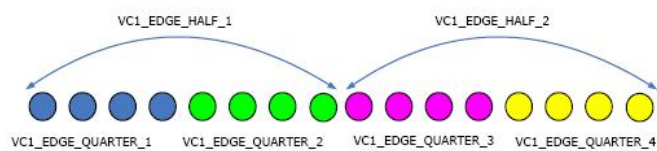
```
{  
  
    const Ipp16u* pSrc;  
    Ipp32s      srcStep;  
    Ipp16u*     pDst;  
    Ipp32s      dstStep;  
    Ipp32s      dx;  
    Ipp32s      dy;  
    IppiSize     roiSize;  
    Ipp32s      bitDepth;  
} IppVCInterpolate_16u;
```

```
typedef struct _IppVCInterpolate_8u
```

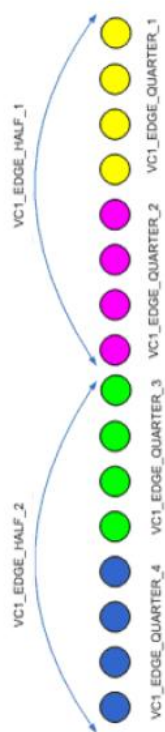
```
{  
  
    const Ipp8u* pSrc;  
    Ipp32s      srcStep;  
    Ipp8u*     pDst;  
    Ipp32s      dstStep;  
    Ipp32s      dx;  
    Ipp32s      dy;  
    IppiSize     roiSize;  
    Ipp32s      bitDepth;  
} IppVCInterpolate_8u;
```

where  $dx$ ,  $dy$  are fractional parts of the motion vector in 1/4 pel units (0, 1, 2, or 3).

### Horizontal Luma Edge

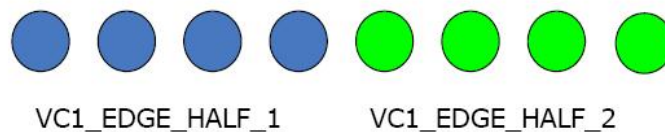


### Vertical Luma Edge



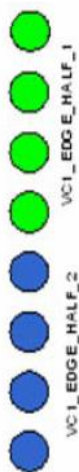
## Horizontal Chroma Edge

---



## Vertical Chroma Edge

---

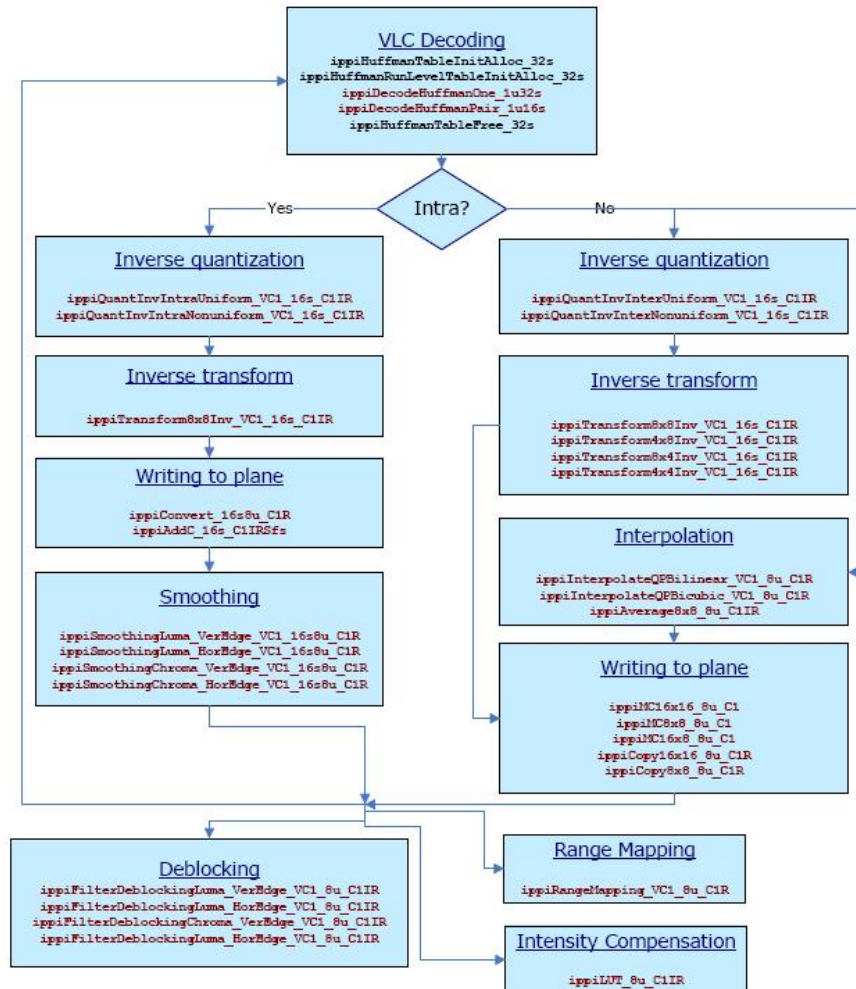




## VC-1 Decoder

See the figure below for VC-1 decoding scheme. The functions not in brown color are auxiliary and are called only once in the process of deblocking.

### VC-1 Decoding Scheme



### VC1 Video Decoder Functions

#### Function Short Name

#### Description

#### Inverse Transform

Transform8x8Inv\_VC1, Transform4x8Inv\_VC1,  
Transform8x4Inv\_VC1, Transform4x4Inv\_VC1

Perform inverse transform of blocks of respective size according to VC-1 standard.

#### Interpolation

Function Short Name	Description
<a href="#">InterpolateQPBilinear_VCl</a>	Performs bilinear quarter-pel interpolation.
<a href="#">InterpolateQPBicubic_VCl</a>	Performs bicubic quarter-pel interpolation.
<a href="#">InterpolateICBilinearBlock_VCl</a>	Performs intensity compensation and bilinear quarter-pel interpolation for motion estimation of the luma or chroma component using an entire motion vector.
<a href="#">InterpolateICBicubicBlock_VCl</a>	Performs intensity compensation and bicubic quarter-pel interpolation for motion estimation of the luma or chroma component using an entire motion vector.
<b>Smoothing</b>	
<a href="#">SmoothingLuma_VerEdge_VCl</a>	Performs smoothing filtering on the vertical edge (internal or external) of luma 16X16 macroblocks.
<a href="#">SmoothingLuma_HorEdge_VCl</a>	Performs smoothing filtering on the horizontal edge (internal or external) of luma 16X16 macroblocks.
<a href="#">SmoothingChroma_VerEdge_VCl</a>	Performs smoothing filtering on the vertical edge (internal or external) of chroma 8X8 blocks.
<a href="#">SmoothingChroma_HorEdge_VCl</a>	Performs smoothing filtering on the horizontal edge (internal or external) of chroma 8X8 blocks.
<b>Deblocking</b>	
<a href="#">FilterDeblockingLuma_VerEdge_VCl</a>	Performs deblocking filtering on the vertical edge (internal or external) of luma 16X16 macroblocks.
<a href="#">FilterDeblockingLuma_HorEdge_VCl</a>	Performs deblocking filtering on the horizontal edge (internal or external) of luma 16X16 macroblocks.
<a href="#">FilterDeblockingChroma_VerEdge_VCl</a>	Performs deblocking filtering on the vertical edge (internal or external) of chroma 8X8 macroblocks.
<a href="#">FilterDeblockingChroma_HorEdge_VCl</a>	Performs deblocking filtering on the horizontal edge (internal or external) of chroma 8X8 macroblocks.
<b>Dequantization</b>	
<a href="#">QuantInvIntraUniform_VCl</a> , <a href="#">QuantInvIntra-Nonuniform_VCl</a>	Perform uniform/nonuniform dequantization of 8X8 intra blocks.
<a href="#">QuantInvInterUniform_VCl</a> , <a href="#">QuantInvInter-Nonuniform_VCl</a>	Perform uniform/nonuniform dequantization of inter blocks.
<b>Range Reduction</b>	

Function Short Name	Description
<a href="#">RangeMapping_VC1</a>	Performs range map transformation.

## Inverse Transform

### Transform8x8Inv\_VC1, Transform4x8Inv\_VC1, Transform8x4Inv\_VC1, Transform4x4Inv\_VC1

*Perform inverse transform of blocks of respective sizes according to VC-1 standard.*

#### Syntax

```
IppStatus ippiTransform8x8Inv_VC1_16s_C1IR(Ipp16s* pSrcDst, Ipp32s srcDstStep, IppiSize sizeNZ);
```

```
IppStatus ippiTransform4x8Inv_VC1_16s_C1IR(Ipp16s* pSrcDst, Ipp32s srcDstStep, IppiSize sizeNZ);
```

```
IppStatus ippiTransform8x4Inv_VC1_16s_C1IR(Ipp16s* pSrcDst, Ipp32s srcDstStep, IppiSize sizeNZ);
```

```
IppStatus ippiTransform4x4Inv_VC1_16s_C1IR(Ipp16s* pSrcDst, Ipp32s srcDstStep, IppiSize sizeNZ);
```

```
IppStatus ippiTransform8x8Inv_VC1_16s_C1R(const Ipp16s* pSrc, Ipp16s* pDst, Ipp32s srcDstStep, IppiSize sizeNZ);
```

```
IppStatus ippiTransform4x8Inv_VC1_16s_C1R(const Ipp16s* pSrc, Ipp16s* pDst, Ipp32s srcDstStep, IppiSize sizeNZ);
```

```
IppStatus ippiTransform8x4Inv_VC1_16s_C1R(const Ipp16s* pSrc, Ipp16s* pDst, Ipp32s srcDstStep, IppiSize sizeNZ);
```

```
IppStatus ippiTransform4x4Inv_VC1_16s_C1R(const Ipp16s* pSrc, Ipp16s* pDst, Ipp32s srcDstStep, IppiSize sizeNZ);
```

#### Parameters

<i>pSrcDst</i>	Pointer to the source and destination block. All samples of the source block should be in range [-2048; 2047). After transforming all samples of the destination block are in range [-512, 511).
<i>pSrc</i>	Pointer to the source block. All samples of the source block should be in range [-2048; 2047).
<i>pDst</i>	Pointer to the destination 8x8 block. After transforming all samples of the destination block are in range [-512, 511).
<i>srcDstStep</i>	Distance between starts of the consecutive lines in the source and destination blocks.

*sizeNZ*

Size of the top left-hand subblock with non-zero coefficients. This parameter is used by the function for improving performance with fast transforming algorithms if *sizeNZ* is less than block size. If *sizeNZ* is equal to the block size then “full” transforming algorithm is used.

## Description

The functions `ippiTransform8x8Inv_Vc1_16s`, `ippiTransform4x8Inv_Vc1_16s`, `ippiTransform8x4Inv_Vc1_16s`, and `ippiTransform4x4Inv_Vc1_16s` are declared in the `ippvc.h` file. These functions perform inverse transform in accordance with Annex A: *Transform Specification* of [SMPT421M].

`ippiTransform8x8Inv_Vc1_16s_C1IR` and `ippiTransform8x8Inv_Vc1_16s_C1R` perform inverse transform of 8x8 inter block with 8x8 transformation type or 8x8 intra block.

`ippiTransform4x8Inv_Vc1_16s_C1IR` and `ippiTransform4x8Inv_Vc1_16s_C1R` perform inverse transform of 4x8 block into 8x8 inter block with 4x8 transformation type.

`ippiTransform8x4Inv_Vc1_16s_C1IR` and `ippiTransform8x4Inv_Vc1_16s_C1R` perform inverse transform of 8x4 subblock into 8x8 inter block with 8x4 transformation type.

`ippiTransform4x4Inv_Vc1_16s_C1IR` and `ippiTransform4x4Inv_Vc1_16s_C1R` perform inverse transform of 4x4 block into 8x8 inter block with 4x4 transformation type.

See below for an example of 8x4 block transform.

## 8x4 Block Inverse Transform

---

16	0	0	-3	0	0	0	0
1	1	0	1	0	0	0	0
3	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

---

In this case `pDstSizeNZ -> width` should be equal to 4, `pDstSizeNZ -> height` should be equal to 3.

## Return Values

`ippStsNoErr`

Indicates no error.

`ippStsNullPtrErr`

Indicates an error condition when at least one of the specified pointers is NULL.

## Example

The code example below shows how to use `ippiTransform8x8Inv_Vc1_16s_C1IR` function.

```
{
Ipp16s* pSrcDstData = pointer_on_source_and_destination_block;
/*
All samples of source block should be in the range [-2048; 2047)
After transforming all samples of destination block will be in the range
[-512, 511)
```

```

*/
Ipp32s srcDstStep = source_and_destination_step; /* Step of the pointer
pSrcDstData (source and destination array) in bytes */
IppiSize dstSizeNZ = block_size_non_zero;
/*
The size of top left subblock with non-zero coefficients. This parameter is used
by this function for improving performance with the help of "fast" transforming
algorithms in the case of sizeNZ less than block size. If sizeNZ is equal to
block size, then "full" transforming algorithm is used.
Example (8x4 block):
16 0 0 -3 0 0 0 0
1  1 0  1 0 0 0 0
3  0 0  1 0 0 0 0
0  0 0  0 0 0 0 0
In this case dstSizeNZ.width=4, dstSizeNZ.height=3
*/
IppStatus result;
/*
ippStsNoErr          Indicates no error
ippStsNullPtrErr     Indicates an error when pSrcDstData is NULL
*/

result = ippiTransform8x8Inv_VC1_16s_C1IR(pSrcDstData, srcDstStep, dstSizeNZ);
/*
Performs inverse transform of 8x8 inter block with 8x8 transformation type or
8x8 intra block in accordance with Annex A "Transform Specification" SMPTE 421M
*/

}

```

The code example below shows how to use ippiTransform8x8Inv\_VC1\_16s\_C1R function.

```

{
Ipp16s* pSrcData = pointer_on_source_block;
/*
All samples of source block should be in the range [-2048; 2047)
*/
Ipp16s* pDstData = pointer_on_destination_block;
/*
After transforming all samples of destination block will be in the range
[-512, 511)
*/
Ipp32s srcStep = source_step; /* Step of the pointer pSrcData (source array) in bytes*/
Ipp32s dstStep = destination_step; /* Step of the pointer pDstData (destination array)
in bytes*/
IppiSize dstSizeNZ = block_size_non_zero;
/*
The size of top left subblock with non-zero coefficients. This parameter is used
by this function for improving performance with the help of "fast" transforming
algorithms in the case of sizeNZ less than block size. If sizeNZ is equal to
block size, then "full" transforming algorithm is used.
Example (8x4 block):
16 0 0 -3 0 0 0 0
1  1 0  1 0 0 0 0
3  0 0  1 0 0 0 0
0  0 0  0 0 0 0 0
In this case dstSizeNZ.width=4, dstSizeNZ.height=3
*/

IppStatus result;
/*
ippStsNoErr          Indicates no error
ippStsNullPtrErr     Indicates an error when pSrcData or pDstData is NULL
*/
}

```

```
*/

result = ippiTransform8x8Inv_Vc1_16s_C1R( pSrcData, srcStep, pDstData, dstStep,
dstSizeNZ );
/*
Performs inverse transform of 8x8 inter block with 8x8 transformation type or
8x8 intra block in accordance with Annex A "Transform Specification" SMPTE 421M
*/

}
```

The code example below shows how to use ippiTransform4x8Inv\_Vc1\_16s\_C1IR function.

```
{
Ipp16s* pSrcDstData = pointer_on_source_and_destination_block;
/*
All samples of source block should be in the range [-2048; 2047)
After transforming all samples of destination block will be in the range
[-512, 511)
*/
Ipp32s srcDstStep = source_and_destination_step; /* Step of the pointer
pSrcDstData (source and destination array) in bytes*/
IppiSize dstSizeNZ[2] = block_size_non_zero;
/*
The size of top left subblock with non-zero coefficients. This parameter is used
by this function for improving performance with the help of "fast" transforming
algorithms in the case of sizeNZ less than block size. If sizeNZ is equal to
block size, then "full" transforming algorithm is used.
Example (8x4 block):
16 0 0 -3 0 0 0 0
1 1 0 1 0 0 0 0
3 0 0 1 0 0 0 0
0 0 0 0 0 0 0 0
In this case dstSizeNZ.width=4, dstSizeNZ.height=3
*/
IppStatus result;
/*
ippiStsNoErr          Indicates no error
ippiStsNullPtrErr     Indicates an error when pSrcDstData is NULL
*/

result = ippiTransform4x8Inv_Vc1_16s_C1IR(pSrcDstData,srcDstStep,dstSizeNZ[0]);
result = ippiTransform4x8Inv_Vc1_16s_C1IR(pSrcDstData+4,srcDstStep,
dstSizeNZ[1]);

/*
Performs inverse transform of 4x8 subblock in 8x8 inter block with 4x8
transformation type in accordance with Annex A "Transform Specification"
SMPTE 421M
*/

}
```

The code example below shows how to use ippiTransform4x8Inv\_Vc1\_16s\_C1R function.

```
{
Ipp16s* pSrcData = pointer_on_source_block;
/*
All samples of source block should be in the range [-2048; 2047)
*/
Ipp16s* pDstData = pointer_on_destination_block;
/*
After transforming all samples of destination block will be in the range [-512,
```

```

511)
*/
Ipp32s srcStep = source_step; /* Step of the pointer pSrcData (source array)
in bytes*/
Ipp32s dstStep = destination_step; /* Step of the pointer pDstData (destination
array) in bytes*/
IppiSize dstSizeNZ[2] = block_size_non_zero;
/*
The size of top left subblock with non-zero coefficients. This parameter is used
by this function for improving performance with the help of "fast" transforming
algorithms in the case of sizeNZ less than block size. If sizeNZ is equal to
block size, then "full" transforming algorithm is used.
Example (8x4 block):
16 0 0 -3 0 0 0 0
1 1 0 1 0 0 0 0
3 0 0 1 0 0 0 0
0 0 0 0 0 0 0 0
In this case dstSizeNZ.width=4, dstSizeNZ.height=3
*/
IppStatus result;
/*
ippStsNoErr          Indicates no error
ippStsNullPtrErr     Indicates an error when pSrcData or pDstData is NULL
*/

result = ippiTransform4x8Inv_VC1_16s_C1R( pSrcData, srcStep, pDstData, dstStep,
dstSizeNZ[0] );
result = ippiTransform4x8Inv_VC1_16s_C1R( pSrcData+4, srcStep, pDstData+4,
dstStep, dstSizeNZ[1] );

/*
Performs inverse transform of 4x8 subblock in 8x8 inter block with 4x8
transformation type in accordance with Annex A "Transform Specification" SMPTE
421M
*/
}

```

The code example below shows how to use ippiTransform8x4Inv\_VC1\_16s\_C1IR function.

```

{
Ipp16s* pSrcDstData = pointer_on_source_and_destination_block;
/*
All samples of source block should be in the range [-2048; 2047)
After transforming all samples of destination block will be in the range [-512,
511)
*/
Ipp32s srcDstStep = source_and_destination_step; /* Step of the pointer
pSrcDstData (source and destination array) in bytes */
IppiSize dstSizeNZ[2] = block_size_non_zero;
/*
The size of top left subblock with non-zero coefficients. This parameter is used
by this function for improving performance with the help of "fast" transforming
algorithms in the case of sizeNZ less than block size. If sizeNZ is equal to
block size, then "full" transforming algorithm is used.
Example (8x4 block):
16 0 0 -3 0 0 0 0
1 1 0 1 0 0 0 0
3 0 0 1 0 0 0 0
0 0 0 0 0 0 0 0
In this case dstSizeNZ.width=4, dstSizeNZ.height=3
*/
IppStatus result;

```

```

/*
ippStsNoErr          Indicates no error
ippStsNullPtrErr     Indicates an error when pSrcDstData is NULL
*/

result = ippiTransform8x4Inv_Vc1_16s_C1R( pSrcDstData, srcDstStep,
dstSizeNZ[0] );
result = ippiTransform8x4Inv_Vc1_16s_C1R( pSrcDstData+(srcDstStep*2),
srcDstStep, dstSizeNZ[1] );

/*
Performs inverse transform of 8x4 subblock in 8x8 inter block with 8x4
transformation type in accordance with Annex A "Transform Specification"
SMPTE 421M
*/

}

```

The code example below shows how to use `ippiTransform8x4Inv_Vc1_16s_C1R` function.

```

{
Ipp16s* pSrcData = pointer_on_source_block;
/*
All samples of source block should be in the range [-2048; 2047)
*/
Ipp16s* pDstData = pointer_on_destination_block;
/*
After transforming all samples of destination block will be in the range
[-512, 511)
*/
Ipp32s srcStep = source_step; /* Step of the pointer pSrcData (source array) in
bytes*/
Ipp32s dstStep = destination_step; /* Step of the pointer pDstData (destination
array) in bytes*/
IppiSize dstSizeNZ[2] = block_size_non_zero;
/*
The size of top left subblock with non-zero coefficients. This parameter is used
by this function for improving performance with the help of "fast" transforming
algorithms in the case of sizeNZ less than block size. If sizeNZ is equal to
block size, then "full" transforming algorithm is used.
Example (8x4 block):
16 0 0 -3 0 0 0 0
1 1 0 1 0 0 0 0
3 0 0 1 0 0 0 0
0 0 0 0 0 0 0 0
In this case dstSizeNZ.width=4, dstSizeNZ.height=3
*/

IppStatus result;
/*
ippStsNoErr          Indicates no error
ippStsNullPtrErr     Indicates an error when pSrcData or pDstData is NULL
*/

result = ippiTransform8x4Inv_Vc1_16s_C1R( pSrcData, srcStep, pDstData, dstStep,
dstSizeNZ[0] );
result = ippiTransform8x4Inv_Vc1_16s_C1R( pSrcData+(srcStep*2), srcStep,
pDstData+(dstStep*2), dstStep, dstSizeNZ[1] );

/*
Performs inverse transform of 8x4 subblock in 8x8 inter block with 8x4
transformation type in accordance with Annex A "Transform Specification"
SMPTE 421M
*/
}

```



```
*/
}
```

The code example below shows how to use `ippiTransform4x4Inv_Vc1_16s_C1IR` function.

```
{
Ipp16s* pSrcDstData = pointer_on_source_and_destination_block;
/*
All samples of source block should be in the range [-2048; 2047)
After transforming all samples of destination block will be in the range
[-512, 511)
*/
Ipp32s srcDstStep = source_and_destination_step; /* Step of the pointer
pSrcDstData (source and destination array) in bytes */
IppiSize dstSizeNZ[4] = block_size_non_zero;
/*
The size of top left subblock with non-zero coefficients. This parameter is used
by this function for improving performance with the help of "fast" transforming
algorithms in the case of sizeNZ less than block size. If sizeNZ is equal to
block size, then "full" transforming algorithm is used.
Example (8x4 block):
16 0 0 -3 0 0 0 0
1 1 0 1 0 0 0 0
3 0 0 1 0 0 0 0
0 0 0 0 0 0 0 0
In this case dstSizeNZ.width=4, dstSizeNZ.height=3
*/
IppStatus result;
/*
ippStsNoErr          Indicates no error
ippStsNullPtrErr     Indicates an error when pSrcDstData is NULL
*/

result = ippiTransform4x4Inv_Vc1_16s_C1IR( pSrcDstData, srcDstStep,
dstSizeNZ[0] );
result = ippiTransform4x4Inv_Vc1_16s_C1IR( pSrcDstData+4, srcDstStep,
dstSizeNZ[1] );
result = ippiTransform4x4Inv_Vc1_16s_C1IR( pSrcDstData+(srcDstStep*2),
srcDstStep, dstSizeNZ[2] );
result = ippiTransform4x4Inv_Vc1_16s_C1IR( pSrcDstData+(srcDstStep*2)+4,
srcDstStep, dstSizeNZ[3] );

/*
Performs inverse transform of 4x4 subblock in 8x8 inter block with 4x4
transformation type in accordance with Annex A "Transform Specification"
SMPTE 421M
*/
}
```

The code example below shows how to use `ippiTransform4x4Inv_Vc1_16s_C1R` function.

```
{
Ipp16s* pSrcData = pointer_on_source_block;
/*
All samples of source block should be in the range [-2048; 2047)
*/
Ipp16s* pDstData = pointer_on_destination_block;
/*
After transforming all samples of destination block will be in the range
[-512, 511)
*/
}
```

```

Ipp32s srcStep = source_step; /* Step of the pointer pSrcData (source array)
in bytes*/
Ipp32s dstStep = destination_step; /* Step of the pointer pDstData (destination
array) in bytes*/
IppiSize dstSizeNZ[4] = block_size_non_zero;
/*
The size of top left subblock with non-zero coefficients. This parameter is used
by this function for improving performance with the help of "fast" transforming
algorithms in the case of sizeNZ less than block size. If sizeNZ is equal to
block size, then "full" transforming algorithm is used.
Example (8x4 block):
16 0 0 -3 0 0 0 0
1  1 0  1 0 0 0 0
3  0 0  1 0 0 0 0
0  0 0  0 0 0 0 0
In this case dstSizeNZ.width=4, dstSizeNZ.height=3
*/

IppStatus result;
/*
ippStsNoErr          Indicates no error
ippStsNullPtrErr     Indicates an error when pSrcData or pDstData is NULL
*/

result = ippiTransform4x4Inv_VCl_16s_C1R( pSrcData, srcStep, pDstData, dstStep,
dstSizeNZ[0] );
result = ippiTransform4x4Inv_VCl_16s_C1R( pSrcData+4, srcStep, pDstData+4,
dstStep, dstSizeNZ[1] );
result = ippiTransform4x4Inv_VCl_16s_C1R( pSrcData+(srcStep*2), srcStep,
pDstData+(dstStep*2), dstStep, dstSizeNZ[2] );
result = ippiTransform4x4Inv_VCl_16s_C1R( pSrcData+(srcStep*2)+4, srcStep,
pDstData+(dstStep*2)+4, dstStep, dstSizeNZ[3] );

/*
Performs inverse transform of 4x4 subblock in 8x8 inter block with 4x4
transformation type in accordance with Annex A "Transform Specification"
SMPTE 421M
*/
}

```

## Interpolation

The interpolation VC-1 decoder functions use the following structures:

```
typedef struct _IppVCInterpolate_8u
{
    const Ipp8u* pSrc;
    Ipp32s      srcStep;
    Ipp8u*      pDst;
    Ipp32s      dstStep;
    Ipp32s      dx;
    Ipp32s      dy;
    IppiSize    roiSize;
    Ipp32s      roundControl;
} IppVCInterpolate_8u;
```

where

<i>pSrc</i>	Pointer to the source.
<i>srcStep</i>	Step of the source array pointer in bytes.
<i>pDst</i>	Pointer to the destination.
<i>dstStep</i>	Step of the destination array pointer in bytes.
<i>dx, dy</i>	Fractional parts of the motion vector in 1/4 pel units for luma (0, 1, 2, or 3) and 1/8 pel units for chroma (0, 1, ..., 7)
<i>roiSize</i>	Flag that specifies the dimensions of the region of interest (can be 16, 8, 4 or 2 in each dimension).
<i>roundControl</i>	Frame level rounding control value as described in section 8.3.7 of [SMPT421M], should be equal to 0 or 1.

The above structure is used by the [InterpolateQPBilinear\\_VC1](#) and [InterpolateQPBicubic\\_VC1](#) functions.

The intensity compensation interpolation functions use the following structure:

```
typedef struct _IppVCInterpolateBlockIC_8u {
    const Ipp8u* pSrc;
    Ipp32s      srcStep;
    Ipp8u*      pDst;
    IppiSize    sizeFrame;
    IppiSize    sizeBlock;
    IppiPoint   pointRefBlockPos;
    IppiPoint   pointVectorQuarterPix;
    Ipp8u*      pLUTTop;
    Ipp8u*      pLUTBottom;
    Ipp32u      OppositePadding;
    Ipp32u      fieldPrediction;
    Ipp32u      RoundControl;
    Ipp32u      isPredBottom;
} IppVCInterpolateBlockIC_8u;
```

See description of the above structure in the descriptions of the [InterpolateICBilinearBlock\\_VC1](#) and [InterpolateICBicubicBlock\\_VC1](#) functions.

## InterpolateQPBilinear\_VC1

*Performs bilinear quarter-pel interpolation.*

---

### Syntax

```
IppStatus ippiInterpolateQPBilinear_VC1_8u_C1R( const IppVCInterpolate_8u* pParams );
IppStatus ippiInterpolateQPBilinear_VC1_8u_C2R( const IppVCInterpolate_8u* pParams );
```

### Parameters

<i>pParams</i>	Pointer to the structure that contains parameters for interpolation:
<i>pParams-&gt;pSrc</i>	Pointer to the source (for the _C2R function, in the NV12 format)
<i>pParams-&gt;srcStep</i>	Distance in bytes between starts of the consecutive lines in the source block
<i>pParams-&gt;pDst</i>	Pointer to the destination (for the _C2R function, in the NV12 format)
<i>pParams-&gt;dstStep</i>	Distance in bytes between starts of the consecutive lines in the destination block
<i>pParams-&gt;dx</i> , <i>pParams-&gt;dy</i>	Fractional parts of the motion vector in 1/4 pel units (0, 1, 2, or 3)

<code>pParams-&gt;bitDepth</code>	Not used by this function, should be equal to 0
<code>pParams-&gt;roundControl</code>	Frame level rounding control value as described in section 8.3.7 of [SMPTE421M], should be equal to 0 or 1
<code>pParams-&gt;roiSize</code>	<p>Block size, should be 16x16:</p> <ul style="list-style-type: none"> <li>• Progressive frames/ 1-MV Macroblocks/ luma blocks</li> <li>• Interlace fields/ 1-MV Macroblocks/ luma blocks</li> </ul> <p>8x8:</p> <ul style="list-style-type: none"> <li>• Progressive frames/ 4-MV Macroblocks/ luma and chroma blocks</li> <li>• Progressive frames/ 4-MV Macroblocks/ chroma blocks</li> <li>• Interlace fields/ 4-MV Macroblocks/ chroma blocks</li> <li>• Interlace fields/ 1-MV Macroblocks/ chroma blocks</li> <li>• Interlace frames/ 1-MV Macroblocks/ chroma blocks</li> </ul> <p>8x4:</p> <ul style="list-style-type: none"> <li>• Interlace frames/ 2-MV Macroblocks/ chroma blocks</li> </ul> <p>4x4:</p> <ul style="list-style-type: none"> <li>• Interlace frames/ 4-MV Macroblocks/ chroma blocks.</li> </ul> <p>For the <code>_C2R</code> function, this block size should be 8x8, 8x4, 4x4 for each U or V component (actually, the real size of a combined UV block is 16x8, 16x4, 8x4).</p>

## Description

The functions `ippiInterpolateQPBilinear_VC1_8u_C1R` and `ippiInterpolateQPBilinear_VC1_8u_C2R` are declared in the `ippvc.h` file. The functions perform bilinear quarter-pel interpolation in accordance with 8.3.6.5.1 of [SMPTE421M].

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition when at least one of the specified pointers is NULL.
<code>ippStsBadArgErr</code>	Indicates an error condition if $dx$ or $dy$ values fall outside [0,3].
<code>ippStsSizeErr</code>	Indicates an error condition in <code>pParams-&gt;roiSize</code> .

## InterpolateQPBicubic\_VC1

*Performs bicubic quarter-pel interpolation.*

---

### Syntax

```
IppStatus ippiInterpolateQPBicubic_VC1_8u_C1R( const IppVCInterpolate_8u* pParams );
```

### Parameters

<code>pParams</code>	Pointer to the structure that contains parameters for interpolation:
<code>pParams-&gt;pSrc</code>	Pointer to the source
<code>pParams-&gt;srcStep</code>	Distance in bytes between starts of the consecutive lines in the source block
<code>pParams-&gt;pDst</code>	Pointer to the destination
<code>pParams-&gt;dstStep</code>	Distance in bytes between starts of the consecutive lines in the destination block
<code>pParams-&gt;dx,</code> <code>Params-&gt;dy</code>	Fractional parts of the motion vector in 1/4 pel units (0, 1, 2, or 3)
<code>pParams-&gt;bitDepth</code>	Not used by this function, should be equal to 0
<code>pParams-&gt;</code> <code>roundControl</code>	Frame level rounding control value as described in section 8.3.7 of [ SMPTE 421M ], should be equal to 0 or 1
<code>pParams-&gt;roiSize</code>	Block size, should be 16x16: <ul style="list-style-type: none"> <li>Progressive frames/ 1-MV Macroblocks/ luma blocks</li> <li>Interlace fields/ 1-MV Macroblocks/ luma blocks</li> <li>Interlace frames/ 1-MV Macroblocks/ luma blocks</li> </ul> 16x8: <ul style="list-style-type: none"> <li>Interlace frames/ 2-MV Macroblocks/ luma blocks</li> </ul>

8x8:

- Progressive frames/ 4-MV Macroblocks/ luma blocks
- Interlace fields/ 4-MV Macroblocks/ luma blocks
- Interlace frames/ 4-MV Macroblocks/ luma blocks.

## Description

The function `ippiInterpolateQPBicubic_VC1_8u_C1R` is declared in the `ippvc.h` file. The function performs bicubic quarter-pel interpolation in accordance with 8.3.6.5.2 of [SMPTE421M].

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition when at least one of the specified pointers is NULL.
<code>ippStsBadArgErr</code>	Indicates an error condition if $dx$ or $dy$ values fall outside [0,3].
<code>ippStsSizeErr</code>	Indicates an error condition in <code>pParams-&gt;roiSize</code> .

## InterpolateICBilinearBlock\_VC1

*Performs intensity compensation and bilinear quarter-pel interpolation for motion estimation of the luma or chroma component using an entire motion vector.*

## Syntax

```
IppStatus ippiInterpolateICBilinearBlock_VC1_8u_C1R( const IppVCInterpolateBlockIC_8u*
interpolateInfo );
```

```
IppStatus ippiInterpolateICBilinearBlock_VC1_8u_C2R( const IppVCInterpolateBlockIC_8u*
interpolateInfo );
```

## Parameters

<code>interpolateInfo</code>	Pointer to an instance of the <code>IppVCInterpolateBlockIC_8u</code> structure that contains parameters for interpolation:
<code>interpolateInfo-&gt;pSrc</code>	Pointer to the start of the reference field (or, in the case of a reference frame, pointer to the start of the reference frame); for the <code>_C2R</code> function, in the NV12 format
<code>interpolateInfo-&gt;srcStep</code>	Step of the reference frame
<code>interpolateInfo-&gt;pDst</code>	Pointer to the destination macroblock

<code>interpolateInfo-&gt;dstStep</code>	Step of the destination buffer
<code>interpolateInfo-&gt;sizeFrame</code>	Dimensions of the reference image planes
<code>interpolateInfo-&gt;sizeBlock</code>	Dimensions of the block to be interpolated; maximum size is 16 in all dimensions for the <code>_C1R</code> function and 8 for the <code>_C2R</code> function
<code>interpolateInfo-&gt;pointRefBlock</code>	Position inside reference frame calculated as sum of the current position and the integer part of motion vector
<code>interpolateInfo-&gt;pointVetoQataPix</code>	Fractional part of the motion vector size, can be 0, 1, 2 or 3
<code>interpolateInfo-&gt;pLUTTop</code>	Pointer to Intensity Compensation (IC) LUT table calculated in accordance with sections 8.3.8 and 10.3.7 of [SMPTE421M] and applied to pixels of the top field. If equal to 0, IC is not applied to this field. If both <code>pLUTTop</code> and <code>pLUTBottom</code> are equal to 0, no IC is applied to this frame.
<code>interpolateInfo-&gt;pLUTBottom</code>	Pointer to IC LUT table calculated in accordance with sections 8.3.8 and 10.3.7 of [SMPTE421M] and applied to pixels of the bottom field. If equal to 0, IC is not applied to this field.
<code>interpolateInfo-&gt;OppositePadding</code>	Flag that specifies padding correspondence between the current frame and the reference frame. 1 - if the current frame is progressive and the reference frame is interlace, or if the current frame is interlace and the reference frame is progressive. 0 - if both (current and reference) are progressive or interlace.
<code>interpolateInfo-&gt;fieldPrediction</code>	Flag that specifies prediction type for the current microblock: 0 - progressive, 1 - field. In a progressive frame all microblocks have progressive prediction type. In an interlace field all microblocks have field prediction type. In the case of an interlace frame, there are microblocks with progressive prediction type and field prediction type.
<code>interpolateInfo-&gt;roundControl</code>	Type of rounding for the current frame as described in section 8.3.7 of [SMPTE421M]; can be equal to 0 or 1
<code>interpolateInfo-&gt;isPredBottom</code>	Flag that specifies type of the reference field in the case of an interlace reference picture. 0 - if the reference field is Top, 1 - if the reference field is Bottom. 0 is for a progressive reference frame.



## Description

The functions `ippiInterpolateICBilinearBlock_VC1_8u_C1R` and `ippiInterpolateICBilinearBlock_VC1_8u_C2R` are declared in the `ippvc.h` file. The functions perform intensity compensation and bilinear quarter-pel interpolation in accordance with 8.3.6.5.1 of [SMPT421M].

The `ippiInterpolateICBilinearBlock_VC1_8u_C2R` function performs intensity compensation and bilinear quarter-pel interpolation for motion estimation of the chroma component in the NV12 format using an entire motion vector.

Note that in the case of progressive intensity compensation, top LUT and bottom LUT are the same.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition when at least one of the specified pointers <code>interpolateInfo</code> , <code>interpolateInfo-&gt;pSrc</code> , <code>interpolateInfo-&gt;pDst</code> is NULL.
<code>ippStsBadArgErr</code>	Indicates an error condition if $dx$ or $dy$ values fall outside $[0,3]$ .
<code>ippStsSizeErr</code>	Indicates an error condition in <code>pParams-&gt;roiSize</code> .

## InterpolateICBicubicBlock\_VC1

*Performs intensity compensation and bicubic quarter-pel interpolation for motion estimation of the luma or chroma component using an entire motion vector.*

## Syntax

```
IppStatus ippiInterpolateICBicubicBlock_VC1_8u_C1R( const IppVCInterpolateBlockIC_8u*
interpolateInfo );
```

## Parameters

<code>interpolateInfo</code>	Pointer to an instance of the <code>IppVCInterpolateBlockIC_8u</code> structure that contains parameters for interpolation:
<code>interpolateInfo-&gt;pSrc</code>	Pointer to the start of the reference field (or, in the case of a reference frame, pointer to the start of the reference frame)
<code>interpolateInfo-&gt;srcStep</code>	Step of the reference frame
<code>interpolateInfo-&gt;pDst</code>	Pointer to the destination macroblock
<code>interpolateInfo-&gt;dstStep</code>	Step of the destination buffer
<code>interpolateInfo-&gt;sizeFrame</code>	Dimensions of the reference image planes
<code>interpolateInfo-&gt;sizeBlock</code>	Dimensions of the block to be interpolated; maximum size is 16 in all dimensions
<code>interpolateInfo-&gt;pointRefBlockPos</code>	Position inside reference frame calculated as sum of the current position and the integer part of motion vector

<del><i>interpolateInfo-&gt;motionVectorSize</i></del>	Fractional part of the motion vector size, can be 0, 1, 2 or 3
<i>interpolateInfo-&gt;pLUTTop</i>	Pointer to Intensity Compensation (IC) LUT table calculated in accordance with sections 8.3.8 and 10.3.7 of [SMPTE421M] and applied to pixels of the top field. If equal to 0, IC is not applied to this field. If both <i>pLUTTop</i> and <i>pLUTBottom</i> are equal to 0, no IC is applied to this frame.
<i>interpolateInfo-&gt;pLUTBottom</i>	Pointer to IC LUT table calculated in accordance with sections 8.3.8 and 10.3.7 of [SMPTE421M] and applied to pixels of the bottom field. If equal to 0, IC is not applied to this field.
<i>interpolateInfo-&gt;OppositePadding</i>	Flag that specifies padding correspondence between the current frame and the reference frame. 1 - if the current frame is progressive and the reference frame is interlace, or if the current frame is interlace and the reference frame is progressive. 0 - if both (current and reference) are progressive or interlace.
<i>interpolateInfo-&gt;fieldPrediction</i>	Flag that specifies prediction type for the current microblock: 0 - progressive, 1 - field. In a progressive frame all microblocks have progressive prediction type. In an interlace field all microblocks have field prediction type. In the case of an interlace frame, there are microblocks with progressive prediction type and field prediction type.
<i>interpolateInfo-&gt;roundControl</i>	Type of rounding for the current frame as described in section 8.3.7 of [SMPTE421M]; can be equal to 0 or 1
<i>interpolateInfo-&gt;isPredBottom</i>	Flag that specifies type of the reference field in the case of an interlace reference picture. 0 - if the reference field is Top, 1 - if the reference field is Bottom. 0 is for a progressive reference frame.

## Description

The function `ippiInterpolateICBicubicBlock_Vc1_8u_C1R` is declared in the `ippvc.h` file. The function performs intensity compensation and bicubic quarter-pel interpolation for motion estimation of the luma or chroma component using an entire motion vector in accordance with 8.3.6.5.1 of [SMPTE421M].

Note that in the case of progressive intensity compensation, top LUT and bottom LUT are the same.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition when at least one of the specified pointers <i>interpolateInfo</i> , <i>interpolateInfo-&gt;pSrc</i> , <i>interpolateInfo-&gt;pDst</i> is NULL.
<code>ippStsBadArgErr</code>	Indicates an error condition if <i>dx</i> or <i>dy</i> values fall outside [0,3].
<code>ippStsSizeErr</code>	Indicates an error condition in <i>pParams-&gt;roiSize</i> .

## Smoothing

### SmoothingLuma\_VerEdge\_VC1

*Performs smoothing filtering on the vertical edge (internal or external) of luma 16X16 macroblocks.*

## Syntax

```
IppStatus ippiSmoothingLuma_VerEdge_VC1_16s8u_C1R( Ipp16s* pSrcDstLeft, Ipp32s
srcDstLeftStep, Ipp16s* pSrcDstRight, Ipp32s srcDstRightStep, Ipp8u* pDst, Ipp32s dstStep,
Ipp32u fieldNeighbourFlag, Ipp32u edgeDisableFlag );
```

## Parameters

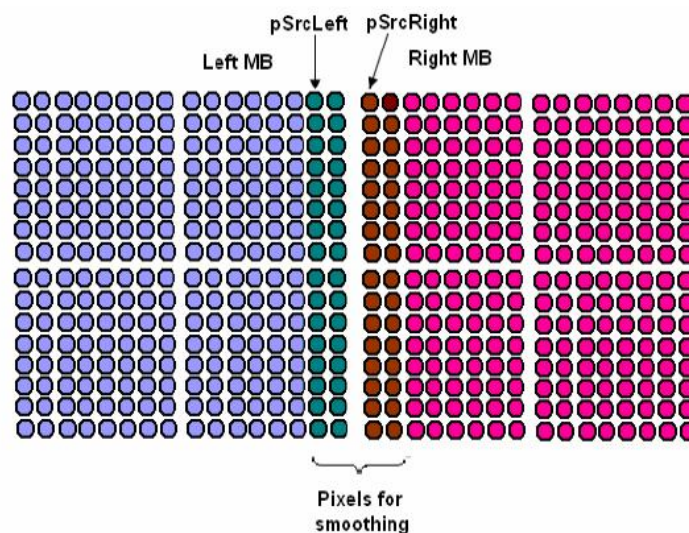
<i>pSrcDstLeft</i>	Pointer to the first pixel of the column of the top left block of the left macroblock, from which the smoothing starts.
<i>srcDstLeftStep</i>	Step for transfer to the next row in the left macroblock.
<i>pSrcDstRight</i>	Pointer to the first pixel of the top left block of the right macroblock.
<i>srcDstRightStep</i>	Step for transfer to the next row in the right macroblock.
<i>pDst</i>	Pointer to the first pixel of the right macroblock in the Y-plane.
<i>dstStep</i>	Distance in bytes between starts of the consecutive lines in the Y-plane.
<i>fieldNeighbourFlag</i>	Indicates the field macroblock property (2 bits): <ul style="list-style-type: none"> <li>• if (<i>fieldNeighbourFlag</i> &amp; <i>VC1_FIELD_LEFT_MB</i>) - the left macroblock is field decoded,</li> <li>• if (<i>fieldNeighbourFlag</i> &amp; <i>VC1_FIELD_RIGHT_MB</i>) - the right macroblock is field decoded.</li> </ul>
<i>edgeDisableFlag</i>	Flag indicating that <ul style="list-style-type: none"> <li>• if (<i>edgeDisableFlag</i> &amp; <i>IPPVC_EDGE_HALF_1</i>), then the upper vertical edge is disabled for smoothing,</li> <li>• if (<i>edgeDisableFlag</i> &amp; <i>IPPVC_EDGE_HALF_2</i>), then the bottom vertical edge is disabled for smoothing.</li> </ul>

## Description

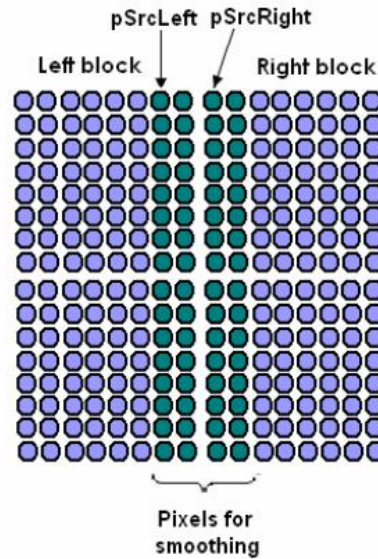
The function `ippiSmoothingLuma_VerEdge_VC1_16s8u_C1R` is declared in the `ippvc.h` file. The function performs smoothing filtering on the vertical edge (internal or external) of the luma 16x16 macroblocks. See 8.5 of [SMPT421M].

## Vertical Smoothing Between Two Macroblocks

---



## Vertical Smoothing Inside One Macroblock



All samples of the source block should be within the range  $[-2048; 2047]$ . After smoothing, all samples of the destination block are within the range  $[-512; 511]$ .

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition when source or destination pointers are <code>NULL</code> .

## SmoothingLuma\_HorEdge\_VC1

*Performs smoothing filtering on the horizontal edge (internal or external) of luma 16X16 macroblocks.*

### Syntax

```
ippStatus ippiSmoothingLuma_HorEdge_VC1_16s8u_C1R( Ipp16s* pSrcDstUpper, Ipp32s
srcDstUpperStep, Ipp16s* pSrcDstBottom, Ipp32s srcDstBottomStep, Ipp8u* pDst, Ipp32s
dstStep, Ipp32u edgeDisableFlag );
```

### Parameters

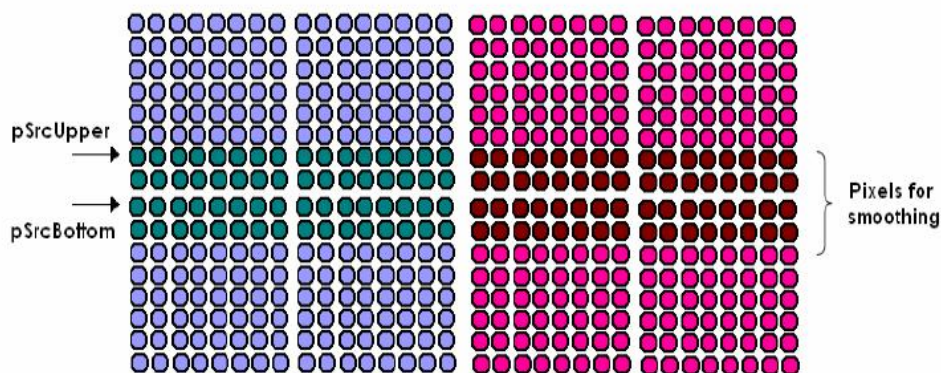
<code>pSrcDstUpper</code>	Pointer to the first pixel of the bottom left block row of the upper macroblock, from which the smoothing starts.
<code>srcDstUpperStep</code>	Step for transfer to the next row in the upper macroblock.
<code>pSrcDstBottom</code>	Pointer to the first pixel of the top left block of the bottom macroblock.
<code>srcDstBottomStep</code>	Step for transfer to the next row in the bottom macroblock.

<i>pDst</i>	Pointer to the first pixel of the bottom macroblock in the Y-plane.
<i>dstStep</i>	Distance in bytes between starts of the consecutive lines in the Y-plane.
<i>edgeDisableFlag</i>	Flag indicating that <ul style="list-style-type: none"><li>• if (<i>edgeDisableFlag</i> &amp; IPPVC_EDGE_HALF_1), then the left horizontal edge is disabled for smoothing,</li><li>• if (<i>edgeDisableFlag</i> &amp; IPPVC_EDGE_HALF_2), then the right horizontal edge is disabled for smoothing.</li></ul>

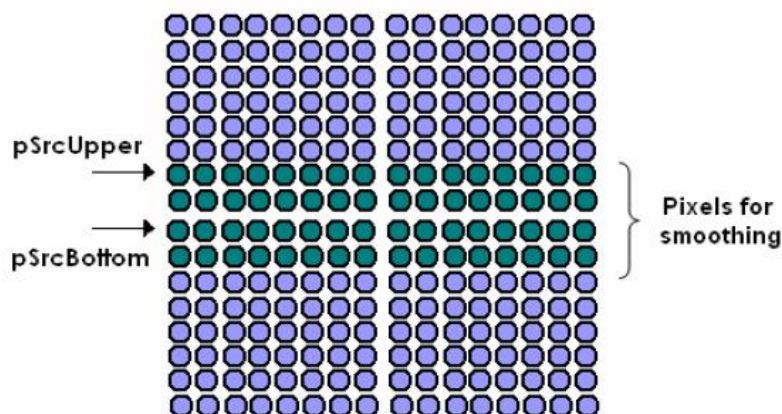
## Description

The function `ippiSmoothingLuma_HorEdge_VC1_16s8u_C1R` is declared in the `ippvc.h` file. The function performs smoothing filtering on the horizontal edge (internal or external) of the luma 16x16 macroblocks. See 8.5 of [SMPT421M].

### Horizontal Smoothing Between Two Macroblocks



### Horizontal Smoothing Inside One Macroblock



All samples of the source block should be within the range  $[-2048; 2047]$ . After smoothing, all samples of the destination block are within the range  $[-512; 511]$ .

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition when source or destination pointers are NULL.

## SmoothingChroma\_VerEdge\_VC1

*Performs smoothing filtering on the vertical edge (internal or external) of chroma blocks.*

---

## Syntax

```
IppStatus ippiSmoothingChroma_VerEdge_VC1_16s8u_C1R( Ipp16s* pSrcDstLeft, Ipp32s
srcDstLeftStep, Ipp16s* pSrcDstRight, Ipp32s srcDstRightStep, Ipp8u* pDst, Ipp32s dstStep
);
```

```
IppStatus ippiSmoothingChroma_VerEdge_VC1_16s8u_C2R( Ipp16s* pSrcDstLeftU, Ipp32u
srcDstLeftStepU, Ipp16s* pSrcDstRightU, Ipp32u srcDstRightStepU, Ipp16s* pSrcDstLeftV,
Ipp32u srcDstLeftStepV, Ipp16s* pSrcDstRightV, Ipp32u srcDstRightStepV, Ipp8u* pDst, Ipp32u
dstStep );
```

## Parameters

<code>pSrcDstLeft</code>	Pointer to the first pixel of the column of the left block, from which the smoothing starts.
<code>srcDstLeftStep</code>	Step for transfer to the next row in the left block.
<code>pSrcDstRight</code>	Pointer to the first pixel of the right block.
<code>srcDstRightStep</code>	Step for transfer to the next row in the right block.
<code>pSrcDstLeftU</code>	Pointer to the first pixel of the column of the left U difference block, from which the smoothing starts. <pre> 0 UUUUUUUU 1 UUUUUUUU ... 7 UUUUUUUU </pre>
<code>srcDstLeftStepU</code>	Step for transfer to the next row in the left U block.
<code>pSrcDstRightU</code>	Pointer to the first pixel of the right U difference block.
<code>srcDstRightStepU</code>	Step for transfer to the next row in the right U block.
<code>pSrcDstLeftV</code>	Pointer to the first pixel of the column of the left V difference block, from which the smoothing starts. <pre> 0 VVVVVVVV 1 VVVVVVVV ... 7 VVVVVVVV </pre>
<code>srcDstLeftStepV</code>	Step for transfer to the next row in the left V difference block.
<code>pSrcDstRightV</code>	Pointer to the first pixel of the right V block.
<code>srcDstRightStepV</code>	Step for transfer to the next row in the right V block.
<code>pDst</code>	Pointer to the first pixel of the right block in the Y-plane.



For `ippiSmoothingChroma_VerEdge_VC1_16s8u_C2R`, pointer to the first pixel of the right block in the UV-plane in the NV12 format:

```
0  UV UV UV UV UV UV UV UV
1  UV UV UV UV UV UV UV UV
...
7  UV UV UV UV UV UV UV UV
```

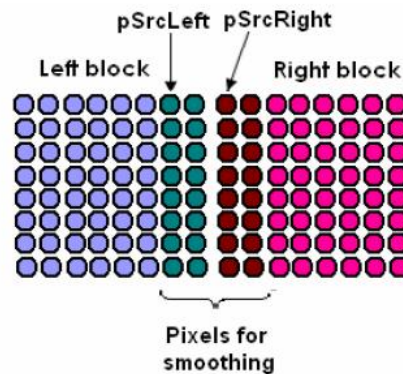
*dstStep*

Distance in bytes between starts of the consecutive lines in the Y-plane. For `ippiSmoothingChroma_VerEdge_VC1_16s8u_C2R`, UV-plane step in bytes.

## Description

The functions `ippiSmoothingChroma_VerEdge_VC1_16s8u_C1R` and `ippiSmoothingChroma_VerEdge_VC1_16s8u_C2R` are declared in the `ippvc.h` file. The function `ippiSmoothingChroma_VerEdge_VC1_16s8u_C1R` performs smoothing filtering on the vertical edge (internal or external) of the chroma 8x8 blocks. The function `ippiSmoothingChroma_VerEdge_VC1_16s8u_C2R` performs smoothing filtering on the vertical edge (internal or external) of the chroma 8x8 difference and 16x8 plane blocks. See 8.5 of [SMPT421M].

## Vertical Smoothing Between Two Chroma Blocks



All samples of the source block should be within the range  $[-2048; 2047]$ . After smoothing, all samples of the destination block are within the range  $[-512; 511]$ .

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition when source or destination pointers are <code>NULL</code> .

## SmoothingChroma\_HorEdge\_VC1

*Performs smoothing filtering on the horizontal edge  
(internal or external) of chroma blocks.*

---

### Syntax

```
IppStatus ippiSmoothingChroma_HorEdge_VC1_16s8u_C1R( Ipp16s* pSrcDstUpper, Ipp32s
srcDstUpperStep, Ipp16s* pSrcDstBottom, Ipp32s srcDstBottomStep, Ipp8u* pDst, Ipp32s
dstStep );
```

```
IppStatus ippiSmoothingChroma_HorEdge_VC1_16s8u_C2R( Ipp16s* pSrcDstUpperU, Ipp32u
srcDstUpperStepU, Ipp16s* pSrcDstBottomU, Ipp32u srcDstBottomStepU, Ipp16s* pSrcDstUpperV,
Ipp32u srcDstUpperStepV, Ipp16s* pSrcDstBottomV, Ipp32u srcDstBottomStepV, Ipp8u* pDst,
Ipp32u dstStep );
```

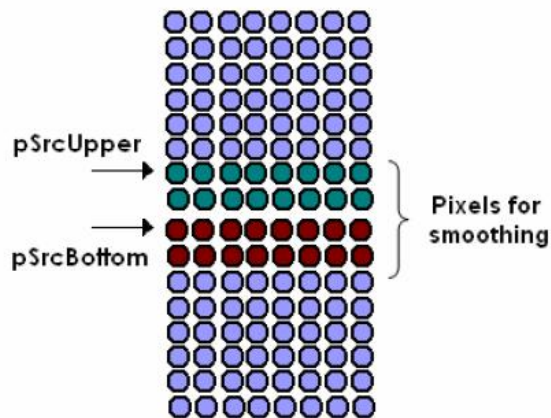
### Parameters

<i>pSrcDstUpper</i>	Pointer to the first pixel of the row of the upper block, from which the smoothing starts.
<i>srcDstUpperStep</i>	Step for transfer to the next row in the upper block.
<i>pSrcDstBottom</i>	Pointer to the first pixel of the bottom block.
<i>srcDstBottomStep</i>	Step for transfer to the next row in the bottom block.
<i>pSrcDstUpperU</i>	Pointer to the first pixel of the column of the upper U difference block, from which the smoothing starts.
<i>srcDstUpperStepU</i>	Step for transfer to the next row in the upper U block.
<i>pSrcDstBottomU</i>	Pointer to the first pixel of the bottom U difference block.
<i>srcDstBottomStepU</i>	Step for transfer to the next row in the bottom U block.
<i>pSrcDstUpperV</i>	Pointer to the first pixel of the row of the upper V difference block, from which the smoothing starts.
<i>srcDstUpperStepV</i>	Step for transfer to the next row in the upper V difference block.
<i>pSrcDstBottomV</i>	Pointer to the first pixel of the bottom V block.
<i>srcDstBottomStepV</i>	Step for transfer to the next row in the bottom V difference block.
<i>pDst</i>	Pointer to the first pixel of the bottom block in the Y-plane. For <code>ippiSmoothingChroma_HorEdge_VC1_16s8u_C2R</code> , pointer to the first pixel of the right block in the UV-plane in the NV12 format: <pre> 0  UV UV UV UV UV UV UV UV 1  UV UV UV UV UV UV UV UV ... 7  UV UV UV UV UV UV UV UV </pre>
<i>dstStep</i>	Distance in bytes between starts of the consecutive lines in the Y-plane. For <code>ippiSmoothingChroma_VerEdge_VC1_16s8u_C2R</code> , UV-plane step in bytes.

## Description

The functions `ippiSmoothingChroma_HorEdge_VC1_16s8u_C1R` and `ippiSmoothingChroma_HorEdge_VC1_16s8u_C2R` are declared in the `ippvc.h` file. The function `ippiSmoothingChroma_HorEdge_VC1_16s8u_C1R` performs smoothing filtering on the horizontal edge (internal or external) of the chroma 8x8 blocks. The function `ippiSmoothingChroma_HorEdge_VC1_16s8u_C2R` performs smoothing filtering on the horizontal edge (internal or external) of the chroma 8x8 difference and 16x8 plane blocks. See 8.5 of [SMPTE421M].

## Horizontal Smoothing Between Two Chroma Block



All samples of the source block should be within the range  $[-2048; 2047]$ . After smoothing, all samples of the destination block are within the range  $[-512, 511]$ .

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition when source or destination pointers are <code>NULL</code> .

## Deblocking

### FilterDeblockingLuma\_VerEdge\_VC1

*Performs deblocking filtering on the vertical edge (internal or external) of luma 16X16 macroblocks.*

---

#### Syntax

```
IppStatus ippiFilterDeblockingLuma_VerEdge_VC1_8u_C1IR( Ipp8u* pSrcDst, Ipp32s srcdstStep,
Ipp32s pQuant, Ipp32u EdgeDisableFlag );
```

#### Parameters

<i>pSrcDst</i>	Pointer to the first pixel of the right block in the Y-plane.
<i>srcdstStep</i>	Distance in bytes between starts of the consecutive lines in the Y-plane.
<i>pQuant</i>	Picture quantizer scale.
<i>EdgeDisableFlag</i>	Flag indicating that <ul style="list-style-type: none"> <li>• if (EdgeDisableFlag &amp; IPPVC_EDGE_QUARTER_1), then the first vertical quarter of block edge is disabled for deblocking,</li> <li>• if (EdgeDisableFlag &amp; IPPVC_EDGE_QUARTER_2), then the second vertical quarter of block edge is disabled for deblocking,</li> <li>• if (EdgeDisableFlag &amp; IPPVC_EDGE_QUARTER_3), then the third vertical quarter of block edge is disabled for deblocking,</li> <li>• if (EdgeDisableFlag &amp; IPPVC_EDGE_QUARTER_4), then the fourth vertical quarter of block edge is disabled for deblocking,</li> <li>• if (EdgeDisableFlag &amp; IPPVC_EDGE_HALF_1), then the upper vertical edge of the block is disabled for deblocking,</li> <li>• if (EdgeDisableFlag &amp; IPPVC_EDGE_HALF_2), then the bottom vertical edge of the block is disabled for deblocking.</li> </ul> <p>Conditions 1, 2, 3, 4 can be used for the internal edge of macroblock in the case of progressive P frames. Conditions 5, 6 can be used for other cases.</p>

#### Description

The function `ippiFilterDeblockingLuma_VerEdge_VC1_8u_C1IR` is declared in the `ippvc.h` file. The function performs deblocking filtering on the vertical edge (internal or external) of the luma 16x16 macroblocks. See 8.6 of [SMPTE421M].

#### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition when at least one of the specified pointers is NULL.

## FilterDeblockingLuma\_HorEdge\_VC1

*Performs deblocking filtering on the horizontal edge (internal or external) of luma 16X16 macroblocks.*

### Syntax

```
IppStatus ippiFilterDeblockingLuma_HorEdge_VC1_8u_C1IR( Ipp8u* pSrcDst, Ipp32s srcdstStep,
Ipp32s pQuant, Ipp32u EdgeDisableFlag );
```

### Parameters

<i>pSrcDst</i>	Pointer to the first pixel of the lower block in the Y-plane.
<i>srcdstStep</i>	Distance between starts of the consecutive lines in the array.
<i>pQuant</i>	Picture quantizer scale.
<i>EdgeDisableFlag</i>	Flag indicating that <ul style="list-style-type: none"> <li>• if (EdgeDisableFlag &amp; IPPVC_EDGE_QUARTER_1), then the first horizontal quarter of block edge is disabled for deblocking,</li> <li>• if (EdgeDisableFlag &amp; IPPVC_EDGE_QUARTER_2), then the second horizontal quarter of block edge is disabled for deblocking,</li> <li>• if (EdgeDisableFlag &amp; IPPVC_EDGE_QUARTER_3), then the third horizontal quarter of block edge is disabled for deblocking,</li> <li>• if (EdgeDisableFlag &amp; IPPVC_EDGE_QUARTER_4), then the fourth horizontal quarter of block edge is disabled for deblocking,</li> <li>• if (EdgeDisableFlag &amp; IPPVC_EDGE_HALF_1), then the left horizontal edge of the block is disabled for deblocking,</li> <li>• if (EdgeDisableFlag &amp; IPPVC_EDGE_HALF_2), then the right horizontal edge of the block is disabled for deblocking.</li> </ul>

Conditions 1, 2, 3, 4 can be used for the internal edge of macroblock in the case of progressive P frames. Conditions 5, 6 can be used for other cases.

### Description

The function `ippiFilterDeblockingLuma_HorEdge_VC1_8u_C1IR` is declared in the `ippvc.h` file. The function performs deblocking filtering on the horizontal edge (internal or external) of the luma 16x16 macroblocks. See 8.6 of [SMPT421M].

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition when at least one of the specified pointers is NULL.

## FilterDeblockingChroma\_VerEdge\_VC1

*Performs deblocking filtering on the vertical edge (internal or external) of chroma 8X8 macroblock.*

---

### Syntax

```
IppStatus ippiFilterDeblockingChroma_VerEdge_VC1_8u_C1IR( Ipp8u* pSrcDst, Ipp32s srcdstStep,
Ipp32s pQuant, Ipp32u EdgeDisableFlag );
```

### Parameters

<i>pSrcDst</i>	Pointer to the first pixel of the right block in the U- or Y-plane.
<i>srcdstStep</i>	Distance in bytes between starts of the consecutive lines in the plane.
<i>pQuant</i>	Picture quantizer scale.
<i>EdgeDisableFlag</i>	Flag indicating that <ul style="list-style-type: none"> <li>if (<code>EdgeDisableFlag &amp; IPPVC_EDGE_HALF_1</code>), then the upper vertical edge of the block is disabled for deblocking,</li> <li>if (<code>EdgeDisableFlag &amp; IPPVC_EDGE_HALF_2</code>), then the bottom vertical edge of the block is disabled for deblocking.</li> </ul>

These conditions can be used for the internal edge of macroblock in the case of progressive P frames. In the other cases, the flag should be equal to 0.

### Description

The function `ippiFilterDeblockingChroma_VerEdge_VC1_8u_C1IR` is declared in the `ippvc.h` file. The function performs deblocking filtering on the vertical edge (internal or external) of the chroma 8x8 macroblock. See 8.6 of [SMPT421M].

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition when at least one of the specified pointers is NULL.

## FilterDeblockingChroma\_HorEdge\_VC1

*Performs deblocking filtering on the horizontal edge (internal or external) of chroma 8X8 macroblock.*

---

### Syntax

```
IppStatus ippiFilterDeblockingChroma_HorEdge_VC1_8u_C1IR( Ipp8u* pSrcDst, Ipp32s srcdstStep,
Ipp32s pQuant, Ipp32u EdgeDisableFlag );
```

### Parameters

<i>pSrcDst</i>	Pointer to the first pixel of the lower block in the U- or Y-plane.
----------------	---

<i>srcDstStep</i>	Distance in bytes between starts of the consecutive lines in the plane.
<i>pQuant</i>	Picture quantizer scale.
<i>EdgeDisableFlag</i>	Flag indicating that <ul style="list-style-type: none"> <li>• if (<code>EdgeDisableFlag &amp; IPPVC_EDGE_HALF_1</code>), then the upper horizontal edge of the block is disabled for deblocking,</li> <li>• if (<code>EdgeDisableFlag &amp; IPPVC_EDGE_HALF_2</code>), then the bottom horizontal edge of the block is disabled for deblocking.</li> </ul> <p>These conditions can be used for the internal edge of macroblock in the case of progressive P frames. In the other cases, the flag should be equal to 0.</p>

## Description

The function `ippiFilterDeblockingChroma_HorEdge_VC1_8u_C1IR` is declared in the `ippvc.h` file. The function performs deblocking filtering on the horizontal edge (internal or external) of the chroma 8x8 macroblock. See 8.6 of [SMPTE421M].

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition when at least one of the specified pointers is NULL.

## Dequantization

## QuantInvIntraUniform\_VC1, QuantInvIntraNonuniform\_VC1

*Perform uniform/nonuniform dequantization of 8X8 intra blocks.*

## Syntax

```
ippStatus ippiQuantInvIntraUniform_VC1_16s_C1IR(Ipp16s* pSrcDst, Ipp32s srcDstStep, Ipp32s doubleQuant, IppSize* pDstSizeNZ );
```

```
ippStatus ippiQuantInvIntraNonuniform_VC1_16s_C1IR(Ipp16s* pSrcDst, Ipp32s srcDstStep, Ipp32s doubleQuant, IppSize* pDstSizeNZ );
```

```
ippStatus ippiQuantInvIntraUniform_VC1_16s_C1R(const Ipp16s* pSrc, Ipp32s srcStep, Ipp16s* pDst, Ipp32s dstStep, Ipp32s doubleQuant, IppSize* pDstSizeNZ);
```

```
ippStatus ippiQuantInvIntraNonuniform_VC1_16s_C1R(const Ipp16s* pSrc, Ipp32s srcStep, Ipp16s* pDst, Ipp32s dstStep, Ipp32s doubleQuant, IppSize* pDstSizeNZ);
```

## Parameters

<i>pSrcDst</i>	Pointer to the source and destination block.
<i>srcDstStep</i>	Distance in bytes between starts of the consecutive lines in the source and destination blocks.

<i>doubleQuant</i>	Dequant coefficient; should be within the range [2, 62].
<i>pDstSizeNZ</i>	Pointer to a size of the top left subblock with non-zero coefficients. This value is calculated by this function and could be used for inverse transformation. See <a href="#">Figure "Inter 8x4 Block Dequantization"</a> .
<i>pSrc</i>	Pointer to the source block.
<i>srcStep</i>	Distance in bytes between starts of the consecutive lines in the source block.
<i>pDst</i>	Pointer to the destination block.
<i>dstStep</i>	Distance in bytes between starts of the consecutive lines in the destination block.

## Description

These functions are declared in the `ippvc.h` file.

The *doubleQuant* parameter should be equal to  $2 * \text{MQANT} + \text{HALFQP}$ , where `MQANT` and `HALFQP` are defined in VC-1 standard.

The functions perform dequantization of all coefficients in intra block (except DC) according to 8.1.2.8 of [SMPTE421M].

In the case of uniform dequantization:

```
a[i]=quant_coef*a[i];
```

In the case of nonuniform dequantization:

```
a[i]=quant_coef*a[i] + sign(a[i])*(quant_coef >> 1);
```

where *a[i]* is the AC coefficient of the intra block, and *quant\_coef* is the dequant coefficient.

## Intra 8x8 Block Dequantization

---

16	0	0	-3	0	1	1	0
1	1	0	1	0	0	0	0
3	0	0	1	0	0	0	0
0	0	0	1	0	0	0	0
0	0	0	-1	0	0	0	0
0	1	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

---

In this case, *pDstSizeNZ* -> width is equal to 7, and *pDstSizeNZ* -> height is equal to 6.

## Return Values

<i>ippStsNoErr</i>	Indicates no error.
--------------------	---------------------



`ippStsNullPtrErr`

Indicates an error condition when *pSrcDst* or *pDstSizeNZ* is NULL.

## Example

The code example below shows how to use `QuantInvIntraUniform_VC1` function.

```
{
Ipp16s* pSrcDstData = pointer_on_source_and_destination_block;
Ipp32s doubleQuant = quant_coefficient; /* It should be an even number in the
range [2,62]; */
Ipp32s srcDstStep = source_and_destination_step; /* Step of the pointer
pSrcDstData (source and destination array) in bytes */
IppiSize dstSizeNZ;
/*
The size of top left subblock with non-zero coefficients. This value is
calculated by this function and could be used for inverse transformation.
Example (intra block):
16 0 0 -3 0 1 1 0
1 1 0 1 0 0 0 0
3 0 0 1 0 0 0 0
0 0 0 1 0 0 0 0
0 0 0 -1 0 0 0 0
0 1 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
In this case dstSizeNZ.width is equal to 7, dstSizeNZ.height is equal to 6
*/
IppStatus result;
/*
ippStsNoErr          Indicates no error
ippStsNullPtrErr     Indicates an error when pSrcDstData is NULL
*/

result = ippiQuantInvIntraUniform_VC1_16s_C1IR( pSrcDstData, srcDstStep,
doubleQuant, &dstSizeNZ );

/* Performs uniform dequantization process for all coefficients in the intra
block (except DC) according to 8.1.2.8 "Inverse AC Coefficient Quantization"
specification SMPTE 421M
*/
}
```

The code example below shows how to use `QuantInvIntraNonuniform_VC1` function.

```
{
Ipp16s* pSrcDstData = pointer_on_source_and_destination_block;
Ipp32s doubleQuant = quant_coefficient; /* It should be an even number in the
range [2,62]; */
Ipp32s srcDstStep = source_and_destination_step; /* Step of the pointer
pSrcDstData (source and destination array) in bytes */
IppiSize dstSizeNZ;
/*
The size of top left subblock with non-zero coefficients. This value is
calculated by this function and could be used for inverse transformation.
Example (intra block):
16 0 0 -3 0 1 1 0
1 1 0 1 0 0 0 0
3 0 0 1 0 0 0 0
0 0 0 1 0 0 0 0
0 0 0 -1 0 0 0 0
0 1 0 0 0 0 0 0
0 0 0 0 0 0 0 0
*/
}
```

```

0 0 0 0 0 0 0
In this case dstSizeNZ.width is equal to 7, dstSizeNZ.height is equal to 6
*/
IppStatus result;
/*
ippStsNoErr      Indicates no error
ippStsNullPtrErr Indicates an error when pSrcDstData is NULL
*/

result = ippiQuantInvIntraNonuniform_VC1_16s_C1IR( pSrcDstData, srcDstStep,
doubleQuant, &dstSizeNZ );

/* Performs nonuniform dequantization process for all coefficients in the intra
block (except DC) according to 8.1.2.8 "Inverse AC Coefficient Quantization"
specification SMPTE 421M
*/
}

```

## QuantInvInterUniform\_VC1, QuantInvInterNonuniform\_VC1

*Perform uniform/nonuniform dequantization of inter blocks.*

---

### Syntax

```

IppStatus ippiQuantInvInterUniform_VC1_16s_C1IR(Ipp16s* pSrcDst, Ipp32s srcDstStep, Ipp32s
doubleQuant, IppiSize roiSize, IppSize* pDstSizeNZ );

IppStatus ippiQuantInvInterNonuniform_VC1_16s_C1IR(Ipp16s* pSrcDst, Ipp32s srcDstStep,
Ipp32s doubleQuant, IppiSize roiSize, IppSize* pDstSizeNZ );

IppStatus ippiQuantInvInterUniform_VC1_16s_C1R(const Ipp16s* pSrc, Ipp32s srcStep, Ipp16s*
pDst, Ipp32s dstStep, Ipp32s doubleQuant, IppiSize roiSize, IppSize* pDstSizeNZ);

IppStatus ippiQuantInvInterNonuniform_VC1_16s_C1R(const Ipp16s* pSrc, Ipp32s srcStep,
Ipp16s* pDst, Ipp32s dstStep, Ipp32s doubleQuant, IppiSize roiSize, IppSize* pDstSizeNZ);

```

### Parameters

<i>pSrcDst</i>	Pointer to the source and destination block.
<i>srcDstStep</i>	Distance in bytes between starts of the consecutive lines in the source and destination blocks.
<i>doubleQuant</i>	Dequant coefficient; should be within the range [2, 62].
<i>roiSize</i>	Inter block size; should be 8x8, 8x4, 4x8 or 4x4.
<i>pDstSizeNZ</i>	Pointer to a size of the top left subblock with non-zero coefficients. This value is calculated by this function and could be used for inverse transformation. See <a href="#">Figure "Inter 8x4 Block Dequantization"</a> .
<i>pSrc</i>	Pointer to the source block.
<i>srcStep</i>	Distance in bytes between starts of the consecutive lines in the source block.
<i>pDst</i>	Pointer to the destination block.
<i>dstStep</i>	Distance in bytes between starts of the consecutive lines in the destination block.

## Description

The functions `ippiQuantInvInterUniform_Vc1_16s_C1IR` and `ippiQuantInvInterNonuniform_Vc1_16s_C1IR` are declared in the `ippvc.h` file.

The *doubleQuant* parameter should be equal to  $2 * \text{MQANT} + \text{HALFQP}$ , where `MQANT` and `HALFQP` are defined in VC-1 standard.

The functions perform dequantization of inter blocks according to 8.1.2.8 of [SMPTE421M].

In the case of uniform dequantization:

```
a[i]=quant_coef*a[i];
```

In the case of nonuniform dequantization:

```
a[i]=quant_coef*a[i] + sign(a[i])*(quant_coef >> 1);
```

where `a[i]` is the intra block coefficient, and `quant_coef` is the dequant coefficient.

## Inter 8x4 Block Dequantization

16	0	0	-3	0	0	0	0
1	1	0	1	0	0	0	0
3	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

In this case, `pDstSizeNZ -> width` is equal to 4, and `pDstSizeNZ -> height` is equal to 3.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition when <code>pSrcDst</code> or <code>pDstSizeNZ</code> is NULL.

## Example

The code example below shows how to use `QuantInvInterUniform_Vc1` function.

```
{
Ipp16s* pSrcDstData = pointer_on_source_and_destination_block;
Ipp32s doubleQuant = quant_coefficient; /* It should be an even number in the
range [2,62];*/
Ipp32s srcDstStep = source_and_destination_step; /* Step of the pointer
pSrcDstData (source and destination array) in bytes */
IppSize roiSize = inter_block_size;
/* It should be:
8x8 (roiSize.width=8, roiSize.height=8)
4x8 (roiSize.width=4, roiSize.height=8)
8x4 (roiSize.width=8, roiSize.height=4)
4x4 (roiSize.width=4, roiSize.height=4)
*/
IppSize dstSizeNZ[2];
/*
```

The size of top left subblock with non-zero coefficients. This value is calculated by this function and could be used for inverse transformation.

Example (inter 8x4 block):

```
16 0 0 -3 0 0 0 0
1  1 0  1 0 0 0 0
3  0 0  1 0 0 0 0
0  0 0  0 0 0 0 0
```

In this case dstSizeNZ.width is equal to 4, dstSizeNZ.height is equal to 3

\*/

IppStatus result;

/\*

ippStsNoErr Indicates no error

ippStsNullPtrErr Indicates an error when pSrcDstData is NULL

ippStsSizeErr Indicates an error when roiSize.height is not equal 4 or 8  
or roiSize.width is not equal 4 or 8

\*/

/\* in case of 8x4\*/

```
result = ippiQuantInvInterUniform_Vc1_16s_C1IR( pSrcDstData, srcDstStep,
doubleQuant, &dstSizeNZ[0] );
```

```
result = ippiQuantInvInterUniform_Vc1_16s_C1IR( pSrcDstData+ (srcDstStep*2),
srcDstStep, doubleQuant, &dstSizeNZ[1] );
```

/\* Performs uniform dequantization process for inter block according to 8.1.2.8  
"Inverse AC Coefficient Quantization" specification SMPTE 421M

\*/

}

The code example below shows how to use QuantInvInterNonuniform\_Vc1 function.

```
{
Ipp16s* pSrcDstData = pointer_on_source_and_destination_block;
Ipp32s doubleQuant = quant_coefficient; /* It should be an even number in the
range [2,62];*/
```

```
Ipp32s srcDstStep = source_and_destination_step; /* Step of the pointer
```

```
pSrcDstData (source and destination array) in bytes */
```

```
IppSize roiSize = inter_block_size;
```

/\* It should be:

8x8 (roiSize.width=8, roiSize.height=8)

4x8 (roiSize.width=4, roiSize.height=8)

8x4 (roiSize.width=8, roiSize.height=4)

4x4 (roiSize.width=4, roiSize.height=4)

\*/

```
IppiSize dstSizeNZ[4];
```

/\*

The size of top left subblock with non-zero coefficients. This value is calculated by this function and could be used for inverse transformation.

Example (inter 4x4 block):

```
16 0 0 -3
1  1 0  1
3  0 0  1
0  0 0  0
```

In this case dstSizeNZ.width is equal to 4, dstSizeNZ.height is equal to 3

\*/

IppStatus result;

/\*

ippStsNoErr Indicates no error

ippStsNullPtrErr Indicates an error when pSrcDstData is NULL

ippStsSizeErr Indicates an error when roiSize.height is not equal to 4 or 8  
or roiSize.width is not equal to 4 or 8

\*/

```

/* in case of 4x4*/
result = ippiQuantInvInterNonuniform_VC1_16s_C1IR( pSrcDstData, srcDstStep,
doubleQuant, &dstSizeNZ[0] );
result = ippiQuantInvInterNonuniform_VC1_16s_C1IR( pSrcDstData+4, srcDstStep,
doubleQuant, &dstSizeNZ[1] );
result = ippiQuantInvInterNonuniform_VC1_16s_C1IR( pSrcDstData+( srcDstStep*2),
srcDstStep, doubleQuant, &dstSizeNZ[2] );
result = ippiQuantInvInterNonuniform_VC1_16s_C1IR( pSrcDstData+(srcDstStep*2)+4,
srcDstStep, doubleQuant, &dstSizeNZ[3] );

/* Performs nonuniform dequantization process for inter block according to
8.1.2.8 "Inverse AC Coefficient Quantization" specification SMPTE 421M
*/
}

```

## Range Reduction

### RangeMapping\_VC1

*Performs range map transformation.*

#### Syntax

```

IppStatus ippiRangeMapping_VC1_8u_C1R( const Ipp8u* pSrc, Ipp32s srcStep, Ipp8u* pDst,
Ipp32s dstStep, IppiSize roiSize, Ipp32s rangeMapParam );

```

#### Parameters

<i>pSrc</i>	Pointer to the source block. Block coefficient could be within the range of [0, 255].
<i>srcStep</i>	Distance in bytes between starts of the consecutive lines in the source block.
<i>pDst</i>	Pointer to the destination block.
<i>dstStep</i>	Distance in bytes between starts of the consecutive lines in the destination block.
<i>roiSize</i>	Size of the source block.
<i>rangeMapParam</i>	Parameter for the range map; should be within the range of [0, 7].

#### Description

The function `ippiRangeMapping_VC1_8u_C1R` is declared in the `ippvc.h` file. The function performs range map transformation according to 6.2.15.1 of [SMPTE421M].

the function calculates each coefficient of the source block as follows:

```
Y[n]=CLIP((((Y[n]-128)*(RANGE_MAPY + 9) + 4) >> 3) + 128);
```

#### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition when <i>pSrc</i> or <i>pDst</i> is NULL.

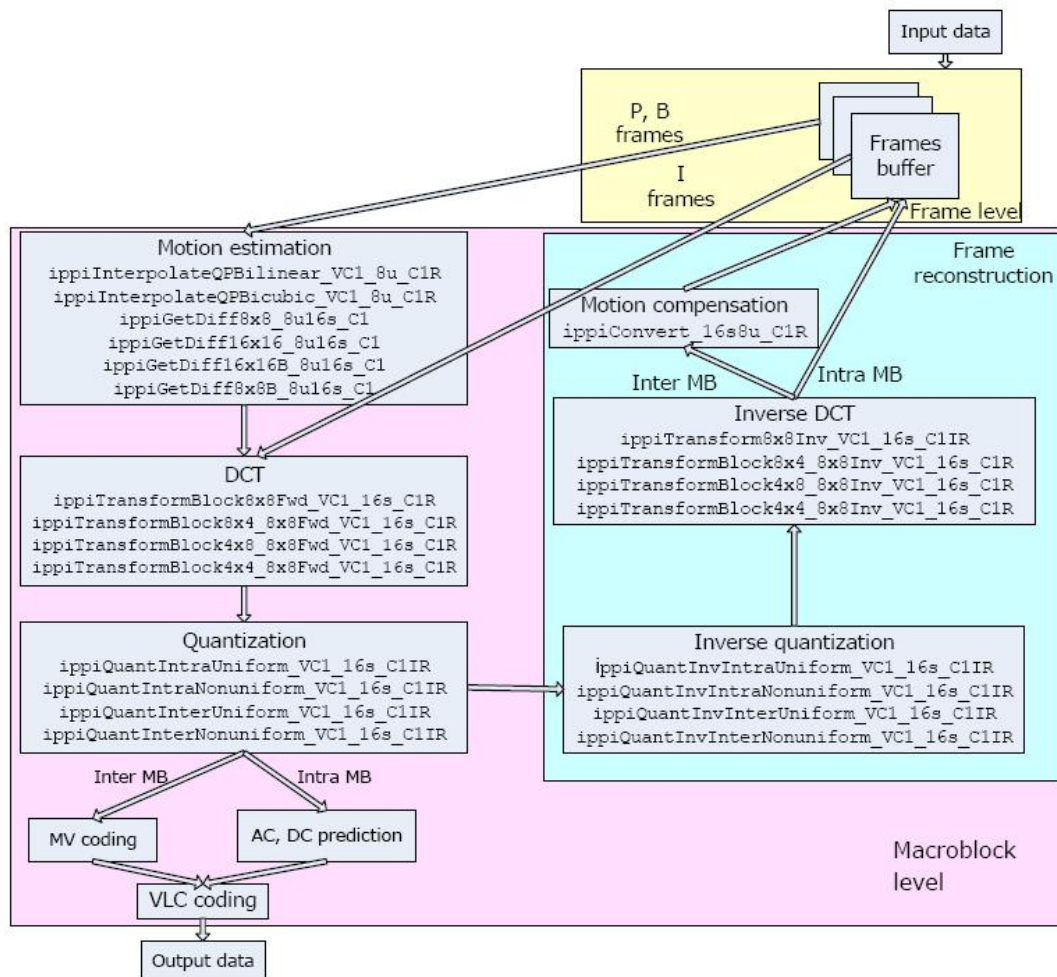
`ippStsOutOfRangeErr`

Indicates an error condition if *rangeMapParam* is out of the range [0, 7].

## VC-1 Encoder

See Figure "VC-1 Encoding Scheme" for VC-1 encoding scheme.

### VC-1 Encoding Scheme



### VC1 Video Encoder Functions

#### Function Short Name

#### Description

#### Forward Transform

`Transform8x8Fwd_VC1`, `Transform4x8Fwd_VC1`, `Transform8x4Fwd_VC1`, `Transform4x4Fwd_VC1`

Perform forward transform of blocks of respective size according to VC-1 standard.

#### Quantization

Function Short Name	Description
<code>QuantIntraUniform_VC1</code> , <code>QuantIntraNonuniform_VC1</code>	Perform uniform/nonuniform quantization of 8X8 intra blocks.
<code>QuantInterUniform_VC1</code> , <code>QuantInterNonuniform_VC1</code>	Perform uniform/nonuniform quantization of inter blocks.

## Forward Transform

### Transform8x8Fwd\_VC1, Transform4x8Fwd\_VC1, Transform8x4Fwd\_VC1, Transform4x4Fwd\_VC1

*Perform forward transform of blocks of respective sizes according to VC-1 standard.*

#### Syntax

```

IppStatus ippiTransform8x8Fwd_VC1_16s_C1IR(Ipp16s* pSrcDst, Ipp32s srcDstStep);
IppStatus ippiTransform4x8Fwd_VC1_16s_C1IR(Ipp16s* pSrcDst, Ipp32s srcDstStep);
IppStatus ippiTransform8x4Fwd_VC1_16s_C1IR(Ipp16s* pSrcDst, Ipp32s srcDstStep);
IppStatus ippiTransform4x4Fwd_VC1_16s_C1IR(Ipp16s* pSrcDst, Ipp32s srcDstStep);
IppStatus ippiTransform8x8Fwd_VC1_16s_C1R(const Ipp16s* pSrc, Ipp32s srcStep, Ipp16s* pDst, Ipp32s dstStep);
IppStatus ippiTransform4x8Fwd_VC1_16s_C1R(const Ipp16s* pSrc, Ipp32s srcStep, Ipp16s* pDst, Ipp32s dstStep);
IppStatus ippiTransform8x4Fwd_VC1_16s_C1R(const Ipp16s* pSrc, Ipp32s srcStep, Ipp16s* pDst, Ipp32s dstStep);
IppStatus ippiTransform4x4Fwd_VC1_16s_C1R(const Ipp16s* pSrc, Ipp32s srcStep, Ipp16s* pDst, Ipp32s dstStep);

```

#### Parameters

<code>pSrcDst</code>	Pointer to the source and destination block. All samples of the source block should be in range [-512, 511). After transforming all samples of the destination block are in range [-2048; 2047).
<code>pSrc</code>	Pointer to the source block. All samples of the source block should be in range [-512, 511).
<code>pDst</code>	Pointer to the destination 8x8 block. After transforming all samples of the destination block are in range [-2048; 2047).
<code>srcStep</code>	Distance in bytes between starts of the consecutive lines in the source block.
<code>dstStep</code>	Distance in bytes between starts of the consecutive lines in the destination block.

## Description

The functions `ippiTransform8x8Fwd_VC1_16s`, `ippiTransform4x8Fwd_VC1_16s`, `ippiTransform8x4Fwd_VC1_16s`, and `ippiTransform4x4Fwd_VC1_16s` are declared in the `ippvc.h` file. These functions perform forward transform in accordance with Annex A: *Transform Specification* of [SMPTE421M].

`ippiTransform8x8Fwd_VC1_16s_C1IR` and `ippiTransform8x8Fwd_VC1_16s_C1R` perform transform of 8x8 inter block with 8x8 transformation type or 8x8 intra block.

`ippiTransform4x8Fwd_VC1_16s_C1IR` and `ippiTransform4x8Fwd_VC1_16s_C1R` perform transform of 4x8 block into 8x8 inter block with 4x8 transformation type.

`ippiTransform8x4Fwd_VC1_16s_C1IR` and `ippiTransform8x4Fwd_VC1_16s_C1R` perform transform of 8x4 subblock into 8x8 inter block with 8x4 transformation type.

`ippiTransform4x4Fwd_VC1_16s_C1IR` and `ippiTransform4x4Fwd_VC1_16s_C1R` perform transform of 4x4 block into 8x8 inter block with 4x4 transformation type.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition when at least one of the specified pointers is NULL.

## Example

The code example below shows how to use `ippiTransform8x8Fwd_VC1_16s_C1IR` function.

```
{
Ipp16s* pSrcDstData = pointer_on_source_and_destination_block;
/*
All samples of source block should be in the range [-255;255]
After transforming all samples of destination block will be
in the range [-2048; 2047)
*/
Ipp32s srcDstStep = source_and_destination_step; /* Step of the pointer
pSrcDstData (source and destination array) in bytes */
IppStatus result;

/*
ippStsNoErr          Indicates no error
ippStsNullPtrErr     Indicates an error when pSrcDstData is NULL
*/

result = ippiTransform8x8Fwd_VC1_16s_C1IR( pSrcDstData, srcDstStep );
/*
Performs transform of 8x8 inter block with 8x8 transformation type or 8x8 intra
block in accordance with Annex A "Transform Specification" of SMPTE 421M
*/
}
```

The code example below shows how to use `ippiTransform8x8Fwd_VC1_16s_C1R` function.

```
{
Ipp16s* pSrcData = pointer_on_source_block;
/*
All samples of source block should be in the range [-255;255]
*/
Ipp16s* pDstData = pointer_on_destination_block;
```



```

/*
After transforming all samples of destination block will be in the
range [-2048; 2047)
*/
Ipp32s srcStep = source_step; /* Step of the pointer pSrcData (source array) in
bytes */
Ipp32s dstStep = destination_step; /* Step of the pointer pDstData (destination
array) in bytes */

IppStatus result;

/*
ippStsNoErr          Indicates no error
ippStsNullPtrErr     Indicates an error when pSrcData or pDstData is NULL
*/

result = ippiTransform8x8Fwd_VC1_16s_C1R( pSrcData, srcStep, pDstData,
dstStep );
/*
Performs transform of 8x8 inter block with 8x8 transformation type or 8x8 intra
block in accordance with Annex A "Transform Specification" of SMPTE 421M
*/

}

```

The code example below shows how to use `ippiTransform4x8Fwd_VC1_16s_C1IR` function.

```

{
Ipp16s* pSrcDstData = pointer_on_source_and_destination_block;
/*
All samples of source block should be in the range [-255;255]
After transforming all samples of destination block will be in the range
[-2048; 2047)
*/
Ipp32s srcDstStep = source_and_destination_step; /* Step of the pointer
pSrcDstData (source and destination array) in bytes */
IppStatus result;

/*
ippStsNoErr          Indicates no error
ippStsNullPtrErr     Indicates an error when pSrcDstData is NULL
*/

result = ippiTransform4x8Fwd_VC1_16s_C1IR(pSrcDstData, srcDstStep);
result = ippiTransform4x8Fwd_VC1_16s_C1IR(pSrcDstData+4, srcDstStep);

/*
Performs transform of 4x8 subblock in 8x8 inter block with 4x8 transformation
type in accordance with Annex A "Transform Specification" of SMPTE 421M
*/

}

```

The code example below shows how to use `ippiTransform4x8Fwd_VC1_16s_C1R` function.

```

{
Ipp16s* pSrcData = pointer_on_source_block;
/*
All samples of source block should be in the range [-255;255]
*/
Ipp16s* pDstData = pointer_on_destination_block;
/*
After transforming all samples of destination block will be

```

```

in the range [-2048; 2047)
*/
Ipp32s srcStep = source_step; /* Step of the pointer pSrcData (source array)
in bytes */
Ipp32s dstStep = destination_step; /* Step of the pointer pDstData (destination
array) in bytes */

IppStatus result;

/*
ippStsNoErr          Indicates no error
ippStsNullPtrErr     Indicates an error when pSrcData or pDstData is NULL
*/

result = ippiTransform4x8Fwd_VC1_16s_C1R( pSrcData, srcStep, pDstData, dstStep );
result = ippiTransform4x8Fwd_VC1_16s_C1R( pSrcData+4, srcStep, pDstData+4,
dstStep );

/*
Performs transform of 4x8 subblock in 8x8 inter block with 4x8 transformation
type in accordance with Annex A "Transform Specification" of SMPTE 421M
*/

}

```

The code example below shows how to use `ippiTransform8x4Fwd_VC1_16s_C1IR` function.

```

{
Ipp16s* pSrcDstData = pointer_on_source_and_destination_block;
/*
All samples of source block should be in the range [-255;255]
After transforming all samples of destination block will be in the range
[-2048; 2047)
*/
Ipp32s srcDstStep = source_and_destination_step; /* Step of the pointer
pSrcDstData (source and destination array) in bytes */
IppStatus result;

/*
ippStsNoErr          Indicates no error
ippStsNullPtrErr     Indicates an error when pSrcDstData is NULL
*/

result = ippiTransform8x4Fwd_VC1_16s_C1IR( pSrcDstData, srcDstStep );
result = ippiTransform8x4Fwd_VC1_16s_C1IR( pSrcDstData+(srcDstStep*2),
srcDstStep );

/*
Performs transform of 8x4 subblock in 8x8 inter block with 8x4 transformation
type in accordance with Annex A "Transform Specification" of SMPTE 421M
*/

}

```

The code example below shows how to use `ippiTransform8x4Fwd_VC1_16s_C1R` function.

```

{
Ipp16s* pSrcData = pointer_on_source_block;
/*
All samples of source block should be in the range [-255;255]
*/
Ipp16s* pDstData = pointer_on_destination_block;
/*

```

```

After transforming all samples of destination block will be in the range
[-2048; 2047)
*/
Ipp32s srcStep = source_step; /* Step of the pointer pSrcData (source array)
in bytes */
Ipp32s dstStep = destination_step; /* Step of the pointer pDstData (destination
array) in bytes */

IppStatus result;

/*
ippStsNoErr          Indicates no error
ippStsNullPtrErr     Indicates an error when pSrcData or pDstData is NULL
*/

result = ippiTransform8x4Fwd_VC1_16s_C1R( pSrcData, srcStep, pDstData,
dstStep );
result = ippiTransform8x4Fwd_VC1_16s_C1R( pSrcData+(srcStep*2), srcStep,
pDstData+(dstStep*2), dstStep );

/*
Performs transform of 8x4 subblock in 8x8 inter block with 8x4 transformation
type in accordance with Annex A "Transform Specification" of SMPTE 421M
*/

}

```

The code example below shows how to use `ippiTransform4x4Fwd_VC1_16s_C1IR` function.

```

{
Ipp16s* pSrcDstData = pointer_on_source_and_destination_block;
/*
All samples of source block should be in the range [-255;255]
After transforming all samples of destination block will be
in the range [-2048; 2047)
*/
Ipp32s srcDstStep = source_and_destination_step; /* Step of the pointer
pSrcDstData (source and destination array) in bytes */
IppStatus result;

/*
ippStsNoErr          Indicates no error
ippStsNullPtrErr     Indicates an error when pSrcDstData is NULL
*/

result = ippiTransform4x4Fwd_VC1_16s_C1IR( pSrcDstData, srcDstStep );
result = ippiTransform4x4Fwd_VC1_16s_C1IR( pSrcDstData+4, srcDstStep );
result = ippiTransform4x4Fwd_VC1_16s_C1IR( pSrcDstData+(srcDstStep*2),
srcDstStep );
result = ippiTransform4x4Fwd_VC1_16s_C1IR( pSrcDstData+(srcDstStep*2)+4,
srcDstStep );

/*
Performs transform of 4x4 subblock in 8x8 inter block with 4x4 transformation
type in accordance with Annex A "Transform Specification" of SMPTE 421M
*/

}

```

The code example below shows how to use `ippiTransform4x4Fwd_VC1_16s_C1IR` function.

```
{
Ipp16s* pSrcData = pointer_on_source_block;
/*
All samples of source block should be in the range [-255;255]
*/
Ipp16s* pDstData = pointer_on_destination_block;
/*
After transforming all samples of destination block will be
in the range [-2048; 2047)
*/
Ipp32s srcStep = source_step; /* Step of the pointer pSrcData (source array)
in bytes*/
Ipp32s dstStep = destination_step; /* Step of the pointer pDstData (destination
array) in bytes*/

IppStatus result;

/*
ippStsNoErr          Indicates no error
ippStsNullPtrErr     Indicates an error when pSrcData or pDstData is NULL
*/

result = ippiTransform4x4Fwd_VC1_16s_C1R( pSrcData, srcStep, pDstData,
dstStep );

result = ippiTransform4x4Fwd_VC1_16s_C1R( pSrcData+4, srcStep, pDstData+4,
dstStep );
result = ippiTransform4x4Fwd_VC1_16s_C1R( pSrcData+(srcStep*2), srcStep,
pDstData+(dstStep*2), dstStep );
result = ippiTransform4x4Fwd_VC1_16s_C1R( pSrcData+(srcStep*2)+4, srcStep,
pDstData+(dstStep*2)+4, dstStep );

/*
Performs transform of 4x4 subblock in 8x8 inter block with 4x4 transformation
type in accordance with Annex A "Transform Specification" of SMPTE 421M
*/
}
```

## Quantization

### QuantIntraUniform\_VC1, QuantIntraNonuniform\_VC1

*Perform uniform/nonuniform quantization of 8X8 intra blocks.*

---

#### Syntax

```
IppStatus ippiQuantIntraUniform_VC1_16s_C1IR( Ipp16s* pSrcDst, Ipp32s srcDstStep, Ipp32s
doubleQuant );

IppStatus ippiQuantIntraNonuniform_VC1_16s_C1IR( Ipp16s* pSrcDst, Ipp32s srcDstStep, Ipp32s
doubleQuant );
```

## Parameters

<i>pSrcDst</i>	Pointer to the source and destination block.
<i>srcDstStep</i>	Distance in bytes between starts of the consecutive lines in the source and destination blocks.
<i>doubleQuant</i>	Quant coefficient within the range [2, 62].

## Description

The functions `ippiQuantIntraUniform_VC1_16s_C1IR` and `ippiQuantIntraNonuniform_VC1_16s_C1IR` are declared in the `ippvc.h` file. The functions perform quantization of all coefficients in intra block (except DC) according to 8.1.2.8 of [ SMPTE 421M ].

In the case of uniform quantization:

```
a[i]=a[i]/quant_coef;
```

In the case of nonuniform quantization:

```
a[i]=(a[i]-sign(a[i]))*(quant_coef >> 1))/quant_coef;
```

where  $a[i]$  is the AC coefficient of the intra block, and `quant_coef` is the quant coefficient.

The *doubleQuant* parameter should be equal to  $2 * \text{MQANT} + \text{HALFQP}$ , where `MQANT` and `HALFQP` are defined in the VC-1 standard. `MQANT` should be within the range [1;31], `HALFQP` should be within the range [0,1] if `MQANT`  $\leq 8$  and should be equal to 0, if `MQANT`  $> 8$ .

## Example

The code example below shows how to use `ippiQuantIntraUniform_VC1_16s_C1IR` function.

```
{
Ipp16s* pSrcDstData = pointer_on_source_and_destination_block;
Ipp32s doubleQuant = quant_coefficient; /* It should be an even number in the
range [2,62]; */
Ipp32s srcDstStep = source_and_destination_step; /* Step of the pointer
pSrcDstData (source and destination array) in bytes */
IppStatus result;

/*
ippStsNoErr          Indicates no error
ippStsNullPtrErr     Indicates an error when pSrcDstData is NULL
ippStsOutOfRangeErr  Indicates an error when quantizer doubleQuant is less than 2
or higher than 62
*/

result = ippiQuantIntraUniform_VC1_16s_C1IR ( pSrcDstData, srcDstStep,
doubleQuant );

/* Performs uniform quantization process for all coefficients in the intra
block (except DC) according to 8.1.2.8 "Inverse AC Coefficient Quantization"
specification of SMPTE 421M
*/
}
```

## Example

The code example below shows how to use `ippiQuantIntraNonuniform_VC1_16s_C1IR` function.

```

{
    Ipp16s* pSrcDstData = pointer_on_source_and_destination_block;
    Ipp32s doubleQuant = quant_coefficient; /* It should be an even number in the
    range [2,62];*/
    Ipp32s srcDstStep = source_and_destination_step; /* Step of the pointer
    pSrcDstData (source and destination array) in bytes */
    IppStatus result;

    /*
    ippStsNoErr          Indicates no error
    ippStsNullPtrErr     Indicates an error when pSrcDstData is NULL
    ippStsOutOfRangeErr  Indicates an error when quantizer doubleQuant is less than 2
    or higher than 62
    */

    result = ippiQuantIntraNonuniform_VC1_16s_C1IR( pSrcDstData, srcDstStep,
    doubleQuant );

    /* Performs nonuniform quantization process for all coefficients in the intra
    block (except DC) according to 8.1.2.8 "Inverse AC Coefficient Quantization"
    specification of SMPTE 421M
    */
}

```

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition when <i>pSrcDst</i> is NULL.
<code>ippStsDivByZeroErr</code>	Indicates an error condition when the <i>doubleQuant</i> quantizer is equal to zero.

## QuantInterUniform\_VC1, QuantInterNonuniform\_VC1

Perform uniform/nonuniform quantization of inter blocks.

### Syntax

```

IppStatus ippiQuantInterUniform_VC1_16s_C1IR( Ipp16s* pSrcDst, Ipp32s srcDstStep, Ipp32s
doubleQuant, IppiSize roiSize );

IppStatus ippiQuantInterNonuniform_VC1_16s_C1IR( Ipp16s* pSrcDst, Ipp32s srcDstStep, Ipp32s
doubleQuant, IppiSize roiSize );

```

### Parameters

<i>pSrcDst</i>	Pointer to the source and destination block.
<i>srcDstStep</i>	Distance in bytes between starts of the consecutive lines in the source and destination blocks.
<i>doubleQuant</i>	Quant coefficient; should be an even number within the range [2, 62].
<i>roiSize</i>	Inter block size; should be 8x8, 8x4, 4x8 or 4x4.

## Description

The functions `ippiQuantInterUniform_VC1_16s_C1IR` and `ippiQuantInterNonuniform_VC1_16s_C1IR` are declared in the `ippvc.h` file. The functions perform quantization of inter blocks according to 8.1.2.8 of [SMPT421M].

In the case of uniform quantization:

```
a[i]=a[i]/quant_coef;
```

In the case of nonuniform quantization:

```
a[i]=(a[i]-sign(a[i])*(quant_coef >> 1))/quant_coef;
```

where  $a[i]$  is the coefficient of the inter block, and `quant_coef` is the quant coefficient.

The `doubleQuant` parameter should be equal to  $2 * \text{MQANT} + \text{HALFQP}$ , where `MQANT` and `HALFQP` are defined in the VC-1 standard. `MQANT` should be within the range  $[1;31]$ , `HALFQP` should be within the range  $[0,1]$  if `MQANT`  $\leq 8$  and should be equal to 0, if `MQANT`  $> 8$ .

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition when <code>pSrcDst</code> is NULL.
<code>ippStsDivByZeroErr</code>	Indicates an error condition when the <code>doubleQuant</code> quantizer is equal to zero.

## Example

The code example below shows how to use `ippiQuantInterUniform_VC1_16s_C1IR` function.

```
{
Ipp16s* pSrcDstData = pointer_on_source_and_destination_block;
Ipp32s doubleQuant = quant_coefficient; /* It should be an even number in the
range [2,62]; */
Ipp32s srcDstStep = source_and_destination_step; /* Step of the pointer
pSrcDstData (source and destination array) in bytes */
IppSize roiSize = inter_block_size;
/* It should be:
8x8 (roiSize.width=8, roiSize.height=8)
4x8 (roiSize.width=4, roiSize.height=8)
8x4 (roiSize.width=8, roiSize.height=4)
4x4 (roiSize.width=4, roiSize.height=4)
*/
IppStatus result;

/*
ippStsNoErr      Indicates no error
ippStsNullPtrErr Indicates an error when pSrcDstData is NULL
ippStsOutOfRangeErr Indicates an error when quantizer doubleQuant is less than 2
or higher than 62
ippStsSizeErr    Indicates an error when roiSize.height is not equal 4 or 8
or roiSize.width is not equal 4 or 8
*/

result = ippiQuantInterUniform_VC1_16s_C1IR( pSrcDstData, srcDstStep,
doubleQuant );

/* Performs uniform quantization process for inter block according to 8.1.2.8
```

```
"Inverse AC Coefficient Quantization" specification of SMPTE 421M
*/
}
```

The code example below shows how to use `ippiQuantInterNonuniform_VC1_16s_C1IR` function.

```
{
Ipp16s* pSrcDstData = pointer_on_source_and_destination_block;
Ipp32s doubleQuant = quant_coefficient; /* It should be an even number in the
range [2,62]; */
Ipp32s srcDstStep = source_and_destination_step; /* Step of the pointer
pSrcDstData (source and destination array) in bytes */
IppSize roiSize = inter_block_size;
/* It should be:
8x8 (roiSize.width=8, roiSize.height=8)
4x8 (roiSize.width=4, roiSize.height=8)
8x4 (roiSize.width=8, roiSize.height=4)
4x4 (roiSize.width=4, roiSize.height=4)
*/
IppStatus result;
/*
ippStsNoErr      Indicates no error
ippStsNullPtrErr Indicates an error when pSrcDstData is NULL
ippStsOutOfRangeErr Indicates an error when quantizer doubleQuant is less than 2
or higher than 62
ippStsSizeErr    Indicates an error when roiSize.height is not equal 4 or 8
or roiSize.width is not equal 4 or 8
*/

result = ippiQuantInterNonuniform_VC1_16s_C1IR( pSrcDstData, srcDstStep,
doubleQuant );

/* Performs nonuniform quantization process for inter block according to
8.1.2.8 "Inverse AC Coefficient Quantization" specification of SMPTE 421M
*/
}
```



# Handling of Special Cases

## A

Some mathematical functions implemented in Intel IPP are not defined for all possible argument values. This appendix describes how the corresponding Intel IPP image processing functions handle situations when their input arguments fall outside the range of function definition or may lead to ambiguously determined output results.

Table A-1 below summarizes these special cases for different functions and lists result values together with status codes returned by the functions. The status codes ending with `Err` (except for the `ippStsNoErr` status) indicate an error. When an error occurs, the function execution is interrupted. All other status codes indicate that the input argument is outside the range, but the function execution is continued with the corresponding result value.

### Special Cases for Intel IPP Image Processing Functions

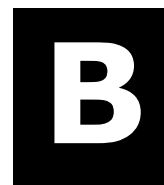
Function Base Name	Data Type	Case Description	Result Value	Status Code
<code>ippiSqrt</code>	16s	Sqrt ( $x < 0$ )	0	<code>ippStsSqrtNegArg</code>
	32f	Sqrt ( $x < 0$ )	<code>NAN_32F</code>	<code>ippStsSqrtNegArg</code>
<code>ippiDiv</code>	8u	Div (0/0)	0	<code>ippStsDivByZero</code>
		Div ( $x/0$ )	<code>IPP_MAX_8U</code>	<code>ippStsDivByZero</code>
	16s	Div (0/0)	0	<code>ippStsDivByZero</code>
		Div ( $x/0$ ), $x > 0$	<code>IPP_MAX_16S</code>	<code>ippStsDivByZero</code>
		Div ( $x/0$ ), $x < 0$	<code>IPP_MIN_16S</code>	<code>ippStsDivByZero</code>
	32f	Div (0/0)	<code>NAN_32F</code>	<code>ippStsDivByZero</code>
		Div ( $x/0$ ), $x > 0$	<code>INF_32F</code>	<code>ippStsDivByZero</code>
		Div ( $x/0$ ), $x < 0$	<code>INF_NEG_32F</code>	<code>ippStsDivByZero</code>
	16sc	Div (0/0)	0	<code>ippStsDivByZero</code>
		Div ( $x/0$ ),	0	<code>ippStsDivByZero</code>
	32sc	Div (0/0)	0	<code>ippStsDivByZero</code>
		Div ( $x/0$ ), $x > 0$	<code>IPP_MAX_32S</code>	<code>ippStsDivByZero</code>
			<code>IPP_MIN_32S</code>	<code>ippStsDivByZero</code>

Function Base Name	Data Type	Case Description	Result Value	Status Code
	32fc	Div (x/0), x<0		
			NAN_32F	
		Div (0/0)	INF_32F	
		Div (x/0), x>0	INF_NEG_32F	
		Div (x/0), x<0		
ippiDivC	all	Div(x/const), const=0	-	ippStsDivByZeroErr
ippiLn	8u	Ln (0)	0	ippStsLnZeroArg
	16s	Ln (0)	IPP_MIN_16S	ippStsLnZeroArg
		Ln (x<0)	IPP_MIN_16S	ippStsLnNegArg
	ippiExp	32f	Ln (x<0)	NAN_32F
Ln(x<IPP_MINABS_32F)			INF_NEG_32F	ippStsLnZeroArg
8u		overflow	IPP_MAX_8U	ippStsNoErr
16s		overflow	IPP_MAX_16S	ippStsNoErr
32f		overflow	INF_32F	ippStsNoErr

Here  $x$  denotes an input value. For the definition of the constants used, see [Image Data Types and Ranges](#) in chapter 2.

Note that flavors of the same math function operating on different data types may produce different results for the equal argument values. However, for a given function and a fixed data type, handling of special cases is the same for all function flavors that have different [descriptors](#) in their names. For example, logarithm function `ippiLn` operating on 16s data treats zero argument values in the same way for all its flavors `ippiLn_16s_C1RSfs`, `ippiLn_16s_C3RSfs`, `ippiLn_16s_C1IRSfs`, and `ippiLn_16s_C3IRSfs`.

# Interpolation in Image Geometric Transform Functions



This appendix describes the interpolation algorithms used in the geometric transformation functions of Intel IPP. For more information about each of the geometric transform functions, see [Chapter 12](#).

## Overview of Interpolation Modes

In geometric transformations, the grid of input image pixels is not necessarily mapped onto the grid of pixels in the output image. Therefore, to compute the pixel intensities in the output image, the geometric transform functions need to interpolate the intensity values of several input pixels that are mapped to a certain neighborhood of the output pixel.

Geometric transformations can use various interpolation algorithms. When calling the geometric transform functions of the Intel IPP, the application code specifies the interpolation mode (that is, the type of interpolation algorithm) by using the parameter *interpolation*. The library supports the following main interpolation modes:

- nearest neighbor interpolation (*interpolation* = `IPPI_INTER_NN`)
- linear interpolation (*interpolation* = `IPPI_INTER_LINEAR`)
- cubic interpolation (*interpolation* = `IPPI_INTER_CUBIC`)
- supersampling (*interpolation* = `IPPI_INTER_SUPER`)
- interpolation with Lanczos window function (*interpolation* = `IPPI_INTER_LANCZOS`).
- interpolation with [two-parameter cubic filters](#) (*interpolation* can be set to one of the `IPPI_INTER_CUBIC2P_BSPLINE`, `IPPI_INTER_CUBIC2P_CATMULLROM`, `IPPI_INTER_CUBIC2P_B05C03`).

For certain functions, the above main interpolation modes can be combined with additional *smoothing of edges* to which the borders of the original image are transformed. To use this option, set the parameter *interpolation* to the bitwise OR of the edge smoothing flag and the desired interpolation mode. For example, in order to rotate an image with cubic interpolation and smooth the rotated image edges, pass the following value to `ippiRotate()`:

```
interpolation = IPPI_INTER_CUBIC | IPPI_SMOOTH_EDGE.
```

Interpolation with edge smoothing option can be used only in those geometric transform functions where this option is explicitly listed in the parameters definition section.

For the function [ResizeSqrPixel](#) the listed above modes can be combined with the additional *antialiasing* option to remove various artefacts of image compression. To use this feature, set the parameter *interpolation* to the bitwise OR of `IPPI_ANTIALIASING` and the desired interpolation mode. Antialiasing does not work with supersampling and nearest neighbor interpolation modes.

Table B-1 lists the supported interpolation modes for the main geometric transform functions that use interpolation.

## Interpolation Modes Supported by Image Geometric Transform Functions

Function Base Name	NN	Lin	Cub	CR	La3	SS	AA	ES
ResizeSqrPixel	x	x	x	x *)	x	x	x	x **)
Remap	x	x	x	x	x	n/a	n/a	x
Rotate	x	x	x	x		n/a	n/a	x
RotateCenter	x	x	x	x		n/a	n/a	x
Shear	x	x	x	x		n/a	n/a	x
WarpAffine	x	x	x	x		n/a	n/a	x
WarpAffineBack	x	x	x			n/a	n/a	
WarpAffineQuad	x	x	x			n/a	n/a	x
WarpPerspective	x	x	x			n/a	n/a	x
WarpPerspective-Back	x	x	x			n/a	n/a	
WarpPerspective-Quad	x	x	x			n/a	n/a	x
WarpBilinear	x	x	x			n/a	n/a	x
WarpBilinearBack	x	x	x			n/a	n/a	
WarpBilinearQuad	x	x	x			n/a	n/a	x

\*) The function `ippiResizeSqrPixel` additionally supports two variants of the interpolation with **two-parameter cubic filters**.

\*\*) The function `ippiResizeSqrPixel` supports two variants of the edge smoothing (see function description).

Here **NN** - nearest neighbor interpolation, **Lin** - linear interpolation, **Cub** - cubic interpolation, **CR** - Catmull-Rom spline, **La3** - interpolation with Lanczos window, **SS** - super sampling interpolation, **AA** - antialiasing, **ES** - edge smoothing.

The sections that follow provide more details on each interpolation mode.

## Mathematical Notation

In this appendix the following notation is used:

$(x_D, y_D)$	pixel coordinates in the destination image (integer values);
$(x_S, y_S)$	the computed coordinates of a point in the source image that is mapped exactly to $(x_D, y_D)$ ;
$S(x, y)$	pixel value (intensity) in the source image;
$D(x, y)$	pixel value (intensity) in the destination image.

## Nearest Neighbor Interpolation

This is the fastest and least accurate interpolation mode. The pixel value in the destination image is set to the value of the source image pixel closest to the point

$$(x_S, y_S) : D(x_D, y_D) = S(\text{round}(x_S), \text{round}(y_S)).$$

To use the nearest neighbor interpolation, set the parameter *interpolation* to `IPPI_INTER_NN`.

## Linear Interpolation

This is the fastest and least accurate interpolation mode. The pixel value in the destination image is set to the value of the source image pixel closest to the point

$$(x_S, y_S) : D(x_D, y_D) = S(\text{round}(x_S), \text{round}(y_S)).$$

The linear interpolation is slower but more accurate than the nearest neighbor interpolation. On the other hand, it is faster but less accurate than cubic interpolation. The linear interpolation algorithm uses source image intensities at the four pixels  $(x_{S0}, y_{S0})$ ,  $(x_{S1}, y_{S0})$ ,  $(x_{S0}, y_{S1})$ ,  $(x_{S1}, y_{S1})$  that are closest to  $(x_S, y_S)$  in the source image:

$$x_{S0} = \text{int}(x_S), x_{S1} = x_{S0} + 1, y_{S0} = \text{int}(y_S), y_{S1} = y_{S0} + 1.$$

First, the intensity values are interpolated along the x-axis to produce two intermediate results  $I_0$  and  $I_1$  (see Figure B-1):

$$I_0 = S(x_S, y_{S0}) = S(x_{S0}, y_{S0}) * (x_{S1} - x_S) + S(x_{S1}, y_{S0}) * (x_S - x_{S0})$$

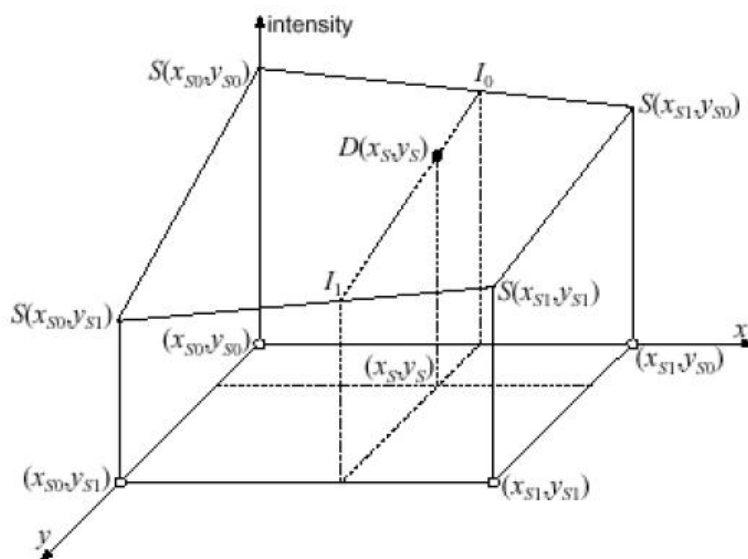
$$I_1 = S(x_S, y_{S1}) = S(x_{S0}, y_{S1}) * (x_{S1} - x_S) + S(x_{S1}, y_{S1}) * (x_S - x_{S0}).$$

Then, the sought-for intensity  $D(x_D, y_D)$  is computed by interpolating the intermediate values  $I_0$  and  $I_1$  along the y-axis:

$$D(x_D, y_D) = I_0 * (y_{S1} - y_S) + I_1 * (y_S - y_{S0}).$$

To use the linear interpolation, set the parameter *interpolation* to IPPI\_INTER\_LINEAR. For images with 8-bit unsigned color channels, the functions `ippiWarpAffine`, `ippiRotate`, and `ippiShear` compute the coordinates  $(x_S, y_S)$  with the accuracy  $2^{-16} = 1/65536$ . For images with 16-bit unsigned color channels, these functions compute the coordinates with floating-point precision.

### Linear Interpolation



## Cubic Interpolation

The cubic interpolation algorithm (see Figure B-2) uses source image intensities at sixteen pixels in the neighborhood of the point  $(x_S, y_S)$  in the source image:

$$x_{S0} = \text{int}(x_S) - 1; x_{S1} = x_{S0} + 1; x_{S2} = x_{S0} + 2; x_{S3} = x_{S0} + 3;$$

$$y_{S0} = \text{int}(y_S) - 1; y_{S1} = y_{S0} + 1; y_{S2} = y_{S0} + 2; y_{S3} = y_{S0} + 3.$$

First, for each  $y_{Sk}$  the algorithm determines four cubic polynomials  $F_0(x)$ ,  $F_1(x)$ ,  $F_2(x)$ , and  $F_3(x)$ :

$$F_k(x) = a_k x^3 + b_k x^2 + c_k x + d_k, \quad 0 \leq k \leq 3$$

such that

$$F_k(x_{S0}) = S(x_{S0}, y_{Sk}); F_k(x_{S1}) = S(x_{S1}, y_{Sk}),$$

$$F_k(x_{S2}) = S(x_{S2}, y_{Sk}); F_k(x_{S3}) = S(x_{S3}, y_{Sk}).$$

In Figure B-2, these polynomials are shown by solid curves.

Then, the algorithm determines a cubic polynomial  $F_y(y)$  such that

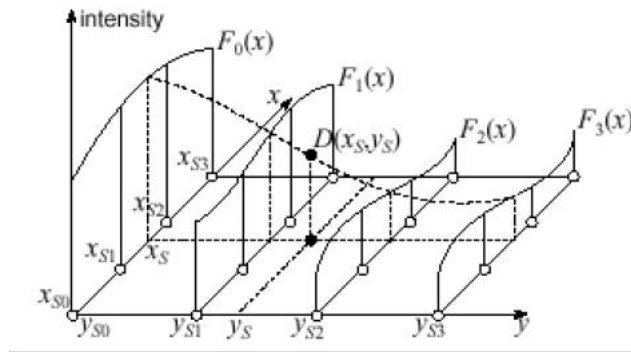
$$F_y(y_{S0}) = F_0(x_S), F_y(y_{S1}) = F_1(x_S), F_y(y_{S2}) = F_2(x_S), F_y(y_{S3}) = F_3(x_S).$$

The polynomial  $F_y(y)$  is represented by the dashed curve in Figure B-2.

Finally, the sought intensity  $D(x_D, y_D)$  is set to the value  $F_y(y_S)$ .

To use the linear interpolation, set the parameter *interpolation* to IPP\_INTER\_CUBIC. For images with 8-bit unsigned color channels, the functions `ippiWarpAffine`, `ippiRotate`, and `ippiShear` compute the coordinates  $(x_S, y_S)$  with the accuracy  $2^{-16} = 1/65536$ . For images with 16-bit unsigned color channels, these functions compute the coordinates with floating-point precision.

### Cubic Interpolation



## Super Sampling

If the destination image is much smaller than the source image, the above interpolation algorithms may skip some pixels in the source image (that is, these algorithms not necessarily use all source pixels when computing intensity of the destination pixels). To use all pixel values of the source image, the `ippiResize` and `ippiResizeCenter` functions support the super-sampling algorithm, which is free of the above drawback.

The super-sampling algorithm is as follows:

1. Divide the source image rectangular ROI (or the whole image, if there is no ROI) into equal rectangles, each rectangle corresponding to some pixel in the destination image. Note that each source pixel is represented by a 1x1 square.
2. Compute a weighted sum of source pixel values for all pixels that are in the rectangle or have a non-zero intersection with the rectangle. If a source pixel is fully contained in the rectangle, the value of that pixel is taken with weight 1. If the rectangle and the square of the source pixel have an intersection of area  $a < 1$ , that pixel's value is taken with weight  $a$ .

Figure B-3 shows the corresponding weight value for each source pixel intersecting with the rectangle.

3. To compute the pixel value in the destination image, divide this weighted sum by the ratio of the source and destination rectangle areas  $S_{Src}/S_{Dst} = 1/xFactor*yFactor$ .

Here *xFactor*, and *yFactor* are the parameters of the functions that specify the factors by which the *x* and *y* dimensions of the source image ROI are changed.

Note that supersampling interpolation can be used only for  $xFactor < 1$ , and  $yFactor < 1$ .

### Supersampling Weights

$\Delta_2$	$\Delta_2$	$\Delta_2$	$\Delta_2 \times \Delta_3$
1	1	1	$\Delta_3$
1	1	1	$\Delta_3$
$\Delta_1$	$\Delta_1$	$\Delta_1$	$\Delta_1 \times \Delta_3$

## Lanczos Interpolation

This method is based on the 3-lobed Lanczos window function as the interpolation function.

The interpolation algorithm uses source image intensities at 36 pixels in the neighborhood of the point  $(x_s, y_s)$  in the source image:

$$x_{s0} = \text{int}(x_s) - 2; x_{s1} = x_{s0} + 1; x_{s2} = x_{s0} + 2; x_{s3} = x_{s0} + 3; x_{s4} = x_{s0} + 4; x_{s5} = x_{s0} + 5;$$

$Y_{S0} = \text{int}(Y_S) - 2; Y_{S1} = Y_{S0} + 1; Y_{S2} = Y_{S0} + 2; Y_{S3} = Y_{S0} + 3; Y_{S4} = Y_{S0} + 4; Y_{S5} = Y_{S0} + 5;$

First, the intensity values are interpolated along the  $x$ -axis to produce six intermediate results  $I_0, I_1, \dots, I_5$ :

$$I_k = \sum_{i=0}^5 a_i \cdot s(x_{Si}, y_{Sk}), 0 \leq k \leq 5$$

Then the intensity  $D(x_D, y_D)$  is computed by interpolating the intermediate values  $I_k$  along the  $y$ -axis:

$$D(x_D, y_D) = \sum_{k=0}^5 b_k \cdot I_k$$

Here  $a_i$  and  $b_k$  are the coefficients defined as

$$a_i = L(x_S - x_{Si}), b_k = L(y_S - y_{Si}),$$

where  $L(x)$  is the Lanczos windowed sinc function:

$$L(x) = \text{sinc}(x) \cdot \text{Lanczos3}(x) = \begin{cases} \frac{\sin(\pi x)}{\pi x} \cdot \frac{\sin(\pi x / 3)}{(\pi x) / 3}, & 0 \leq |x| < 3 \\ 0, & 3 \leq |x| \end{cases}$$

To use this interpolation, set the parameter *interpolation* to `IPPI_INTER_LANCZOS`.

## Interpolation with Two-Parameter Cubic Filters

The two-parameter family of cubic filters have kernels of the form:

$$k(x) = \frac{1}{6} \begin{cases} (12 - 9B - 6C)|x|^3 + (-18 + 12B + 6C)|x|^2 + (6 - 2B) & |x| < 1 \\ (-B - 6C)|x|^3 + (6B + 30C)|x|^2 + (-12B - 48C)|x| + (8B + 24C) & 1 \leq |x| < 2 \\ 0 & \text{otherwise} \end{cases}$$

where  $B$  and  $C$  are two parameters; their variations give different approximation. The Intel IPP function `ippiResizeSqrPixel` uses three members of this family, specifically cubic  $B$ -spline ( $B=1, C=0$ ), Catmull-Rom spline ( $B=0, C=1/2$ ), and special filter with  $B=1/2, C=3/10$ .

These interpolation methods additionally filter the output to improve quality of an image.



# Bibliography

---

This bibliography provides a list of publications that might be helpful to you in using the image processing subset of Intel IPP. This list is not complete; it serves only as a starting point. The books [Rog85], [Rog90], and [Foley90] are good resources of information on image processing and computer graphics, with mathematical formulas and code examples.

- [Aka96] A.Akansu, M.Smith (editors). *Subband and Wavelet transform. Design and Applications*, Kluwer Academic Publishers, 1996.
- [AP922] *A Fast Precise Implementation of 8x8 Discrete Cosine Transform Using the Streaming SIMD Extensions and MMX™ Instructions*, Application Note AP922, Intel Corp. Order number 742474, 1999.
- [APMF] *Fast Algorithms for Median Filtering*, Application Note, Intel Corp. Document number 79835, 2001.
- [AVS] GB/T 200090.2-2006. China Standard. Information Technology. *Coding of Audio-Visual Objects - Part 2: Visual* (02/2006).
- [Bert01] M.Bertalmio, A.L.Bertozzi, G.Sapiro. *Navier-Stokes, Fluid Dynamics, and Image and Video Inpainting*. Proc. ICCV 2001, pp.1335-1362, 2001.
- [Bor86] G.Borgefors. *Distance Transformations in Digital Images*. Computer Vision, Graphics, and Image Processing 34, 1986.
- [Bou99] J-Y.Bouguet. *Pyramidal Implementation of the Lucas-Kanade Feature Tracker*. OpenCV Documentation, Microprocessor Research Lab, Intel Corporation, 1999.
- [Canny86] J. Canny. *A Computational Approach to Edge Detection*, IEEE Trans. on Pattern Analysis and Machine Intelligence 8(6), 1986.
- [Davis97] J.Davis and Bobick. *The Representation and Recognition of Action Using Temporal Templates*. MIT Media Lab Technical Report 402, 1997.
- [Davis99] J.Davis and G.Bradski. *Real-Time Motion Template Gradients Using Intel(R) Computer Vision Library*. IEEE ICCV'99 FRAME-RATE WORKSHOP, 1999.
- [DICOM] Digital Imaging and Communications in Medicine (DICOM), published by National Electrical Manufacturers Association, 2003.
- [Feig92] E. Feig and S. Winograd. *Fast Algorithms for the Discrete Cosine Transform*, IEEE Trans. Signal Processing, vol. 40, no. 9, pp. 2174-2193, Sep. 1992.
- [Felz04] P. Felzenszwalb, D. Hattenlocher. *Distance Transforms of Sampled Functions*. Cornell Computing and Information Science Technical Report TR2004-1663, September 2004.
- [Foley90] James D. Foley, Andries van Dam, Steven K. Feiner, and John F. Hughes. *Computer Graphics — Principles and Practice*, Second Edition. Addison Wesley, 1990.
- [Gon93] R.C. Gonzalez and R.E.Wood. *Digital Image Processing*. Prentice Hall, 1993.
- [Hir05] K. Hirakawa, T.W. Parks, *Adaptive Homogeneity-Directed Demosaicing Algorithm*, IEEE Trans. Image Processing, March, 2005.
- [IEC61834] IEC 61834. International Electrochemical Commission. *Specifications of Consumer-Use Digital VCRs using 6.3 mm magnetic tape (the "Blue Book" DV specification)*.

- [IEEE] *IEEE Standard Specifications for the Implementations of 8X8 Inverse Discrete Cosine Transform*, IEEE #1180 (1997).
- [IPL] *Intel Image Processing Library Reference Manual*. Intel Corp. Order number 663791, 1999.
- [ISO11172] ISOCD 11172. Information Technology. *Coding of Moving Pictures and Associated Audio for Digital Storage Media at up to about 1.5 Mbit/s* (1993).
- [ISO13818] ISO/IEC 13818. Information Technology. *Coding of Moving Pictures and Associated Audio Information*, (11/94).
- [ISO14496] International Standard ISO/IEC 14496-2. Information Technology. Coding of Audio-Visual Objects - Part 2: Visual.
- [ISO14496A] ISO/IEC 14496-2:1999/Amd.1:2000(E) . Information Technology. *Coding of Audio-Visual Objects*. Part2:Visual. Amendment 1: Visual Extensions (01/00).
- [ISO10918] International Standard ISO/IEC 10918-1, *Digital Compression and Coding of Continuous Tone Still Images*, Appendix A - *Requirements and guidelines*.
- [ISO15444] International Standard ISO/IEC 15444-1, *JPEG 2000 Image coding system*, part 1: Core coding system.
- [ISO29199] International Standard ISO/IEC 29199-2, *JPEG XR Image coding system*, part 2: Image coding specification. (2009-08-15)
- [ITUH261] ITU-T Recommendation H.261. *Line transmission of non-telephone signals. Video codec for audiovisual services at p x 64 kbits* (03/93).
- [ITUH263] ITU-T Recommendation H.263. Series H: AUDIOVISUAL AND MULTIMEDIA SYSTEMS. *Infrastructure of audiovisual services - Coding of moving video. Video coding for low bit rate communication* (02/98).
- [ITUH264] ITU-T Recommendation H.264. Series H: AUDIOVISUAL AND MULTIMEDIA SYSTEMS. *Infrastructure of audiovisual services - Coding of moving video. Advanced video coding for generic audiovisual services*. (ITU-T Rec. H.264 | ISO/IEC 14496-10:2005)(03/05).
- [ITUH264\_07] ITU-T Recommendation H.264. Series H: AUDIOVISUAL AND MULTIMEDIA SYSTEMS. *Infrastructure of audiovisual services - Coding of moving video. Advanced video coding for generic audiovisual services*. (ITU-T Rec. H.264 | ISO/IEC 14496-10:2008)(11/07).
- [ITU709] ITU-R Recommendation BT.709, *Basic Parameter Values for the HDTV Standard for the Studio and International Programme Exchange* [formerly CCIR Rec.709] ITU, Geneva, Switzerland, 1990.
- [Jae95] Jaehne, Bernd. *Digital Image Processing*, 3rd Edition, Springer-Verlag, Berlin, 1995.
- [Jae97] Jaehne, Bernd. *Practical Handbook on Image Processing for Scientific Applications*, CRC Press, New York, 1997.
- [Jack01] Jack, Keith. *Video Demystified: a Handbook for the Digital Engineer*, LLH Technology Publishing, 3rd Edition, 2001.
- [JVTG050] JVT-G050. *ITU-T Recommendation and Final Draft International Standard of Joint Video Specification* (ITU-T Rec. H.264 | ISO/IEC 14496-10 AVC) (03/03).
- [Kadew01] P.KadewTraKuPong, R.Bowden. *An Improved Adaptive Background Mixture Model for Real-Time Tracking with Shadow Detection*. Proc. 2nd European Workshop on Advanced Video-Based Surveillance Systems, 2001.
- [Lein02] R.Leinhart, J.Maydt. An Extended Set of Haar-like Features for Rapid Object Detection. IEEE ICIP, vol.1, pp. 900-903, Sep. 2002.
- [Li03] L.Li, W.Huang, I.Gu, Q.Tian. *Foreground Object Detection from Videos Containing Complex Background*. Proc.ACM Multimedia Conference, Berkley, 2003.

- 
- [Lim90] Jae S.Lim. *Two-Dimensional Signal and Image Processing*, Prentice Hall, Englewood Cliffs, NJ, 1990.
- [Lotufo00] R.Lotufo, A.Falcão. *The Ordered Queue and the Optimality of the Watershed Algorithm*. Mathematical Morphology and its Applications to Image and Signal Processing, vol.18, pp.341-350. Kluwer Academic Publishers. Palo Alto, USA, June 2000.
- [Lowe04] D.G.Lowe. *Distinctive Image Features from Scale-Invariant Keypoints*. International Journal of Computer Vision, vol.60, No.2, pp. 91-110, 2004.
- [Malvar03] H.S.Malvar, G.J.Sullivan. *Transform, Scaling & Color Space Impact of Professional Extensions*, ISO/IEC JTC/SC29/WG11 and ITU-T SG16 Q.6 Document JVT-H031, Geneva, May 2003.
- [Malvar03-1] H.S.Malvar, G.J.Sullivan. *YCoCg-R: A Color Space with RGB Reversibility and Low Dynamic Range*, Joint Video Team (JVT) of ISO/IEC MPEG & ITU-T VCEG, Document No.JVT-1014r3, July 2003.
- [Meyer92] F.Meyer. *Color Image Segmentation*, 4th International Conference on Image Processing and its Applications, p.4, Maastricht, April 1993.
- [Meyer94] F.Meyer. *Topographic Distance and Watershed Lines*. Signal Processing, No.38, pp.113-125, 1994.
- [Myler93] H.Myler, A.Weeks. *Computer Imaging Recipes in C*, Prentice Hall, 1993.
- [Otsu79] N. Otsu. *A Threshold Selection Method From Gray Level Histograms*. IEEE Transactions on Systems, Man, and Cybernetics, vol.9, No.1, January 1979, pp. 62-66.
- [Puetter05] R.C.Puetter, T.R.Gosnell, and Amos Yahil. *Digital Image Reconstruction: Deblurring and Denoising*, Annual Review of Astronomy and Astrophysics, 2005.
- [Randy97] Randy Crane. *A Simplified Approach to Image Processing*, Prentice Hall PTR, 1997.
- [Rao90]] K.R. Rao and P. Yip. *Discrete Cosine Transform. Algorithms, Advantages, Applications*. Academic Press, Inc, London, 1990.
- [Ric72] W.Richardson. *Bayesian-Based Iterative Method of Image Reconstruction*. Journal of the Optical Society of America, vol.62, No.1, January 1972.
- [Ritter96] G.Ritter, J.Wilson. *Computer Vision. Algorithms in Image Algebra*. CRC Press, 1996.
- [Rog85] David Rogers. *Procedural Elements for Computer Graphics*. McGraw-Hill, 1985.
- [Rog90] David Rogers and J.Alan Adams. *Mathematical Elements for Computer Graphics*. McGraw-Hill, 1990.
- [S3TC] *S3 Texture Compression*. <http://en.wikipedia.org/wiki/S3TC>
- [Sak98] T. Sakamoto, C. Nakanishi, and T. Hase, *Software pixel interpolation for digital still cameras suitable for a 32-bit MCU*, IEEE Trans. Consumer Electronics, vol. 44, No. 4, November 1998.
- [Serra82] J.Serra. *Image Analysis and Mathematical Morphology*. Academic Press, London, 1982.
- [Schu94] Dale A. Schumacher. *A comparison of digital halftoning techniques*, Graphic Gems III, Academic Press, 1994, pp. 57-71.
- [Sha98] Tom Shanley. *Pentium Pro and Pentium II System Architecture*. Addison-Wesley, 1998.
- [SMPTE314M] SMPTE 314M-2005 for Television - *Data Structure for DV-Based Audio, Data and Compressed Video - 25 and 50 Mb/s*. The Society of Motion Picture and Television Engineers (09/05).
- [SMPTE370M] SMPTE 370M-2002 for Television - *Data Structure for DV-Based Audio, Data and Compressed Video at 100 Mb/s, 1080/60i, 1080/50i, 720/60p*. The Society of Motion Picture and Television Engineers (07/02).

- [SMPTE370M-06] SMPTE 370M-2006 for Television - *Data Structure for DV-Based Audio, Data and Compressed Video at 100 Mb/s, 1080/60i, 1080/50i, 720/60p, 720/50p*. The Society of Motion Picture and Television Engineers (04/06).
- [SMPTE421M] SMPTE 421M. Final Draft SMPTE Standard - *VC-1 Compressed Video Bitstream Format and Decoding Process* (01/06).
- [Telea04] A.Telea. *An Image Inprinting Technique Based on the Fast Marching Method*. Journal of Graphic Tools, vol.9, No.1, ACM Press, 2004.
- [Tho91] Spencer W. Thomas and Rod G. Bogart. *Color dithering*, Graphic Gems II, Academic Press, 1991, pp. 72-77.
- [Ulichney93] R.Ulichney. *Digital halftoning*. MIT press, 1993.
- [Vincent91] L.Vincent, P.Soille. *Watershed in Digital Spaces: An Efficient Algorithm Based on Immersion Simulation*. IEEE Transactions of Pattern Analysis and Machine Intelligence, vol.3, No.6, June 1991, pp.583-598.
- [Vincent93] L.Vincent. *Morphological Gray Scale Reconstruction in Image Analysis: Applications and Efficient Algorithms*. IEEE Transactions on Image Processing, vol.2, No.2, April 1993.
- [Viola01] P.Viola, M.J.Jones. *Rapid Object Detection using a Boosted Cascade of Simple Features*. Proceedings IEEE Conf. on Computer Vision and Pattern Recognition (CVPR 2001), 2001.
- [Wang02] Z.Wang, A.C.Bovik. *A Universal Image Quality Index*. IEEE Signal Processing Letters, vol.9, No.3, March 2002, pp.81-84.
- [Wolberg96] G.Wolberg. *Digital Image Warping*. IEEE Computer Society Press, 1996.

# Glossary

---

absolute colors	Colors specified by each pixel's coordinates in a color space. Intel Integrated Performance Primitives for image processing use images with absolute colors.
alpha channel	A color channel, also known as the opacity channel, that can be used in color models; for example, the RGBA model.
arithmetic operation	An operation that adds, subtracts, multiplies, divides, or squares the image pixel values.
color-twist matrix	A matrix used to multiply the pixel components in one color space for determining the components in another color space.
DCT	Acronym for the discrete cosine transform. See <a href="#">Discrete Cosine Transforms</a> in Chapter 10 of this manual.
dilation	A morphological operation that sets each output pixel to the minimum of the corresponding input pixel and its 8 neighbors.
dyadic operation	An operation that has two input images. It can have other input parameters as well.
element-wise operation	An element-wise operation performs the same operation on each element of a vector, or uses the elements of the same position in multiple vectors as inputs to the operation.
erosion	A morphological operation that sets each output pixel to the maximum of the corresponding input pixel and its 8 neighbors.
four-channel model	A color model that uses four color channels; for example, the RGBA color model.
gray scale image	An image characterized by a single intensity channel so that each intensity value corresponds to a certain shade of gray.
in-place operation	An operation whose output image is one of the input images.
linear filtering	In this manual, 2D convolution operations.
linear image transforms	In this manual, the discrete cosine transform (DCT).
MMX™ technology	An enhancement to the Intel® architecture aimed at better performance in multimedia and communications applications. The technology uses four additional data types, eight 64-bit MMX registers, and 57 additional instructions implementing the SIMD (single instruction, multiple data) technique.
monadic operation	An operation that has a single input image. It can have other input parameters as well.
morphological operation	An erosion, dilation, or their combinations.
not-in-place operation	An operation whose output is an image other than the input image(s). See in-place operation.
pixel depth	The number of bits determining each channel intensity for a single pixel in the image.

pixel-oriented ordering	Storing the image information in such an order that the values of all color channels for each pixel are clustered; for example, RGBRGB... .
planar-oriented ordering	Storing the image information so that all data of one color channel follow all data of another channel, thus forming a separate “plane” for each channel; for example, RRRRRGGGGGBBBBB....
region of interest	A rectangular image region on which an operation acts (or processing occurs).
RGB	Red-green-blue. A three-channel color model that uses red, green, and blue color channels.
RGBA	Red-green-blue-alpha. A four-channel color model that uses red, green, blue, and alpha (or opacity) channels.
ROI	See identity matrix.
row-major order	The default storage method for arrays in C. Memory representation is such that the rows of an array are stored contiguously. For example, for the array <code>a[3][4]</code> , the element <code>a[1][0]</code> immediately follows <code>a[0][3]</code> .
saturation	Using saturation arithmetic, when a number exceeds the data-range limit for its data type, it saturates to the upper data-range limit. For example, a signed word greater than <code>7FFFh</code> saturates to <code>7FFFh</code> . When a number is less than the lower data-range limit, it saturates to the lower data-range. For example, a signed word less than <code>8000h</code> saturates to <code>8000h</code> .
Streaming SIMD Extensions	The enhancement to the Intel architecture instruction set for the next generation processors. It incorporates a group of general-purpose floating-point instructions operating on packed data, additional packed integer instructions, together with cacheability control and state management instructions. These instructions significantly improve performance of applications using compute-intensive processing of floating-point and integer data.
three-channel model	A color model that uses three color channels; for example, the RGB color model.

# Index

## A

- Abs 140
- AbsDiff 141
- AbsDiffC 142
- absolute color images 47
- Add 107
- Add128\_JPEG 967
- Add8x8 1132
- Add8x8HP 1133
- AddBackPredPB\_H263 1329
- AddC 111
- AddC8x8 1134
- AddProduct 115
- AddRandGauss\_Direct 92
- AddRandUnifrom\_Direct 91
- AddRotateShift 737
- AddSquare 114
- AddWeighted 116
- alpha channel 47
- alpha composition functions
  - AlphaComp 171
  - AlphaCompC 173
  - AlphaPremul 175
  - AlphaPremulC 176
- AlphaCompColorKey 340
- anchor cell 48
- And 154
- AndC 155
- ApplyHaarClassifier 898
- arithmetic functions
  - absolute value 140
  - addition 107
  - addition of a constant 111
  - addition with accumulation 114
  - complement 151
  - division 132
  - exponential 149
  - logarithm 147
  - multiplication 118
  - multiplication with scaling 123
  - square 143
  - square root 145
  - subtraction 127
- arithmetic operations 107
- Average16x16 1134
- Average8x8 1134
- AVS Decoder functions
  - FilterDeblockingChroma\_HorEdge\_AVS 1498

- AVS Decoder functions (*continued*)
  - FilterDeblockingChroma\_VerEdge\_AVS 1496
  - FilterDeblockingLuma\_HorEdge\_AVS 1494
  - FilterDeblockingLuma\_VerEdge\_AVS 1493
  - InterpolateLumaBlock\_AVS 1486
  - ReconstructChromaInter\_AVS 1492
  - ReconstructChromaIntra\_AVS 1491
  - ReconstructLumaInter\_AVS 1490
  - ReconstructLumaIntra\_AVS 1489
  - WeightPrediction\_AVS 1487
- AVS Encoder functions
  - DisassembleChroma420Intra\_AVS 1503
  - DisassembleLumaIntra\_AVS 1502
  - TransformQuant8x8Fwd\_AVS 1501

## B

- BGR555ToYCbCr\_JPEG 923
- BGR555ToYCbCr411 266
- BGR555ToYCbCr411LS\_MCU 943
- BGR555ToYCbCr420 263
- BGR555ToYCbCr422 235
- BGR555ToYCbCr422LS\_MCU 941
- BGR555ToYCbCr444LS\_MCU 939
- BGR555ToYUV420 217
- BGR565ToBGR 300
- BGR565ToYCbCr\_JPEG 923
- BGR565ToYCbCr411 266
- BGR565ToYCbCr411LS\_MCU 943
- BGR565ToYCbCr420 263
- BGR565ToYCbCr422 235
- BGR565ToYCbCr422LS\_MCU 941
- BGR565ToYCbCr444LS\_MCU 939
- BGR565ToYUV420 217
- BGRToBGR565 299
- BGRToCbYCr422 237
- BGRToCbYCr422\_709HDTV 238
- BGRToHLS 281
- BGRToLab 273
- BGRToY\_JPEG 917
- BGRToYCbCr\_JPEG 921
- BGRToYCbCr411 264
- BGRToYCbCr411\_JPEG 930
- BGRToYCbCr411LS\_MCU 942
- BGRToYCbCr420 252
- BGRToYCbCr420\_709CSC 253
- BGRToYCbCr420\_709HDTV 254
- BGRToYCbCr422 233

BGRToYCbCr422\_JPEG 927  
 BGRToYCbCr422LS\_MCU 940  
 BGRToYCbCr444LS\_MCU 939  
 BGRToYCoCg 287  
 BGRToYCoCg\_Rev 290  
 BGRToYCrCb420 263  
 BGRToYCrCb420\_709CSC 255  
 BGRToYUV420 212  
 Bidir\_H264 1400  
 BidirWeight\_H264 1404  
 BiDirWeightBlock\_H264 1404  
 BiDirWeightBlockImplicit\_H264 1406  
 BidirWeightImplicit\_H264 1406  
 bitstream parsing 1283, 1309, 1321  
 borders, in neighborhood operations 430, 432  
 BoundSegments 882

## C

CABACEncodeBin\_H264 1476  
 CABACEncodeBinBypass\_H264 1477  
 CABACEncodeResidualBlock\_H264 1478  
 CABACFree\_H264 1475  
 CABACGetContexts\_H264 1472  
 CABACGetSize\_H264 1471  
 CABACGetStreamSize\_H264 1476  
 CABACInit\_H264 1473  
 CABACInitAlloc\_H264 1474  
 CABACSetStream\_H264 1475  
 CABACTerminateSlice\_H264 1477  
 CalcGlobalMV\_MPEG4 1291  
 camera calibration and 3D reconstruction 901  
 Canny 815  
 CannyGetSize 814  
 CbYCr422ToBGR 239  
 CbYCr422ToBGR\_709HDTV 240  
 CbYCr422ToRGB 237  
 CbYCr422ToYCbCr411 314  
 CbYCr422ToYCbCr420 311  
 CbYCr422ToYCbCr420\_Interlace 312  
 CbYCr422ToYCbCr420\_Rotate 1189  
 CbYCr422ToYCbCr422 310  
 CbYCr422ToYCrCb420 313  
 CFAToRGB 296  
 ChangeSpriteBrightness\_MPEG4 1291  
 chromaticity coordinates 186  
 chromaticity diagram 186  
 CIE Lab color model 187  
 CIE LUV color model 187  
 CIE XYZ color model 187  
 CMYK color model 187  
 CMYKToYCCK\_JPEG 932  
 CMYKToYCCK411LS\_MCU 945  
 CMYKToYCCK422LS\_MCU 944  
 CMYKToYCCK444LS\_MCU 943  
 color conversion functions  
     between RGB and HLS 278  
     between RGB and HSV 283  
     between RGB and Lab 273  
     between RGB and LUV 270  
     between RGB and XYZ 268

color conversion functions (*continued*)  
     between RGB and YCbCr 220  
     between RGB and YCbCr422 229  
     between RGB and YCC 276  
     between RGB and YCoCg 285  
     between RGB and YCoCg-R 290  
     between RGB and YUV 203  
     between RGB and YUV420 210  
     between RGB and YUV422 206  
     color to gray scale 294  
     color twist 334, 335  
     gamma correction 341  
 color conversion functions, combined, for JPEG codec 934  
 color conversion functions, for JPEG codec 914  
 color gamut 186  
 color median filter 456  
 color spaces  
     CIE Lab 187  
     CIE LUV 187  
     CIE XYZ 187  
     CMYK 187  
     HLS 187  
     HSV 187  
     Photo YCC 187  
     RGB 187  
     YCbCr 187  
     YCCK 187  
     YCoCg 187  
     YCoCg-R 187  
     YUV 187  
 ColorToGray 295  
 ColorTwist 335  
 ColorTwist32f 336  
 combined quantization, DCT, and level shift functions, for JPEG  
 codec 959  
 Compare 381  
 compare functions 380  
 CompareC 382  
 CompareEqualEps 383  
 CompareEqualEpsC 384  
 CompColorKey 338  
 Complement 151  
 compressed macroblock 1240  
 compressed segment 1240  
 ComputeThreshold\_Otsu 378  
 conversion functions, for JPEG codec 981  
 Convert 60  
 ConvFull 485  
 ConvValid 487  
 Copy 69  
 Copy16x16 1130  
 Copy16x16HP 1130  
 Copy16x16QP\_MPEG4 1285  
 Copy16x8HP 1130  
 Copy16x8QP\_MPEG4 1285  
 Copy8x4HP 1130  
 Copy8x8 1130  
 Copy8x8HP 1130  
 Copy8x8QP\_MPEG4 1285  
 CopyConstBorder 74  
 CopyManaged 72



CopyReplicateBorder 76  
CopySubpix 82  
CopySubpixIntersect 83  
CopyWrapBorder 79  
correction of camera lens distortion  
    CreateMapCameraUndistort 904  
    UndistortGetSize 902  
    UndistortRadial 903  
CountInRange 615  
CountZeros8x8 1271  
CplxExtendToPack 574  
CreateMapCameraUndistort 904  
CrossCorrFull\_Norm 676  
CrossCorrFull\_NormLevel 685  
CrossCorrSame\_Norm 679  
CrossCorrValid 683  
CrossCorrValid\_Norm 682  
CrossCorrValid\_NormLevel 690

## D

data exchange functions  
    adding Gaussian noise 92  
    adding uniform noise 91  
    copying 69  
    copying between images 72  
    copying between images, adding border pixels 74, 79  
    copying between images, replicated border 76  
    copying between images, subpixel precision 82  
    copying intersection values, subpixel precision 83  
    duplicating gray scale image 86  
    initializing 67  
    swap channels 88  
    transpose image 86  
data types 34  
DCT block 1240  
DCT2x4x8Frw 1270  
DCT2x4x8Inv 1265  
DCT8x4x2To4x4Inv\_DV 1267  
DCT8x8Fwd 586  
DCT8x8Fwd\_8u16s\_C2P2 1236  
DCT8x8FwdLS 589  
DCT8x8Inv 588  
DCT8x8Inv\_2x2 590  
DCT8x8Inv\_4x4 590  
DCT8x8Inv\_A10 588  
DCT8x8Inv\_AANTransposed\_16s\_C1R 1227  
DCT8x8Inv\_AANTransposed\_16s\_P2C2R 1228  
DCT8x8Inv\_AANTransposed\_16s8u\_C1R 1228  
DCT8x8Inv\_AANTransposed\_16s8u\_P2C2R 1229  
DCT8x8InvLSClip 590  
DCT8x8InvOrSet 1230  
DCT8x8To2x2Inv 591  
DCT8x8To4x4Inv 591  
DCTFwd 583  
DCTFwdFree 581  
DCTFwdGetBufSize 582  
DCTFwdInitAlloc 579  
DCTInv 585  
DCTInvFree 581  
DCTInvGetBufSize 583

DCTInvInitAlloc 580  
DCTQuantFwd8x8\_JPEG 960  
DCTQuantFwd8x8LS\_JPEG 961  
DCTQuantInv8x8\_JPEG 963  
DCTQuantInv8x8LS\_1x1\_JPEG 965  
DCTQuantInv8x8LS\_2x2\_JPEG 965  
DCTQuantInv8x8LS\_4x4\_JPEG 965  
DCTQuantInv8x8LS\_JPEG 964  
DCTQuantInv8x8To2x2LS\_JPEG 966  
DCTQuantInv8x8To4x4LS\_JPEG 966  
DecimateFilterColumn 447  
DecimateFilterRow 447  
DecodeCAVLCCChroma422DcCoeffs\_H264 1362  
DecodeCAVLCCChromaDcCoeffs\_H264 1362  
DecodeCAVLCCCoeffs\_H264 1361  
DecodeCBProgrAttach\_JPEG2K 1051  
DecodeCBProgrFree\_JPEG2K 1051  
DecodeCBProgrGetStateSize 1049  
DecodeCBProgrInit\_JPEG2K 1050  
DecodeCBProgrInitAlloc\_JPEG2K 1050  
DecodeChromaBlock\_AVS 1485  
DecodeCodeBlock\_JPEG2K 1048  
DecodeCoeffsInter\_H261 1312  
DecodeCoeffsInter\_MPEG4 1297  
DecodeCoeffsInterRVLCBack\_MPEG4 1298  
DecodeCoeffsIntra\_H261 1311  
DecodeCoeffsIntra\_H263 1324  
DecodeCoeffsIntra\_MPEG4 1295  
DecodeCoeffsIntraRVLCBack\_MPEG4 1296  
DecodeDCIntra\_H263 1323  
DecodeDCIntra\_MPEG4 1294  
DecodeExpGolombOne\_H264 1364  
DecodeGetBufSize\_JPEG2K 1047  
DecodeHuffman8x8\_ACFirst\_JPEG 1006  
DecodeHuffman8x8\_ACRefine\_JPEG 1007  
DecodeHuffman8x8\_DCFirst\_JPEG 1004  
DecodeHuffman8x8\_DCRefine\_JPEG 1005  
DecodeHuffman8x8\_Direct\_JPEG 1003  
DecodeHuffman8x8\_JPEG 1002  
DecodeHuffmanOne 1105  
DecodeHuffmanOne\_JPEG 1014  
DecodeHuffmanPair 1106  
DecodeHuffmanRow\_JPEG 1016  
DecodeHuffmanSpecFree\_JPEG 999  
DecodeHuffmanSpecGetBufSize\_JPEG 998  
DecodeHuffmanSpecInit\_JPEG 998  
DecodeHuffmanSpecInitAlloc\_JPEG 999  
DecodeHuffmanStateFree\_JPEG 1001  
DecodeHuffmanStateGetBufSize\_JPEG 1000  
DecodeHuffmanStateInit\_JPEG 1000  
DecodeHuffmanStateInitAlloc\_JPEG 1001  
DecodeLumaBlockInter\_AVS 1484  
DecodeLumaBlockIntra\_AVS 1484  
DeconvFFT 492  
DeconvFFTFree 491  
DeconvFFTInitAlloc 490  
DeconvLR 494  
DeconvLRFree 493  
DeconvLRInitAlloc 492  
deconvolution functions  
    DeconvFFT 492

deconvolution functions (*continued*)

- DeconvFFTFree 491
- DeconvFFTInitAlloc 490
- DeconvLR 494
- DeconvLRFree 493
- DeconvLRInitAlloc 492
- deinterlace filtering functions
  - YCbCr420ToYCrCb420\_Filter 322
- DeinterlaceBlend 1204
- DeinterlaceBlendFree 1205
- DeinterlaceBlendGetSize 1199
- DeinterlaceBlendInit 1200
- DeinterlaceBlendInitAlloc 1202
- DeinterlaceEdgeDetect 1197
- DeinterlaceFilterCAVT 1194
- DeinterlaceFilterTriangle 1193
- DeinterlaceMedianThreshold 1196
- DeinterlaceMotionAdaptive 1198
- deinterlacing 1193
- DemosaicAHD 298
- denoising functions
  - DeinterlaceBlendFree 1205
  - DeinterlaceBlendGetSize 1199
  - DeinterlaceBlendInit 1200
  - DeinterlaceBlendInitAlloc 1202
  - FilterDenoiseAdaptive 1210
  - FilterDenoiseAdaptiveFree 1209
  - FilterDenoiseAdaptiveInitAlloc 1209
  - FilterDenoiseCAST 1207
  - FilterDenoiseCASTInit 1206
  - FilterDenoiseMosquito 1213
  - FilterDenoiseMosquitoFree 1212
  - FilterDenoiseMosquitoInitAlloc 1212
  - FilterDenoiseSmooth 1208
- dequantization, of the DCT coefficients 959
- DequantTransformResidual\_H264 1366
- DequantTransformResidual\_SISP\_H264 1369
- DequantTransformResidualAndAdd\_H264 1367
- descriptors 35
- DFTFree 557
- DFTFwd 558
- DFTGetBufSize 557
- DFTInitAlloc 556
- DFTInv 560
- DiffPredFirstRow\_JPEG 1008
- DiffPredRow\_JPEG 1010
- Dilate 396
- Dilate3x3 393
- DilateBorderReplicate 402
- DisassembleChroma420Intra\_AVS 1503
- DisassembleLumaIntra\_AVS 1502
- Distance transform 826
- DistanceTransform 826
- Div 132
- Div\_Round 135
- DivC 137
- DotProd 151
- DotProdCol 153, 479
- DownsampleFour\_H263 1332
- DownsampleFour16x16\_H263 1345
- downsampling, image 198

- Dup 86
- DV 1239
- DV decoder functions
  - inverse discrete cosine transformation 1265, 1267
  - inverse quantization 1257, 1258, 1262
  - variable length decoding 1247, 1251, 1255, 1256, 1257
- DV encoder functions
  - color conversion 1272, 1273, 1274, 1275, 1276, 1277, 1279
  - discrete cosine transformation 1270, 1271
- DXT1 compression 1065
- DXT5 compression 1065

## E

- EigenValsVecs 817
- EigenValsVecsGetBufferSize 816
- EncodeChromaDcCoeffsCAVLC\_H264\_16s 1470
- EncodeCoeffsCAVLC\_H264\_16s 1467
- EncodeCoeffsInter\_H261 1317
- EncodeCoeffsInter\_H263 1342
- EncodeCoeffsInter\_MPEG4 1308
- EncodeCoeffsIntra\_H261 1316
- EncodeCoeffsIntra\_H263 1341
- EncodeCoeffsIntra\_MPEG4 1307
- EncodeDCIntra\_H263 1340
- EncodeDCIntra\_MPEG4 1306
- EncodeFree\_JPEG2K 1042
- EncodeGetDist\_JPEG2K 1047
- EncodeGetRate\_JPEG2K 1046
- EncodeGetTermPassLen\_JPEG2K 1045
- EncodeHuffman8x8\_ACFirst\_JPEG 996
- EncodeHuffman8x8\_ACRefine\_JPEG 997
- EncodeHuffman8x8\_DCFirst\_JPEG 994
- EncodeHuffman8x8\_DCRefine\_JPEG 995
- EncodeHuffman8x8\_Direct\_JPEG 990
- EncodeHuffman8x8\_JPEG 989
- EncodeHuffmanOne\_JPEG 1013
- EncodeHuffmanRawTableInit\_JPEG 984
- EncodeHuffmanRow\_JPEG 1015
- EncodeHuffmanSpecFree\_JPEG 987
- EncodeHuffmanSpecGetBufSize\_JPEG 985
- EncodeHuffmanSpecInit\_JPEG 986
- EncodeHuffmanSpecInitAlloc\_JPEG 986
- EncodeHuffmanStateFree\_JPEG 989
- EncodeHuffmanStateGetBufSize\_JPEG 987
- EncodeHuffmanStateInit\_JPEG 988
- EncodeHuffmanStateInitAlloc\_JPEG 988
- EncodeInitAlloc\_JPEG2K 1041
- EncodeLoadCodeBlock\_JPEG2K 1042
- EncodeStoreBits\_JPEG2K 1044
- Enumerators in IPPI 41
- Erode 397
- Erode3x3 394
- ErodeBorderReplicate 404
- error messages 38
- Exp 149
- ExpandPlane\_H264 1379

**F**

- feature detection functions
  - Canny 815
  - CannyGetSize 814
  - EigenValsVecs 817
  - EigenValsVecsGetBufferSize 816
  - MinEigenVal 821
  - MinEigenValGetBufferSize 820
- FFTFree 549
- FFTFwd 550
- FFTGetBufSize 550
- FFTInitAlloc 548
- FFTInv 554
- FilterScharrHorizGetBufferSize 518
- Filter 458
- Filter\_Round16s 462
- Filter\_Round32f 462
- Filter\_Round32s 462
- Filter32f 460
- Filter8x8\_H261 1318
- FilterBilateral 444
- FilterBilateralGetBufSize 442
- FilterBilateralInit 443
- FilterBlockBoundaryHorEdge\_H263 1335
- FilterBlockBoundaryVerEdge\_H263 1335
- FilterBox 430, 432
- FilterColumn 466
- FilterColumn32f 468
- FilterColumnPipeline 476
- FilterColumnPipeline\_Low 476
- FilterColumnPipelineGetBufferSize 473
- FilterColumnPipelineGetBufferSize\_Low 473
- FilterDeblocking16x16\_HorEdge\_H263 1336
- FilterDeblocking16x16\_VerEdge\_H263 1336
- FilterDeblocking8x8HorEdge\_H263 1335
- FilterDeblocking8x8HorEdge\_MPEG4 1300
- FilterDeblocking8x8VerEdge\_H263 1335
- FilterDeblocking8x8VerEdge\_MPEG4 1300
- FilterDeblockingChroma\_HorEdge\_AVS 1498
- FilterDeblockingChroma\_HorEdge\_H264 1453
- FilterDeblockingChroma\_HorEdge\_VC1 1538
- FilterDeblockingChroma\_VerEdge\_AVS 1496
- FilterDeblockingChroma\_VerEdge\_H264 1447
- FilterDeblockingChroma\_VerEdge\_MBAFF\_H264 1451
- FilterDeblockingChroma\_VerEdge\_VC1 1538
- FilterDeblockingChroma422HorEdge\_H264 1453
- FilterDeblockingChroma422VerEdge\_H264 1447
- FilterDeblockingChroma422VerEdge\_MBAFF\_H264 1451
- FilterDeblockingChromaHorEdge\_H264 1453
- FilterDeblockingChromaVerEdge\_H264 1447
- FilterDeblockingLuma\_HorEdge\_AVS 1494
- FilterDeblockingLuma\_HorEdge\_H264 1445
- FilterDeblockingLuma\_HorEdge\_VC1 1537
- FilterDeblockingLuma\_VerEdge\_AVS 1493
- FilterDeblockingLuma\_VerEdge\_H264 1443
- FilterDeblockingLuma\_VerEdge\_MBAFF\_H264 1444
- FilterDeblockingLuma\_VerEdge\_VC1 1536
- FilterDenoiseAdaptive 1210
- FilterDenoiseAdaptiveFree 1209
- FilterDenoiseAdaptiveInitAlloc 1209
- FilterDenoiseCAST 1207
- FilterDenoiseCASTInit 1206
- FilterDenoiseMosquito 1213
- FilterDenoiseMosquitoFree 1212
- FilterDenoiseMosquitoInitAlloc 1212
- FilterDenoiseSmooth 1208
- FilterDeringingSmooth8x8\_MPEG4 1301
- FilterDeringingThreshold\_MPEG4 1301
- FilterGauss 511
- FilterGaussBorder 540
- FilterGaussGetBufferSize 524
- FilterGetBufSize 460
- FilterHipass 515
- filtering functions
  - bilateral filter 442
  - box filter 430, 432
  - color median filter 456
  - column filter 466
  - decimate filters 447
  - Gaussian filter 511
  - general rectangular filter 458
  - highpass filter 515
  - Laplacian filter 510
  - lowpass filter 516
  - max filter 437
  - max filter border replication 441
  - max filter working buffer size 439
  - median filter 450
  - median filter with weighted center pixel 455
  - min filter 435
  - min filter border replication 439
  - min filter working buffer size 438
  - Prewitt filter 495
  - Roberts filter 508
  - row filter 469
  - Scharr filter 498
  - sharpening filter 517
  - Sobel filter 500, 502
- FilterLaplace 510
- FilterLaplacianBorder 538
- FilterLaplacianGetBufferSize 523
- FilterLowpass 516
- FilterLowpassBorder 541
- FilterLowpassGetBufferSize 525
- FilterMax 437
- FilterMaxBorderReplicate 441
- FilterMaxGetBufferSize 439
- FilterMedian 450
- FilterMedianColor 456
- FilterMedianCross 454
- FilterMedianGetBufferSize 449
- FilterMedianHoriz 452
- FilterMedianVert 453
- FilterMedianWeightedCenter3x3 455
- FilterMin 435
- FilterMinBorderReplicate 439
- FilterMinGetBufferSize 438
- FilterPrewittHoriz 495
- FilterPrewittVert 496
- FilterRobertsDown 508
- FilterRobertsUp 509

FilterRoundGetBufSize16s 464  
FilterRoundGetBufSize32f 464  
FilterRoundGetBufSize32s 464  
FilterRow 469  
FilterRow32f 471  
FilterRowBorderPipeline 474  
FilterRowBorderPipeline\_Low 474  
FilterRowBorderPipelineGetBufferSize 472  
FilterRowBorderPipelineGetBufferSize\_Low 472  
FilterScharrHoriz 498  
FilterScharrHorizBorder 526  
FilterScharrVert 499  
FilterScharrVertBorder 528  
FilterScharrVertGetBufferSize 519  
FilterSharpen 517  
FilterSobelCross 507  
FilterSobelCrossBorder 537  
FilterSobelCrossGetBufferSize 523  
FilterSobelHoriz 500  
FilterSobelHorizBorder 529  
FilterSobelHorizGetBufferSize 520  
FilterSobelHorizSecond 503  
FilterSobelHorizSecondBorder 533  
FilterSobelHorizSecondGetBufferSize 521  
FilterSobelNegVertBorder 531  
FilterSobelNegVertGetBufferSize 520  
FilterSobelVert 502  
FilterSobelVertBorder 531  
FilterSobelVertGetBufferSize 520  
FilterSobelVertSecond 504  
FilterSobelVertSecondBorder 534  
FilterSobelVertSecondGetBufferSize 522  
FilterWiener 481  
FilterWienerGetBufferSize 480  
FindPeaks3x3 628  
FindPeaks3x3GetBufferSize 628  
fixed filters 495  
fixed filters with border 518  
FLCDecode4x4\_JPEGXR 1084  
FLCEncode4x4\_JPEGXR 1083  
flood fill functions  
    FloodFill 838  
    FloodFill\_Grad 840  
    FloodFill\_Range¶ 843  
    FloodFillGetSize 837  
    FloodFillGetSize\_Grad 838  
FloodFill 838  
FloodFill\_Grad 840  
FloodFill\_Range 843  
FloodFillGetSize 837  
FloodFillGetSize\_Grad 838  
ForegroundGaussian 891  
ForegroundGaussianFree 890  
ForegroundGaussianInitAlloc 889  
ForegroundHistogram 885  
ForegroundHistogramFree 885  
ForegroundHistogramInitAlloc 883  
ForegroundHistogramUpdate 888  
format conversion functions  
    BGR565ToBGR 300  
    BGRTToBGR565 299

format conversion functions (*continued*)  
    CbYCr422ToYCbCr411 314  
    CbYCr422ToYCbCr420 311  
    CbYCr422ToYCbCr420\_Interlace 312  
    CbYCr422ToYCbCr422 310  
    CbYCr422ToYCrCb420 313  
    RGB565ToRGB 300  
    RGBToRGB565 299  
    YCbCr411 329  
    YCbCr411ToYCbCr420 332  
    YCbCr411ToYCbCr422 330  
    YCbCr411ToYCrCb420 334  
    YCbCr411ToYCrCb422 331  
    YCbCr420 315  
    YCbCr420To422\_Interlace 319  
    YCbCr420ToCbYCr422 320  
    YCbCr420ToCbYCr422\_Interlace 320  
    YCbCr420ToYCbCr411 324  
    YCbCr420ToYCbCr422 316  
    YCbCr420ToYCbCr422\_Filter 317  
    YCbCr420ToYCrCb420 321  
    YCbCr420ToYCrCb420\_Filter 322  
    YCbCr422 301  
    YCbCr422To420\_Interlace 305  
    YCbCr422ToCbYCr422 302  
    YCbCr422ToYCbCr411 307  
    YCbCr422ToYCbCr420 303  
    YCbCr422ToYCrCb420 306  
    YCrCb420ToCbYCr422 327  
    YCrCb420ToYCbCr411 328  
    YCrCb420ToYCbCr420 327  
    YCrCb420ToYCbCr422 325  
    YCrCb420ToYCbCr422\_Filter 326  
    YCrCb422ToYCbCr411 310  
    YCrCb422ToYCbCr420 309  
    YCrCb422ToYCbCr422 308  
forward discrete cosine transformation 1236  
Fourier transforms  
    DFTFree 557  
    DFTFwd 558  
    DFTIGetBufSize 557  
    DFTInitAlloc 556  
    DFTInv 560  
    FFTFree 549  
    FFTFwd 550  
    FFTGetBufSize 550  
    FFTInitAlloc 548  
    FFTInv 554  
FrameFieldSAD16x16 1182  
Free 57  
FreeHuffmanTable\_DV 1257  
function context 46  
function naming conventions 33  
function parameters 36  
function prototypes 36

## G

gamma correction 186  
GammaFwd 341  
GammaInv 343

GenScaleLevel8x8\_H264 1466  
 GenSobelKernel 525  
 geometric transform functions  
   affine warp 751  
   bilinear warp 775  
   mirroring 726  
   perspective warp 763  
   remapping 728  
   rotation 733  
   shear 745  
   resizing 697, 718  
 GetAffineBound 759  
 GetAffineQuad 758  
 GetAffineTransform 760  
 GetBilinearBound 783  
 GetBilinearQuad 782  
 GetBilinearTransform 784  
 GetCentralMoment 638  
 GetDiff16x16 1137  
 GetDiff16x16B 1144  
 GetDiff16x8 1138  
 GetDiff16x8B 1145  
 GetDiff4x4 1143  
 GetDiff8x16 1140  
 GetDiff8x16B 1147  
 GetDiff8x4 1141  
 GetDiff8x4B 1148  
 GetDiff8x8 1139  
 GetDiff8x8B 1146  
 GetDistanceTransformMask 830  
 GetHaarClassifierSize 897  
 GetHuffmanStatistics8x8\_ACFirst\_JPEG 992  
 GetHuffmanStatistics8x8\_ACRrefine\_JPEG 993  
 GetHuffmanStatistics8x8\_DCFirst\_JPEG 992  
 GetHuffmanStatistics8x8\_JPEG 991  
 GetHuffmanStatisticsOne\_JPEG 1012  
 GetHuMoments 640  
 GetLibVersion 53  
 GetNormalizedCentralMoment 639  
 GetNormalizedSpatialMoment 638  
 GetPerspectiveBound 771  
 GetPerspectiveQuad 770  
 GetPerspectiveTransform 771  
 GetPyramidDownROI 864  
 GetResizeFract 715  
 GetResizeRect 713  
 GetRotateBound 738  
 GetRotateQuad 737  
 GetRotateShift 736  
 GetShearBound 750  
 GetShearQuad 749  
 GetSpatialMoment 637  
 GradientColorToGray 835  
 GrayDilateBorder 417, 418

## H

H.263 decoder inverse quantization  
   QuantInvInter\_H263 1328  
   QuantInvIntra\_H263 1326

H.263 decoder prediction functions  
   AddBackPredPB\_H263 1329  
 H.263 decoder resampling functions  
   DownsampleFour\_H263 1332  
   Resample\_H263 1330  
   SpatialInterpolation\_H263 1334  
   UnsampleFour\_H263 1332  
   UnsampleFour8x8\_H263 1333  
 H.263 Encoder functions  
   DownsampleFour16x16\_H263 1345  
   EncodeCoeffsInter\_H263 1342  
   EncodeCoeffsIntra\_H263 1341  
   EncodeDCIntra\_H263 1340  
   QuantInter\_H263 1344  
   QuantIntra\_H263 1343  
 H.263+ functions  
   DownsampleFour\_H263 1332  
   FilterBlockBoundaryHorEdge\_H263 1335  
   FilterDeblocking16x16HorEdge\_H263 1336  
   FilterDeblocking8x8HorEdge\_H263 1335  
   ReconstructCoeffsInter\_H263 1338  
   ReconstructCoeffsIntra\_H263 1337  
   Resample\_H263 1330  
   SpatialInterpolation\_H263 1334  
   UnsampleFour\_H263 1332  
   UnsampleFour8x8\_H263 1333  
 H.263+ middle-level functions  
   ReconstructCoeffsInter\_H263 1338  
   ReconstructCoeffsIntra\_H263 1337  
 H.263+ VLC decoding  
   DecodeCoeffsInter\_H263 1325  
   DecodeCoeffsIntra\_H263 1324  
   DecodeDCIntra\_H263 1323  
 H263 decoder boundary filtering functions  
   FilterBlockBoundaryHorEdge\_H263 1335  
   FilterBlockBoundaryVerEdge\_H263 1335  
   FilterDeblocking16x16HorEdge\_H263 1336  
   FilterDeblocking16x16VerEdge\_H263 1336  
   FilterDeblocking8x8HorEdge\_H263 1335  
   FilterDeblocking8x8VerEdge\_H263 1335  
 H264 Decoder functions  
   Bidir\_H264 1400  
   BidirWeight\_H264 1404  
   BiDirWeightBlock\_H264 1404  
   BiDirWeightBlockImplicit\_H264 1406  
   BidirWeightImplicit\_H264 1406  
   DequantTransformResidual\_H264 1366  
   DequantTransformResidual\_SISP\_H264 1369  
   DequantTransformResidualAndAdd\_H264 1367  
   ExpandPlane\_H264 1379  
   FilterDeblockingChroma\_HorEdge\_H264 1453  
   FilterDeblockingChroma\_VerEdge\_H264 1447  
   FilterDeblockingChroma\_VerEdge\_MBAFF\_H264 1451  
   FilterDeblockingChroma422HorEdge\_H264 1453  
   FilterDeblockingChroma422VerEdge\_H264 1447  
   FilterDeblockingChroma422VerEdge\_MBAFF\_H264 1451  
   FilterDeblockingChromaHorEdge\_H264 1453  
   FilterDeblockingChromaVerEdge\_H264 1447  
   FilterDeblockingLuma\_HorEdge\_H264 1445  
   FilterDeblockingLuma\_VerEdge\_H264 1443  
   FilterDeblockingLuma\_VerEdge\_MBAFF\_H264 1444

H264 Decoder functions (*continued*)

high profile functions 1414, 1419, 1421, 1430, 1432, 1434, 1435, 1436, 1438, 1440  
InterpolateBlock\_H264 1400  
InterpolateChroma\_H264 1389  
InterpolateChromaBlock\_H264 1397  
InterpolateChromaBottom\_H264 1396  
InterpolateChromaTop\_H264 1394  
InterpolateLuma\_H264 1380  
InterpolateLumaBlock\_H264 1388  
InterpolateLumaBottom\_H264 1386  
InterpolateLumaTop\_H264 1383  
PredictIntra\_16x16\_H264 1374  
PredictIntra\_4x4\_H264 1371  
PredictIntraChroma8x8\_H264 1375  
ReconstructChroma422Inter4x4\_H264High 1417  
ReconstructChroma422Intra4x4\_H264High 1425  
ReconstructChroma422IntraHalf4x4\_H264High 1423  
ReconstructChromaInter4x4\_H264High 1414  
ReconstructChromaInter4x4MB\_H264 1414  
ReconstructChromaInterMB\_H264 1409  
ReconstructChromaIntra4x4\_H264High 1421  
ReconstructChromaIntra4x4MB\_H264 1421  
ReconstructChromaIntraHalf4x4High\_H264 1419  
ReconstructChromaIntraHalves4x4MB\_H264 1419  
ReconstructChromaIntraHalvesMB\_H264 1411  
ReconstructChromaIntraMB\_H264 1412  
ReconstructLumaInter4x4\_H264High 1430  
ReconstructLumaInter4x4MB\_H264 1430  
ReconstructLumaInter8x8\_H264High 1435  
ReconstructLumaInter8x8MB\_H264 1435  
ReconstructLumaInterMB\_H264 1427  
ReconstructLumaIntra16x16\_H264High 1440  
ReconstructLumaIntra16x16MB\_H264 1439, 1440  
ReconstructLumaIntra4x4\_H264High 1434  
ReconstructLumaIntra4x4MB\_H264 1434  
ReconstructLumaIntra8x8\_H264High 1438  
ReconstructLumaIntra8x8MB\_H264 1438  
ReconstructLumaIntraHalf4x4\_H264High 1432  
ReconstructLumaIntraHalf4x4MB\_H264 1432  
ReconstructLumaIntraHalf8x8\_H264High 1436  
ReconstructLumaIntraHalf8x8MB\_H264 1436  
ReconstructLumaIntraHalfMB\_H264 1428  
ReconstructLumaIntraMB\_H264 1429  
TransformDequantChromaDC\_H264 1365  
TransformDequantChromaDC\_SISP\_H264 1370  
TransformDequantLumaDC\_H264 1364  
TransformPrediction\_H264 1369  
TransformResidual4x4Inv\_H264 1365  
UnidirWeight\_H264 1403  
UniDirWeightBlock\_H264 1403  
WeightedAverage\_H264 1402

H264 Encoder functions

EncodeChromaDcCoeffsCAVLC\_H264\_16s 1470  
EncodeCoeffsCAVLC\_H264\_16s 1467  
GenScaleLevel8x8\_H264 1466  
QuantizeResidual4x4Fwd\_H264 1462  
QuantLuma8x8\_H264 1465  
QuantLuma8x8Inv\_H264 1479  
TransformLuma8x8Fwd\_H264 1464  
TransformLuma8x8InvAddPred\_H264 1480

H264 Encoder functions (*continued*)

TransformQuantChromaDC\_H264 1459  
TransformQuantLumaDC\_H264 1461  
TransformQuantResidual\_H264 1463  
TransformResidual4x4Fwd\_H264 1462  
H264 video encoder functions  
CABACEncodeBin\_H264 1476  
CABACEncodeBinBypass\_H264 1477  
CABACEncodeResidualBlock\_H264 1478  
CABACFree\_H264 1475  
CABACGetContexts\_H264 1472  
CABACGetSize\_H264 1471  
CABACGetStreamSize\_H264 1476  
CABACInit\_H264 1473  
CABACInitAlloc\_H264 1474  
CABACSetStream\_H264 1475  
CABACTerminateSlice\_H264 1477

Haar Features 893

HaarClassifierFree 896

HaarClassifierInitAlloc 894

high definition photo coding 1073

Hint arguments for Image Moment functions 632

HistogramEven 613

HistogramRange 611

HLS color model 187

HLSToBGR 282

HLSToRGB 279

horizontal sampling, in a JPEG codec 968

Hough transform 823

Hough transform functions

HoughLine 824

HoughLine\_Region 825

HoughLineGetSize 823

HoughLine 824

HoughLine\_Region 825

HoughLineGetSize 823

HSV color model 187

HSVToRGB 284

Huffman codec functions, for JPEG codec 983

HuffmanDecodeSegment\_DV 1251

HuffmanDecodeSegment\_DV100 1255

HuffmanDecodeSegmentOnePass\_DV 1256

HuffmanRunLevelTableInitAlloc 1104

HuffmanTableFree 1105

HuffmanTableInitAlloc 1103

## I

ICTFwd\_JPEG2K 1059

ICTInv\_JPEG2K 1060

image area extension 430, 432

image bit depth 47

image data storage 47

image format conversion 299

image segmentation 875

ImageJaehne 93

ImageRamp 95

InitAllocHuffmanTable\_DV 1247

initialization functions

creating a test image 93, 95

Inpaint 873

InpaintFree 873  
InpaintingInitAlloc 871  
Integral 597  
intensity transformation functions  
    LUT 348  
    LUT\_Cubic 355  
    LUT\_Linear 351  
    LUTPalette 358  
    LUTPaletteSwap 358  
    ReduceBits 346  
    ToneMapLinear 361  
    ToneMapMean 361  
INTER macroblock decoding 1310, 1322  
interfield compression 1239  
InterpolateAverage16x16 1131  
InterpolateAverage16x8 1131  
InterpolateAverage8x4 1131  
InterpolateBlock\_H264 1400  
InterpolateChroma\_H264 1389  
InterpolateChromaBlock\_H264 1397  
InterpolateChromaBottom\_H264 1396  
InterpolateChromaTop\_H264 1394  
InterpolateICBicubicBlock\_VC1 1525  
InterpolateICBilinearBlock\_VC1 1523  
InterpolateLuma\_H264 1380  
InterpolateLumaBlock\_AVS 1486  
InterpolateLumaBlock\_H264 1388  
InterpolateLumaBottom\_H264 1386  
InterpolateLumaTop\_H264 1383  
InterpolateQPBicubic\_VC1 1522  
InterpolateQPBilinear\_VC1 1520  
interpolation method 693  
INTRA macroblock decoding 1310, 1322  
inverse discrete cosine transformation 1227  
ipiFloodFill 838  
IPP functionality  
    image statistics 593  
    alpha composition 170  
    computer vision 809  
    data exchange functions 59  
    geometric transforms 693  
    H.261 video decoder 1309  
    H.263 video decoder 1321  
    image arithmetic and logical operations 105  
    image color conversion functions 179  
    intensity transformation 346  
    linear transforms 545  
    morphological functions 387  
    MPEG-4 video decoder 1283  
    threshold and compare functions 363  
ippGetStatusString 55  
ippiAbs 140  
ippiAbsDiff 141  
ippiAbsDiffC 142  
ippiAdd 107  
ippiAdd128\_JPEG 967  
ippiAdd8x8 1132  
ippiAdd8x8HP 1133  
ippiAddBackPredPB\_H263 1329  
ippiAddC 111  
ippiAddC8x8 1134  
ippiAddProduct 115  
ippiAddRandGauss\_Direct 92  
ippiAddRandUniform\_Direct 91  
ippiAddRotateShift 737  
ippiAddSquare 114  
ippiAddWeighted 116  
ippiAlphaComp 171  
ippiAlphaCompC 173  
ippiAlphaCompColorKey 340  
ippiAlphaPremul 175  
ippiAlphaPremulC 176  
ippiAnd 154  
ippiAndC 155  
ippiAverage16x16 1134  
ippiAverage8x8 1134  
ippiBGR555ToYCbCr\_JPEG 923  
ippiBGR555ToYCbCr411 266  
ippiBGR555ToYCbCr411LS\_MCU 943  
ippiBGR555ToYCbCr420 263  
ippiBGR555ToYCbCr422 235  
ippiBGR555ToYCbCr422LS\_MCU 941  
ippiBGR555ToYCbCr444LS\_MCU 939  
ippiBGR565ToBGR 300  
ippiBGR565ToYCbCr411 266  
ippiBGR565ToYCbCr411LS\_MCU 943  
ippiBGR565ToYCbCr420 263  
ippiBGR565ToYCbCr422 235  
ippiBGR565ToYCbCr422LS\_MCU 941  
ippiBGR565ToYCbCr444LS\_MCU 939  
ippiBGRTToBGR565 299  
ippiBGRTToCbYCr422 237  
ippiBGRTToCbYCr422\_709HDTV 238  
ippiBGRTToHLS 281  
ippiBGRTToLab 273  
ippiBGRTToY\_JPEG 917  
ippiBGRTToYCbCr\_JPEG 921  
ippiBGRTToYCbCr411\_JPEG 930  
ippiBGRTToYCbCr411LS\_MCU 942  
ippiBGRTToYCbCr420 252  
ippiBGRTToYCbCr420\_709CSC 253  
ippiBGRTToYCbCr420\_709HDTV 254  
ippiBGRTToYCbCr422 233  
ippiBGRTToYCbCr422\_JPEG 927  
ippiBGRTToYCbCr422LS\_MCU 940  
ippiBGRTToYCbCr444LS\_MCU 939  
ippiBGRTToYCoCg 287  
ippiBGRTToYCoCg\_Rev 290  
ippiBGRTToYCrCb420 263  
ippiBGRTToYCrCb420\_709CSC 255  
ippiBGRTToYUV420 212  
ippiBidir\_H264 1400  
ippiBidirWeight\_H264 1404  
ippiBiDirWeightBlock\_H264 1404  
ippiBiDirWeightBlockImplicit\_H264 1406  
ippiBidirWeightImplicit\_H264 1406  
ippiBoundSegments 882  
ippiCABACEncodeBin\_H264 1476  
ippiCABACEncodeBinBypass\_H264 1477  
ippiCABACEncodeResidualBlock\_H264 1478  
ippiCABACFree\_H264 1475  
ippiCABACGetContexts\_H264 1472

ippiCABACGetSize_H264	1471	ippiCrossCorrValid_Norm	682
ippiCABACGetStreamSize_H264	1476	ippiCrossCorrValid_NormLevel	690
ippiCABACInit_H264	1473	ippiDCT2x4x8Frw	1270
ippiCABACInitAlloc_H264	1474	ippiDCT2x4x8Inv	1265
ippiCABACSetStream_H264	1475	ippiDCT8x4x2To4x4Inv_DV	1267
ippiCABACTerminateSlice_H264	1477	ippiDCT8x8Fwd	586
ippiCalcGlobalMV_MPEG4	1291	ippiDCT8x8Fwd_8u16s_C2P2	1236
ippiCanny	815	ippiDCT8x8FwdLS	589
ippiCannyGetSize	814	ippiDCT8x8Inv	588
ippiCbYCr422ToBGR	239	ippiDCT8x8Inv_2x2	590
ippiCbYCr422ToBGR_709HDTV	240	ippiDCT8x8Inv_4x4	590
ippiCbYCr422ToRGB	237	ippiDCT8x8Inv_A10	588
ippiCbYCr422ToYCbCr411	314	ippiDCT8x8Inv_AANTransposed_16s_C1R	1227
ippiCbYCr422ToYCbCr420	311	ippiDCT8x8Inv_AANTransposed_16s_P2C2R	1228
ippiCbYCr422ToYCbCr420_Interlace	312	ippiDCT8x8Inv_AANTransposed_16s8u_C1R	1228
ippiCbYCr422ToYCbCr422	310	ippiDCT8x8Inv_AANTransposed_16s8u_P2C2R	1229
ippiCbYCr422ToYCbCr422_Rotate	1189	ippiDCT8x8InvLSClip	590
ippiCbYCr422ToYCrCb420	313	ippiDCT8x8InvOrSet	1230
ippiCFAToRGB	296	ippiDCT8x8To2x2Inv	591
ippiChangeSpriteBrightness_MPEG4	1291	ippiDCT8x8To4x4Inv	591
ippiCMYKToYCK_JPEG	932	ippiDCTFwd	583
ippiCMYKToYCK411LS_MCU	945	ippiDCTFwdFree	581
ippiCMYKToYCK422LS_MCU	944	ippiDCTFwdGetBufSize	582
ippiCMYKToYCK444LS_MCU	943	ippiDCTFwdInitAlloc	579
ippiColorToGray	295	ippiDCTInv	585
ippiColorTwist	335	ippiDCTInvFree	581
ippiColorTwist32f	336	ippiDCTInvGetBufSize	583
ippiColumnPipeline_Low	476	ippiDCTInvInitAlloc	580
ippiCompare	381	ippiDCTQuantFwd8x8_JPEG	960
ippiCompareC	382	ippiDCTQuantFwd8x8LS_JPEG	961
ippiCompareEqualEps	383	ippiDCTQuantInv8x8_4x4LS_JPEG	966
ippiCompareEqualEpsC	384	ippiDCTQuantInv8x8_JPEG	963
ippiCompColorKey	338	ippiDCTQuantInv8x8LS_1x1_JPEG	965
ippiComplement	151	ippiDCTQuantInv8x8LS_2x2_JPEG	965
ippiComputeThreshold_Otsu	378	ippiDCTQuantInv8x8LS_4x4_JPEG	965
ippiConvert	60	ippiDCTQuantInv8x8LS_JPEG	964
ippiConvFull	485	ippiDCTQuantInv8x8To2x2LS_JPEG	966
ippiConvValid	487	ippiDCTQuantInv8x8To4x4LS_JPEG	966
ippiCopy	69	ippiDecimateFilterColumn	447
ippiCopy16x16	1130	ippiDecimateFilterRow	447
ippiCopy16x16HP	1130	ippiDecodeCAVLCChroma422DcCoeffs_H264	1362
ippiCopy16x16QP_MPEG4	1285	ippiDecodeCAVLCChromaDcCoeffs_H264	1362
ippiCopy16x8HP	1130	ippiDecodeCAVLCCoeffs_H264	1361
ippiCopy16x8QP_MPEG4	1285	ippiDecodeCBProgrAttach_JPEG2K	1051
ippiCopy8x4HP	1130	ippiDecodeCBProgrFree_JPEG2K	1051
ippiCopy8x8	1130	ippiDecodeCBProgrGetStateSize	1049
ippiCopy8x8HP	1130	ippiDecodeCBProgrInit_JPEG2K	1050
ippiCopy8x8QP_MPEG4	1285	ippiDecodeCBProgrInitAlloc_JPEG2K	1050
ippiCopyConstBorder	74	ippiDecodeChromaBlock_AVS	1485
ippiCopyManaged	72	ippiDecodeCodeBlock_JPEG2K	1048
ippiCopyReplicateBorder	76	ippiDecodeCoeffsInter_H261	1312
ippiCopySubpix	82	ippiDecodeCoeffsInter_MPEG4	1297
ippiCopySubpixIntersect	83	ippiDecodeCoeffsInterRVLCBack_MPEG4	1298
ippiCopyWrapBorder	79	ippiDecodeCoeffsIntra_H261	1311
ippiCountInRange	615	ippiDecodeCoeffsIntra_H263	1324
ippiCountZeros8x8	1271	ippiDecodeCoeffsIntra_MPEG4	1295
ippiCreateMapCameraUndistort	904	ippiDecodeCoeffsIntraRVLCBack_MPEG4	1296
ippiCrossCorrFull_Norm	676	ippiDecodeDCIntra_H263	1323
ippiCrossCorrFull_NormLevel	685	ippiDecodeDCIntra_MPEG4	1294
ippiCrossCorrSame_Norm	679	ippiDecodeExpGolombOne_H264	1364
ippiCrossCorrValid	683	ippiDecodeGetBufSize_JPEG2K	1047



ippiDecodeHuffman8x8\_ACFirst\_JPEG 1006  
ippiDecodeHuffman8x8\_ACRrefine\_JPEG 1007  
ippiDecodeHuffman8x8\_DCFirst\_JPEG 1004  
ippiDecodeHuffman8x8\_DCRefine\_JPEG 1005  
ippiDecodeHuffman8x8\_Direct\_JPEG 1003  
ippiDecodeHuffman8x8\_JPEG 1002  
ippiDecodeHuffmanOne 1105  
ippiDecodeHuffmanOne\_JPEG 1014  
ippiDecodeHuffmanPair 1106  
ippiDecodeHuffmanRow\_JPEG 1016  
ippiDecodeHuffmanSpecFree\_JPEG 999  
ippiDecodeHuffmanSpecGetBufSize\_JPEG 998  
ippiDecodeHuffmanSpecInit\_JPEG 998  
ippiDecodeHuffmanSpecInitAlloc\_JPEG 999  
ippiDecodeHuffmanStateFree\_JPEG 1001  
ippiDecodeHuffmanStateGetBufSize\_JPEG 1000  
ippiDecodeHuffmanStateInit\_JPEG 1000  
ippiDecodeHuffmanStateInitAlloc\_JPEG 1001  
ippiDecodeLumaBlockInter\_AVS 1484  
ippiDecodeLumaBlockIntra\_AVS 1484  
ippiDeconvFFT 492  
ippiDeconvFFTFree 491  
ippiDeconvFFTInitAlloc 490  
ippiDeconvLR 494  
ippiDeconvLRFree 493  
ippiDeconvLRInitAlloc 492  
ippiDeinterlaceBlend 1204  
ippiDeinterlaceBlendFree 1205  
ippiDeinterlaceBlendGetSize 1199  
ippiDeinterlaceBlendInit 1200  
ippiDeinterlaceBlendInitAlloc 1202  
ippiDeinterlaceEdgeDetect 1197  
ippiDeinterlaceFilterCAVT 1194  
ippiDeinterlaceFilterTriangle 1193  
ippiDeinterlaceMedianThreshold 1196  
ippiDeinterlaceMotionAdaptive 1198  
ippiDemosaicAHD 298  
ippiDequantTransformResidual\_H264 1366  
ippiDequantTransformResidual\_SISP\_H264 1369  
ippiDequantTransformResidualAndAdd\_H264 1367  
ippiDFTFree 557  
ippiDFTFwd 558  
ippiDFTGetBufSize 557  
ippiDFTInitAlloc 556  
ippiDFTInv 560  
ippiDiffPredFirstRow\_JPEG 1008  
ippiDiffPredRow\_JPEG 1010  
ippiDilate 396  
ippiDilate3x3 393  
ippiDilateBorderReplicate 402  
ippiDisassembleChroma420Intra\_AVS 1503  
ippiDisassembleLumaIntra\_AVS 1502  
ippiDistanceTransform 826  
ippiDiv 132  
ippiDiv\_Round 135  
ippiDivC 137  
ippiDotProd 151  
ippiDotProdCol 153, 479  
ippiDownsampleFour\_H263 1332  
ippiDownsampleFour16x16\_H263 1345  
ippiDup 86  
ippiEigenValsVecs 817  
ippiEigenValsVecsGetBufferSize 816  
ippiEncodeChromaDcCoeffsCAVLC\_H264\_16s 1470  
ippiEncodeCoeffsCAVLC\_H264\_16s 1467  
ippiEncodeCoeffsInter\_H261 1317  
ippiEncodeCoeffsInter\_H263 1342  
ippiEncodeCoeffsInter\_MPEG4 1308  
ippiEncodeCoeffsIntra\_H261 1316  
ippiEncodeCoeffsIntra\_H263 1341  
ippiEncodeCoeffsIntra\_MPEG4 1307  
ippiEncodeDCIntra\_H263 1340  
ippiEncodeDCIntra\_MPEG4 1306  
ippiEncodeFree\_JPEG2K 1042  
ippiEncodeGetDist\_JPEG2K 1047  
ippiEncodeGetRate\_JPEG2K 1046  
ippiEncodeGetTermPassLen\_JPEG2K 1045  
ippiEncodeHuffman8x8\_ACFirst\_JPEG 996  
ippiEncodeHuffman8x8\_ACRrefine\_JPEG 997  
ippiEncodeHuffman8x8\_DCFirst\_JPEG 994  
ippiEncodeHuffman8x8\_DCRefine\_JPEG 995  
ippiEncodeHuffman8x8\_Direct\_JPEG 990  
ippiEncodeHuffman8x8\_JPEG 989  
ippiEncodeHuffmanOne\_JPEG 1013  
ippiEncodeHuffmanRawTableInit\_JPEG 984  
ippiEncodeHuffmanRow\_JPEG 1015  
ippiEncodeHuffmanSpecFree\_JPEG 987  
ippiEncodeHuffmanSpecGetBufSize\_JPEG 985  
ippiEncodeHuffmanSpecInit\_JPEG 986  
ippiEncodeHuffmanSpecInitAlloc\_JPEG 986  
ippiEncodeHuffmanStateFree\_JPEG 989  
ippiEncodeHuffmanStateGetBufSize\_JPEG 987  
ippiEncodeHuffmanStateInit\_JPEG 988  
ippiEncodeHuffmanStateInitAlloc\_JPEG 988  
ippiEncodeInitAlloc\_JPEG2K 1041  
ippiEncodeLoadCodeBlock\_JPEG2K 1042  
ippiEncodeStoreBits\_JPEG2K 1044  
ippiEncodeTableInitAlloc 1237  
ippiErode 397  
ippiErode3x3 394  
ippiErodeBorderReplicate 404  
ippiExp 149  
ippiExpandPlane\_H264 1379  
ippiFFTFree 549  
ippiFFTFwd 550  
ippiFFTGetBufSize 550  
ippiFFTInitAlloc 548  
ippiFFTInv 554  
ippiFilter 458  
ippiFilter\_Round16s 462  
ippiFilter\_Round32f 462  
ippiFilter\_Round32s 462  
ippiFilter32f 460  
ippiFilter8x8\_H261 1318  
ippiFilterBilateral 444  
ippiFilterBilateralGetBufSize 442  
ippiFilterBilateralInit 443  
ippiFilterBlockBoundaryHorEdge\_H263 1335  
ippiFilterBlockBoundaryVerEdge\_H263 1335  
ippiFilterBox 430, 432  
ippiFilterColumn 466  
ippiFilterColumn32f 468

ippiFilterColumnPipeline 476  
ippiFilterColumnPipeline\_Low 476  
ippiFilterColumnPipelineGetBufferSize 473  
ippiFilterColumnPipelineGetBufferSize\_Low 473  
ippiFilterDeblocking16x16\_HorEdge\_H263 1336  
ippiFilterDeblocking16x16\_VerEdge\_H263 1336  
ippiFilterDeblocking8x8HorEdge\_H263 1335  
ippiFilterDeblocking8x8HorEdge\_MPEG4 1300  
ippiFilterDeblocking8x8VerEdge\_H263 1335  
ippiFilterDeblocking8x8VerEdge\_MPEG4 1300  
ippiFilterDeblockingChroma\_HorEdge\_AVS 1498  
ippiFilterDeblockingChroma\_HorEdge\_H264 1453  
ippiFilterDeblockingChroma\_HorEdge\_VC1 1538  
ippiFilterDeblockingChroma\_VerEdge\_AVS 1496  
ippiFilterDeblockingChroma\_VerEdge\_H264 1447  
ippiFilterDeblockingChroma\_VerEdge\_MBAFF\_H264 1451  
ippiFilterDeblockingChroma\_VerEdge\_VC1 1538  
ippiFilterDeblockingChroma422HorEdge\_H264 1453  
ippiFilterDeblockingChroma422VerEdge\_H264 1447  
ippiFilterDeblockingChroma422VerEdge\_MBAFF\_H264 1451  
ippiFilterDeblockingChromaHorEdge\_H264 1453  
ippiFilterDeblockingChromaVerEdge\_H264 1447  
ippiFilterDeblockingLuma\_HorEdge\_AVS 1494  
ippiFilterDeblockingLuma\_HorEdge\_H264 1445  
ippiFilterDeblockingLuma\_HorEdge\_VC1 1537  
ippiFilterDeblockingLuma\_VerEdge\_AVS 1493  
ippiFilterDeblockingLuma\_VerEdge\_H264 1443  
ippiFilterDeblockingLuma\_VerEdge\_MBAFF\_H264 1444  
ippiFilterDeblockingLuma\_VerEdge\_VC1 1536  
ippiFilterDenoiseAdaptive 1210  
ippiFilterDenoiseAdaptiveFree 1209  
ippiFilterDenoiseAdaptiveInitAlloc 1209  
ippiFilterDenoiseCAST 1207  
ippiFilterDenoiseCASTInit 1206  
ippiFilterDenoiseMosquito 1213  
ippiFilterDenoiseMosquitoFree 1212  
ippiFilterDenoiseMosquitoInitAlloc 1212  
ippiFilterDenoiseSmooth 1208  
ippiFilterDeringingSmooth8x8\_MPEG4 1301  
ippiFilterDeringingThreshold\_MPEG4 1301  
ippiFilterGauss 511  
ippiFilterGaussBorder 540  
ippiFilterGetBufSize 460  
ippiFilterHipass 515  
ippiFilterLaplace 510  
ippiFilterLaplacianBorder 538  
ippiFilterLowpass 516  
ippiFilterLowpassBorder 541  
ippiFilterLowpassGetBufferSize 525  
ippiFilterMax 437  
ippiFilterMaxBorderReplicate 441  
ippiFilterMaxGetBufferSize 439  
ippiFilterMedian 450  
ippiFilterMedianColor 456  
ippiFilterMedianCross 454  
ippiFilterMedianGetBufferSize 449  
ippiFilterMedianHoriz 452  
ippiFilterMedianVert 453  
ippiFilterMedianWeightedCenter3x3 455  
ippiFilterMin 435  
ippiFilterMinBorderReplicate 439  
ippiFilterMinGetBufferSize 438  
ippiFilterPrewittHoriz 495  
ippiFilterPrewittVert 496  
ippiFilterRobertsDown 508  
ippiFilterRobertsUp 509  
ippiFilterRoundGetBufSize16s 464  
ippiFilterRoundGetBufSize32f 464  
ippiFilterRoundGetBufSize32s 464  
ippiFilterRow 469  
ippiFilterRow32f 471  
ippiFilterRowBorderPipeline 474  
ippiFilterRowBorderPipeline\_Low 474  
ippiFilterRowBorderPipelineGetBufferSize 472  
ippiFilterRowBorderPipelineGetBufferSize\_Low 472  
ippiFilterScharrHoriz 498  
ippiFilterScharrHorizBorder 526  
ippiFilterScharrHorizGetBufferSize 518  
ippiFilterScharrVert 499  
ippiFilterScharrVertBorder 528  
ippiFilterScharrVertGetBufferSize 519  
ippiFilterSharpen 517  
ippiFilterSobelCross 507  
ippiFilterSobelCrossBorder 537  
ippiFilterSobelCrossGetBufferSize 523  
ippiFilterSobelHoriz 500  
ippiFilterSobelHorizBorder 529  
ippiFilterSobelHorizGetBufferSize 520  
ippiFilterSobelHorizSecond 503  
ippiFilterSobelHorizSecondBorder 533  
ippiFilterSobelHorizSecondGetBufferSize 521  
ippiFilterSobelNegVertBorder 531  
ippiFilterSobelNegVertGetBufferSize 520  
ippiFilterSobelVert 502  
ippiFilterSobelVertBorder 531  
ippiFilterSobelVertGetBufferSize 520  
ippiFilterSobelVertSecond 504  
ippiFilterSobelVertSecondBorder 534  
ippiFilterSobelVertSecondGetBufferSize 522  
ippiFilterWiener 481  
ippiFilterWienerGetBufferSize 480  
ippiFindPeaks3x3 628  
ippiFindPeaks3x3GetBufferSize 628  
ippiFLCEncode4x4\_JPEGXR 1083  
ippiFloodFill\_Grad 840  
ippiFloodFill\_Range 843  
ippiFloodFillGetSize 837  
ippiFloodFillGetSize\_Grad 838  
ippiForegroundGaussian 891  
ippiForegroundGaussianFree 890  
ippiForegroundGaussianInitAlloc 889  
ippiForegroundHistogram 885  
ippiForegroundHistogramFree 885  
ippiForegroundHistogramInitAlloc 883  
ippiForegroundHistogramUpdate 888  
ippiFrameFieldSAD16x16 1182  
ippiFree 57  
ippiFreeHuffmanTable\_DV 1257  
ippiFVLCDcode4x4\_JPEGXR 1084  
ippiGammaFwd 341  
ippiGammaInv 343  
ippiGenScaleLevel8x8\_H264 1466

ippiGenSobelKernel 525  
ippiGetAffineBound 759  
ippiGetAffineQuad 758  
ippiGetAffineTransform 760  
ippiGetBilinearBound 783  
ippiGetBilinearQuad 782  
ippiGetBilinearTransform 784  
ippiGetCentralMoment 638  
ippiGetDiff16x16 1137  
ippiGetDiff16x16B 1144  
ippiGetDiff16x8 1138  
ippiGetDiff16x8B 1145  
ippiGetDiff4x4 1143  
ippiGetDiff8x16 1140  
ippiGetDiff8x16B 1147  
ippiGetDiff8x4 1141  
ippiGetDiff8x4B 1148  
ippiGetDiff8x8 1139  
ippiGetDiff8x8B 1146  
ippiGetDistanceTransformMask 830  
ippiGetHaarClassifierSize 897  
ippiGetHuffmanStatistics8x8\_ACFirst\_JPEG 992  
ippiGetHuffmanStatistics8x8\_ACRrefine\_JPEG 993  
ippiGetHuffmanStatistics8x8\_DCFfirst\_JPEG 992  
ippiGetHuffmanStatistics8x8\_JPEG 991  
ippiGetHuffmanStatisticsOne\_JPEG 1012  
ippiGetHuMoments 640  
ippiGetLibVersion 53  
ippiGetNormalizedCentralMoment 639  
ippiGetNormalizedSpatialMoment 638  
ippiGetPerspectiveBound 771  
ippiGetPerspectiveQuad 770  
ippiGetPerspectiveTransform 771  
ippiGetPyramidDownROI 864  
ippiGetResizeFract 715  
ippiGetResizeRect 713  
ippiGetRotateBound 738  
ippiGetRotateQuad 737  
ippiGetRotateShift 736  
ippiGetShearBound 750  
ippiGetShearQuad 749  
ippiGetSpatialMoment 637  
ippiGradientColorToGray 835  
ippiGrayDilateBorder 417, 418  
ippiHaarClassifierFree 896  
ippiHaarClassifierInitAlloc 894  
ippiHistogramEven 613  
ippiHistogramRange 611  
ippiHLSToBGR 282  
ippiHLSToRGB 279  
ippiHoughLine 824  
ippiHoughLine\_Region 825  
ippiHoughLineGetSize 823  
ippiHSVToRGB 284  
ippiHuffmanDecodeSegment\_DV 1251  
ippiHuffmanDecodeSegment\_DV100 1255  
ippiHuffmanDecodeSegmentOnePass\_DV 1256  
ippiHuffmanRunLevelTableInitAlloc 1104  
ippiHuffmanTableFree 1105  
ippiHuffmanTableInitAlloc 1103  
ippiICTFwd\_JPEG2K 1059  
ippiICTInv\_JPEG2K 1060  
ippiImageJaehne 93  
ippiImageRamp 95  
ippiInitAllocHuffmanTable\_DV 1247  
ippiInpaint 873  
ippiInpaintFree 873  
ippiInpaintInitAlloc 871  
ippiIntegral 597  
ippiInterpolateAverage16x16 1131  
ippiInterpolateAverage16x8 1131  
ippiInterpolateAverage8x4 1131  
ippiInterpolateAverage8x8 1131  
ippiInterpolateBlock\_H264 1400  
ippiInterpolateChroma\_H264 1389  
ippiInterpolateChromaBlock\_H264 1397  
ippiInterpolateChromaBottom\_H264 1396  
ippiInterpolateChromaTop\_H264 1394  
ippiInterpolateICBicubicBlock\_VC1 1525  
ippiInterpolateICBilinearBlock\_VC1 1523  
ippiInterpolateLuma\_H264 1380  
ippiInterpolateLumaBlock\_AVS 1486  
ippiInterpolateLumaBlock\_H264 1388  
ippiInterpolateLumaBottom\_H264 1386  
ippiInterpolateLumaTop\_H264 1383  
ippiInterpolateQPBicubic\_VC1 1522  
ippiInterpolateQPBilinear\_VC1 1520  
ippiJoin422LS\_MCU 982  
ippiLabelMarkers 875  
ippiLabelMarkersGetBufferSize 875  
ippiLabToBGR 275  
ippiFilterGaussGetBufferSize 524  
ippiFilterLaplacianGetBufferSize 523  
ippiLn 147  
ippiLShiftC 167  
ippiLUT 348  
ippiLUT\_Cubic 355  
ippiLUT\_Linear 351  
ippiLUTPalette 358  
ippiLUTPaletteSwap 358  
ippiLUVToRGB 272  
ippiMagnitude 567  
ippiMagnitudePack 568  
ippiMalloc 56  
ippiMax 619  
ippiMaxEvery 626  
ippiMaxIndx 621  
ippiMC16x16 1107  
ippiMC16x16B 1118  
ippiMC16x4 1115  
ippiMC16x4B 1128  
ippiMC16x8 1108  
ippiMC16x8B 1119  
ippiMC16x8BUV 1129  
ippiMC16x8UV 1116  
ippiMC2x2 1115  
ippiMC2x2B 1127  
ippiMC2x4 1113  
ippiMC2x4B 1125  
ippiMC4x2 1114  
ippiMC4x2B 1126  
ippiMC4x4 1113

ippiMC4x4B 1124	ippiOBMC8x8QP_MPEG4 1286
ippiMC4x8 1112	ippiOpticalFlowPyrLKFree 849
ippiMC4x8B 1123	ippiOpticalFlowPyrLKInitAlloc 848
ippiMC8x16 1109	ippiOr 157
ippiMC8x16B 1120	ippiOrC 159
ippiMC8x4 1111	ippiPackToCplxExtend 573
ippiMC8x4B 1122	ippiPCTFwd_HDP 1073
ippiMC8x8 1109	ippiPCTFwd_JPEGXR 1076
ippiMC8x8B 1121	ippiPCTFwd16x16_HDP 1073
ippiMean 603	ippiPCTFwd16x16_JPEGXR 1076
ippiMean_StdDev 606	ippiPCTFwd8x16_HDP 1073
ippiMeanAbsDev16x16 1161	ippiPCTFwd8x16_JPEGXR 1076
ippiMeanAbsDev8x8 1160	ippiPCTFwd8x8_HDP 1073
ippiMedian 1195	ippiPCTFwd8x8_JPEGXR 1076
ippiMin 616	ippiPCTInv_HDP 1074
ippiMinEigenVal 821	ippiPCTInv_JPEGXR 1077
ippiMinEigenValGetBufferSize 820	ippiPCTInv16x16_HDP 1074
ippiMinEvery 627	ippiPCTInv16x16_JPEGXR 1077
ippiMinIndx 618	ippiPCTInv8x16_HDP 1074
ippiMinMax 622	ippiPCTInv8x16_JPEGXR 1077
ippiMinMaxIndx 624	ippiPCTInv8x8_HDP 1074
ippiMirror 726	ippiPCTInv8x8_JPEGXR 1077
ippiMomentFree 634	ippiPhase 570
ippiMomentGetStateSize 634	ippiPhasePack 571
ippiMomentInit 635	ippiPredictIntra_16x16_H264 1374
ippiMomentInitAlloc 633	ippiPredictIntra_4x4_H264 1371
ippiMoments 635	ippiPredictIntraChroma8x8_H264 1375
ippiMorpAdvGetSize 408	ippiPutIntraBlock 1238
ippiMorpGrayGetSize 417	ippiPutNonIntraBlock 1239
ippiMorphAdvFree 406	ippiPyramidFree 860
ippiMorphAdvInit 407, 416	ippiPyramidInitAlloc 859
ippiMorphAdvInitAlloc 405, 414	ippiPyramidLayerDown 866
ippiMorphBlackhatBorder 412	ippiPyramidLayerDownFree 862
ippiMorphGradientBorder 413	ippiPyramidLayerUp 868
ippiMorphGrayFree 415	ippiPyramidLayerUpFree 864
ippiMorphologyFree 400	ippiPyramidLayerUpInitAlloc 862
ippiMorphologyGetSize 401	ippiPyrDown 856
ippiMorphologyInit 400	ippiPyrDownGetBufSize 855
ippiMorphologyInitAlloc 399	ippiPyrUp 857
ippiMorphOpenBorder 408, 410	ippiPyrUpGetBufSize 855
ippiMorphReconstructErode 423	ippiQualityIndex 665
ippiMorphReconstructGetBufferSize 420	ippiQuant_MPEG2 1235
ippiMorphTophatBorder 411	ippiQuantFwd8x8_JPEG 955
ippiMul 118	ippiQuantFwdRawTableInit_JPEG 954
ippiMulC 120	ippiQuantFwdTableInit_JPEG 955
ippiMulCScale 124	ippiQuantInter_H263 1344
ippiMulPack 562	ippiQuantInter_MPEG4 1305
ippiMulPackConj 566	ippiQuantInterGetSize_MPEG4 1304
ippiMulScale 123	ippiQuantInterInit_MPEG4 1304
ippiNorm_Inf 642	ippiQuantInterNonuniform_VC1 1554
ippiNorm_L1 644	ippiQuantInterUniform_VC1 1554
ippiNorm_L2 646	ippiQuantIntra_H263 1343
ippiNormDiff_Inf 649	ippiQuantIntra_MPEG2 1235
ippiNormDiff_L1 651	ippiQuantIntra_MPEG4 1305
ippiNormDiff_L2 654	ippiQuantIntraGetSize_MPEG4 1304
ippiNormRel_Inf 656	ippiQuantIntraInit_MPEG4 1304
ippiNormRel_L1 658	ippiQuantIntraNonuniform_VC1 1552
ippiNormRel_L2 661	ippiQuantIntraUniform_VC1 1552
ippiNot 164	ippiQuantInv_DV 1257
ippiOBMC16x16HP_MPEG4 1286	ippiQuantInv_MPEG2 1226
ippiOBMC8x8HP_MPEG4 1286	ippiQuantInv8x8_JPEG 959

ippiQuantInvInter\_H263 1328  
ippiQuantInvInter\_MPEG4 1293  
ippiQuantInvInterGetSize\_MPEG4 1293  
ippiQuantInvInterInit\_MPEG4 1292  
ippiQuantInvInterNonuniform\_VC1 1542  
ippiQuantInvInterUniform\_VC1 1542  
ippiQuantInvIntra\_H263 1326  
ippiQuantInvIntra\_MPEG2 1226  
ippiQuantInvIntra\_MPEG4 1293  
ippiQuantInvIntraGetSize\_MPEG4 1293  
ippiQuantInvIntraInit\_MPEG4 1292  
ippiQuantInvIntraNonuniform\_VC1 1539  
ippiQuantInvIntraUniform\_VC1 1539  
ippiQuantInvTableInit\_JPEG 958  
ippiQuantizeResidual4x4Fwd\_H264 1462  
ippiQuantLuma8x8\_H264 1465  
ippiQuantLuma8x8Inv\_H264 1479  
ippiQuantWeightBlockInv\_DV 1258  
ippiQuantWeightBlockInv\_DV100 1262  
ippiRangeMapping\_VC1 1545  
ippiRCTFwd\_JPEG2K 1055  
ippiRCTInv\_JPEG2K 1057  
ippiReconstructChroma422Inter4x4\_H264High 1417  
ippiReconstructChroma422Intra4x4\_H264High 1425  
ippiReconstructChroma422IntraHalf4x4\_H264High 1423  
ippiReconstructChromaInter\_AVS 1492  
ippiReconstructChromaInter4x4\_H264High 1414  
ippiReconstructChromaInter4x4MB\_H264 1414  
ippiReconstructChromaInterMB\_H264 1409  
ippiReconstructChromaIntra\_AVS 1491  
ippiReconstructChromaIntra4x4\_H264High 1421  
ippiReconstructChromaIntra4x4MB\_H264 1421  
ippiReconstructChromaIntraHalf4x4High\_H264 1419  
ippiReconstructChromaIntraHalves4x4MB\_H264 1419  
ippiReconstructChromaIntraHalvesMB\_H264 1411  
ippiReconstructChromaIntraMB\_H264 1412  
ippiReconstructCoeffsInter\_H261 1314  
ippiReconstructCoeffsInter\_H263 1338  
ippiReconstructCoeffsInter\_MPEG4 1299  
ippiReconstructCoeffsIntra\_H261 1313  
ippiReconstructCoeffsIntra\_H263 1337  
ippiReconstructDCTBlock\_MPEG1 1220  
ippiReconstructDCTBlock\_MPEG2 1223  
ippiReconstructDCTBlockIntra\_MPEG1 1222  
ippiReconstructDCTBlockIntra\_MPEG2 1224  
ippiReconstructLumaInter\_AVS 1490  
ippiReconstructLumaInter4x4\_H264High 1430  
ippiReconstructLumaInter4x4MB\_H264 1430  
ippiReconstructLumaInter8x8\_H264High 1435  
ippiReconstructLumaInter8x8MB\_H264 1435  
ippiReconstructLumaIntera4x4MB\_H264 1434  
ippiReconstructLumaInteraHalf4x4MB\_H264 1432  
ippiReconstructLumaInterMB\_H264 1427  
ippiReconstructLumaIntra\_AVS 1489  
ippiReconstructLumaIntra16x16\_H264High 1440  
ippiReconstructLumaIntra16x16MB\_H264 1439, 1440  
ippiReconstructLumaIntra4x4\_H264High 1434  
ippiReconstructLumaIntra8x8\_H264High 1438  
ippiReconstructLumaIntra8x8MB\_H264 1438  
ippiReconstructLumaIntraHalf4x4\_H264High 1432  
ippiReconstructLumaIntraHalf8x8\_H264High 1436  
ippiReconstructLumaIntraHalf8x8MB\_H264 1436  
ippiReconstructLumaIntraHalfMB\_H264 1429  
ippiReconstructPredFirstRow\_JPEG 1011  
ippiReconstructPredRow 1011  
ippiRectStsDev 608  
ippiReduceBits 346  
ippiRemap 728  
ippiResample\_H263 1330  
ippiResampleRow 102  
ippiResampleRowGetBorderWidth 102  
ippiResampleRowGetSize 100  
ippiResampleRowInit 101  
ippiResampleRowReplicateBorder 102  
ippiResize 697  
ippiResizeCCRotate 1191  
ippiResizeCenter 701  
ippiResizeFilter 725  
ippiResizeFilterGetSize 724  
ippiResizeFilterInit 724  
ippiResizeGetBufSize 712  
ippiResizeShift 716  
ippiResizeSqrPixel 705  
ippiResizeSqrPixelGetBufSize 714  
ippiResizeYUV420 722  
ippiResizeYUV420GetBufSize 723  
ippiResizeYUV422 720  
ippiRGB555ToYCbCr\_JPEG 920  
ippiRGB565ToRGB 300  
ippiRGB565ToYCbCr\_JPEG 920  
ippiRGB565ToYUV420 214  
ippiRGB565ToYUV422 209  
ippiRGBToCbYCr422 236  
ippiRGBToCbYCr422Gamma 236  
ippiRGBToGray 294  
ippiRGBToHLS 278  
ippiRGBToHSV 283  
ippiRGBToLUV 270  
ippiRGBToRGB565 299  
ippiRGBToXYZ 268  
ippiRGBToY\_JPEG 915  
ippiRGBToYCbCr 220  
ippiRGBToYCbCr\_JPEG 918  
ippiRGBToYCbCr411\_JPEG 928  
ippiRGBToYCbCr411LS\_MCU 938  
ippiRGBToYCbCr420 246, 250  
ippiRGBToYCbCr422 229  
ippiRGBToYCbCr422\_JPEG 925  
ippiRGBToYCbCr422LS\_MCU 936  
ippiRGBToYCbCr444LS\_MCU 935  
ippiRGBToYCC 276  
ippiRGBToYCoCg 285  
ippiRGBToYUV 203  
ippiRGBToYUV420 210  
ippiRGBToYUV422 206  
ippiRotate 733  
ippiRotateCenter 741  
ippiRShiftC 165  
ippiSAD16x16 1163  
ippiSAD16x16Blocks4x4 1173  
ippiSAD16x16Blocks8x8 1172

ippiSAD16x16xN 1164	ippiSSD4x4 1155
ippiSAD16x16xNI 1164	ippiSSD8x8 1154
ippiSAD16x8 1165	ippiSub 127
ippiSAD4x4 1170	ippiSub128_JPEG 967
ippiSAD4x4xN 1171	ippiSub16x16 1150, 1151
ippiSAD4x4xNI 1171	ippiSub8x8 1150
ippiSAD4x8 1169	ippiSubC 129
ippiSAD8x16 1166	ippiSubSAD8x8 1151
ippiSAD8x4 1169	ippiSum 595
ippiSAD8x8 1167	ippiSumsDiff16x16Blocks4x4 1183
ippiSAD8x8xN 1168	ippiSumsDiff16x16Blocks4x4_8u16s_C1 1183
ippiSAD8x8xNI 1168	ippiSumsDiff8x8Blocks4x4 1184
ippiSampleDown411LS_MCU 978	ippiSumsDiff8x8Blocks4x4_8u16s_C1 1184
ippiSampleDown422LS_MCU 978	ippiSumWindowColumn 434
ippiSampleDown444LS_MCU 977	ippiSumWindowRow 433
ippiSampleDownH2V1_JPEG 969	ippiSuperSampling 718
ippiSampleDownH2V2_JPEG 970	ippiSuperSamplingGetBufSize 720
ippiSampleDownRowH2V1_Box_JPEG 970	ippiSwapChannels 88
ippiSampleDownRowH2V2_Box_JPEG 973	ippiTextureDecodeBlock 1071
ippiSampleLine 96	ippiTextureDecodeBlockFromRGB 1070
ippiSampleUp411LS_MCU 980	ippiTextureDecodeBlockFromRGBA 1069
ippiSampleUp444LS_MCU 979	ippiTextureDecodeBlockToRG 1070
ippiSampleUpH2V1_JPEG 973	ippiTextureEncodeBlock 1067
ippiSampleUpH2V2_JPEG 975	ippiTextureEncodeBlock4x4FromRGBA 1068
ippiSampleUpRowH2V1_Triangle_JPEG 976	ippiTextureEncodeBlockFromRG 1067
ippiSampleUpRowH2V2_Triangle_JPEG 976	ippiTextureEncodeBlockFromRGBA 1066
ippiSAT8x8D 1181	ippiTextureEncodeBlockFromYCoCg 1072
ippiSATD16x16 1174	ippiThreshold 364
ippiSATD16x8 1175	ippiThreshold_GT 366
ippiSATD4x4 1180	ippiThreshold_GTVal 372
ippiSATD4x8 1179	ippiThreshold_LT 368
ippiSATD8x16 1176	ippiThreshold_LTVal 374
ippiSATD8x4 1178	ippiThreshold_LTValGTVal 376
ippiSATD8x8 1177	ippiThreshold_Val 370
ippiSBGRToYCoCg 287	ippiTiltedHaarClassifierInitAlloc 895
ippiSBGRToYCoCg_Rev 291	ippiTiltedIntegral 601
ippiScale 64	ippiTiltedRectStdDev 609
ippiScanFwd 1188	ippiTiltedSqrIntegral 602
ippiScanInv 1187	ippiToneMapLinear 361
ippiSegmentGradient 881	ippiToneMapMean 361
ippiSegmentGradientGetBufferSize 880	ippiTransform4x4Fwd_VC1_16s_C1IR 1547
ippiSegmentWatershed 877	ippiTransform4x4Fwd_VC1_16s_C1R 1547
ippiSegmentWatershedGetBufferSize 877	ippiTransform4x4Inv_VC1_16s_C1IR 1511
ippiSet 67	ippiTransform4x4Inv_VC1_16s_C1R 1511
ippiShear 745	ippiTransform4x8Fwd_VC1_16s_C1IR 1547
ippiSmoothingChroma_HorEdge_VC1 1534	ippiTransform4x8Fwd_VC1_16s_C1R 1547
ippiSmoothingChroma_VerEdge_VC1 1532	ippiTransform4x8Inv_VC1_16s_C1IR 1511
ippiSmoothingLuma_HorEdge_VC1 1529	ippiTransform4x8Inv_VC1_16s_C1R 1511
ippiSmoothingLuma_VerEdge_VC1 1527	ippiTransform8x4Fwd_VC1_16s_C1IR 1547
ippiSpatialInterpolation_H263 1334	ippiTransform8x4Fwd_VC1_16s_C1R 1547
ippiSplit422LS_MCU 981	ippiTransform8x4Inv_VC1_16s_C1IR 1511
ippiSqr 143	ippiTransform8x4Inv_VC1_16s_C1R 1511
ippiSqrDiff16x16 1152	ippiTransform8x8Fwd_VC1_16s_C1IR 1547
ippiSqrDiff16x16B 1153	ippiTransform8x8Fwd_VC1_16s_C1R 1547
ippiSqrDistanceFull_Norm 670	ippiTransform8x8Inv_VC1_16s_C1IR 1511
ippiSqrDistanceSame_Norm 673	ippiTransform8x8Inv_VC1_16s_C1R 1511
ippiSqrDistanceValid_Norm 675	ippiTransformDequantChromaDC_H264 1365
ippiSqrIntegral 599	ippiTransformDequantChromaDC_SISP_H264 1370
ippiSqrt 145	ippiTransformDequantLumaDC_H264 1364
ippiSRHNInitAlloc_PSF2x2 909	ippiTransformLuma8x8Fwd_H264 1464
ippiSRHNInitAlloc_PSF3x3 909	ippiTransformLuma8x8InvAddPred_H264 1480

iippiWTFwdGetBufSize 795  
iippiWTFwdInitAlloc 793  
iippiWTFwdRow\_B53\_JPEG2K 1018  
iippiWTFwdRow\_D97\_JPEG2K 1026  
iippiWTInv 802  
iippiWTInv\_B53\_JPEG2K 1036  
iippiWTInv\_D97\_JPEG2K 1039  
iippiWTInvCol\_B53\_JPEG2K 1023  
iippiWTInvCol\_D97\_JPEG2K 1031  
iippiWTInvColLift\_B53\_JPEG2K 1025  
iippiWTInvColLift\_D97\_JPEG2K 1032  
iippiWTInvFree 800  
iippiWTInvGetBufSize 801  
iippiWTInvInitAlloc 799  
iippiWTInvRow\_B53\_JPEG2K 1020  
iippiWTInvRow\_D97\_JPEG2K 1027  
iippiXor 161  
iippiXorC 162  
iippiXYZToRGB 269  
iippiYCbCr411 329  
iippiYCbCr411ToBGR 265  
iippiYCbCr411ToBGR\_JPEG 931  
iippiYCbCr411ToBGR555 267  
iippiYCbCr411ToBGR555LS\_MCU 951  
iippiYCbCr411ToBGR565 267  
iippiYCbCr411ToBGR565LS\_MCU 951  
iippiYCbCr411ToBGRLS\_MCU 950  
iippiYCbCr411ToRGB\_JPEG 929  
iippiYCbCr411ToRBLG\_MCU 947  
iippiYCbCr411ToYCbCr420 332  
iippiYCbCr411ToYCbCr422 330  
iippiYCbCr411ToYCrCb420 334  
iippiYCbCr411ToYCrCb422 331  
iippiYCbCr420 315  
iippiYCbCr420To422\_Interlace 319  
iippiYCbCr420ToBGR 256  
iippiYCbCr420ToBGR\_709CSC 257  
iippiYCbCr420ToBGR\_709HDTV 258  
iippiYCbCr420ToBGR444Dither 262  
iippiYCbCr420ToBGR555Dither 262  
iippiYCbCr420ToBGR565Dither 262  
iippiYCbCr420ToCbYCr422 320  
iippiYCbCr420ToCbYCr422\_Interlace 320  
iippiYCbCr420ToRBG 247  
iippiYCbCr420ToRGB444Dither 249  
iippiYCbCr420ToRGB555Dither 249  
iippiYCbCr420ToRGB565Dither 249  
iippiYCbCr420ToYCbCr411 324  
iippiYCbCr420ToYCbCr422 316  
iippiYCbCr420ToYCbCr422\_Filter 317  
iippiYCbCr420ToYCrCb420 321  
iippiYCbCr420ToYCrCb420\_Filter 322  
iippiYCbCr422 301  
iippiYCbCr422To420\_Interlace 305  
iippiYCbCr422ToBGR 234  
iippiYCbCr422ToBGR\_JPEG 928  
iippiYCbCr422ToBGR444 244  
iippiYCbCr422ToBGR444Dither 245  
iippiYCbCr422ToBGR555 244  
iippiYCbCr422ToBGR555Dither 245  
iippiYCbCr422ToBGR555LS\_MCU 950

ippiYCbCr422ToBGR565 244  
ippiYCbCr422ToBGR565Dither 245  
ippiYCbCr422ToBGR565LS\_MCU 950  
ippiYCbCr422ToBGR565LS\_MCU 949  
ippiYCbCr422ToCbYCr422 302  
ippiYCbCr422ToRGB 230, 231  
ippiYCbCr422ToRGB\_JPEG 926  
ippiYCbCr422ToRGB444 241  
ippiYCbCr422ToRGB444Dither 242  
ippiYCbCr422ToRGB555 241  
ippiYCbCr422ToRGB555Dither 242  
ippiYCbCr422ToRGB565 241  
ippiYCbCr422ToRGB565Dither 242  
ippiYCbCr422ToRGB565LS\_MCU 946  
ippiYCbCr422ToYCbCr411 307  
ippiYCbCr422ToYCbCr420 303  
ippiYCbCr422ToYCbCr420 306  
ippiYCbCr444ToBGR555LS\_MCU 948  
ippiYCbCr444ToBGR565LS\_MCU 948  
ippiYCbCr444ToBGR565LS\_MCU 947  
ippiYCbCr444ToBGR565LS\_MCU 945  
ippiYCbCrToBGR 222  
ippiYCbCrToBGR\_709CSC 223  
ippiYCbCrToBGR\_JPEG 922  
ippiYCbCrToBGR444 226  
ippiYCbCrToBGR444Dither 228  
ippiYCbCrToBGR555 226  
ippiYCbCrToBGR555\_JPEG 924  
ippiYCbCrToBGR555Dither 228  
ippiYCbCrToBGR565 226  
ippiYCbCrToBGR565\_JPEG 924  
ippiYCbCrToBGR565Dither 228  
ippiYCbCrToRGB 221  
ippiYCbCrToRGB\_JPEG 919  
ippiYCbCrToRGB444 224  
ippiYCbCrToRGB444Dither 225  
ippiYCbCrToRGB555 224  
ippiYCbCrToRGB555\_JPEG 921  
ippiYCbCrToRGB555Dither 225  
ippiYCbCrToRGB565 224  
ippiYCbCrToRGB565\_JPEG 921  
ippiYCbCrToRGB565Dither 225  
ippiYCCK444ToCMYKLS\_MCU 952  
ippiYCCKToCMYK\_JPEG 933  
ippiYCCKToCMYK411LS\_MCU 953  
ippiYCCKToCMYK422LS\_MCU 953  
ippiYCCToRGB 277  
ippiYCiCgToSBGR 289  
ippiYCoCgToBGR 288  
ippiYCoCgToBGR\_Rev 292  
ippiYCoCgToRGB 286  
ippiYCoCgToSBGR\_Rev 293  
ippiYCrCb411ToYCbCr422\_16x4x5MB\_DV 1273  
ippiYCrCb411ToYCbCr422\_5MBDV 1272  
ippiYCrCb411ToYCbCr422\_8x8MB\_DV 1273  
ippiYCrCb411ToYCbCr422\_EdgeDV 1274  
ippiYCrCb411ToYCbCr422\_ZoomOut2\_5MBDV 1272  
ippiYCrCb411ToYCbCr422\_ZoomOut2\_EdgeDV 1274  
ippiYCrCb411ToYCbCr422\_ZoomOut4\_5MBDV 1272  
ippiYCrCb411ToYCbCr422\_ZoomOut4\_EdgeDV 1274  
ippiYCrCb411ToYCbCr422\_ZoomOut8\_5MBDV 1272

ippiYCrCb411ToYCbCr422\_ZoomOut8\_EdgeDV 1274  
ippiYCrCb420ToCbYCr422 327  
ippiYCrCb420ToRGB 251  
ippiYCrCb420ToYCbCr411 328  
ippiYCrCb420ToYCbCr420 327  
ippiYCrCb420ToYCbCr422 325  
ippiYCrCb420ToYCbCr422\_5MBDV 1275  
ippiYCrCb420ToYCbCr422\_8x8x5MB\_DV 1276  
ippiYCrCb420ToYCbCr422\_Filter 326  
ippiYCrCb420ToYCbCr422\_ZoomOut2\_5MBDV 1275  
ippiYCrCb420ToYCbCr422\_ZoomOut4\_5MBDV 1275  
ippiYCrCb420ToYCbCr422\_ZoomOut8\_5MBDV 1275  
ippiYCrCb422ToRGB 232  
ippiYCrCb422ToYCbCr411 310  
ippiYCrCb422ToYCbCr420 309  
ippiYCrCb422ToYCbCr422 308  
ippiYCrCb422ToYCbCr422\_10HalvesMB16x8\_DV100 1279  
ippiYCrCb422ToYCbCr422\_5MBDV 1277  
ippiYCrCb422ToYCbCr422\_8x4x5MB\_DV 1279  
ippiYCrCb422ToYCbCr422\_ZoomOut2\_5MBDV 1277  
ippiYCrCb422ToYCbCr422\_ZoomOut4\_5MBDV 1277  
ippiYCrCb422ToYCbCr422\_ZoomOut8\_5MBDV 1277  
ippiYUV420ToBGR 213  
ippiYUV420ToBGR444 218  
ippiYUV420ToBGR444Dither 219  
ippiYUV420ToBGR555 218  
ippiYUV420ToBGR555Dither 219  
ippiYUV420ToBGR565 218  
ippiYUV420ToBGR565Dither 219  
ippiYUV420ToRGB 211  
ippiYUV420ToRGB444 215  
ippiYUV420ToRGB444Dither 216  
ippiYUV420ToRGB555 215  
ippiYUV420ToRGB555Dither 216  
ippiYUV420ToRGB565 215  
ippiYUV420ToRGB565Dither 216  
ippiYUV422ToRGB 207  
ippiYUVToRGB 205  
ippiZigzagFwd8x8 97  
ippiZigzagInv8x8 99  
ippjGetLibVersion 914  
ippsApplyHaarClassifier 898

## J

Join422LS\_MCU 982  
JPEG 2000 Codec  
    entropy coding and decoding functions 1040  
    wavelet transform functions 1017  
JPEG Codec  
    color conversion functions 914  
    combined color conversion functions 934  
    combined quantization, DCT, and level shift functions 959  
    Huffman codec functions 983  
    level shift functions 966  
    library version 914  
    quantization functions 954  
    sampling functions 968



JPEG coding  
   color conversion functions 915, 917, 918, 919, 920, 921, 922, 923, 924, 925, 926, 927, 928, 929, 930, 931, 932, 933  
   combined DCT functions 960, 961, 963, 964, 965, 966  
   level shift functions 967  
 JPEG coding, lossless 1008  
 JPEG XR adaptive variable and fixed length coding 1078  
 JPEG XR adaptive VLC/FLC functions 1078  
 JPEG XR coding 1075, 1076, 1078  
 JPEG XR core transform 1076

## L

LabelMarkers 875  
 LabelMarkersGetBufferSize 875  
 LabToBGR 275  
 level shift functions, for JPEG codec 966  
 library version information 53  
 lines padding 48  
 Ln 147  
 logical functions  
   bitwise AND 154  
   bitwise NOT 164  
   bitwise OR 157  
   bitwise XOR 161  
   left shift 167  
   right shift 165  
 logical operations 153  
 LShiftC 167  
 LUT 348  
 LUT\_Cubic 355  
 LUT\_Linear 351  
 LUTPalette 358  
 LUTPaletteSwap 358  
 LUVToRGB 272

## M

macroblock 1240, 1310  
 macropixel 198  
 Magnitude 567  
 MagnitudePack 568  
 Malloc 56  
 mapping, of data range 64  
 mask, in median filtering 448  
 Max 619  
 MaxEvery 626  
 MaxIndx 621  
 MC16x16 1107  
 MC16x16B 1118  
 MC16x4 1115  
 MC16x4B 1128  
 MC16x8 1108  
 MC16x8B 1119  
 MC16x8BUV 1129  
 MC16x8UV 1116  
 MC2x2 1115  
 MC2x2B 1127  
 MC2x4 1113

MC2x4B 1125  
 MC4x2 1114  
 MC4x2B 1126  
 MC4x4 1113  
 MC4x4B 1124  
 MC4x8 1112  
 MC4x8B 1123  
 MC8x16 1109  
 MC8x16B 1120  
 MC8x4 1111  
 MC8x4B 1122  
 MC8x8 1109  
 MC8x8B 1121  
 MCU, minimum coded units 934  
 Mean 603  
 Mean\_StdDev 606  
 MeanAbsDev16x16 1161  
 MeanAbsDev8x8 1160  
 Median 1195  
 median filters 448  
 memory allocation 55  
 memory allocation functions  
   ippiFree 57  
   ippiMalloc 56  
 Min 616  
 MinEigenVal 821  
 MinEigenValGetBufferSize 820  
 MinEvery 627  
 MinIndx 618  
 MinMax 622  
 MinMaxIndx 624  
 Mirror 726  
 MomentFree 634  
 MomentGetStateSize 634  
 MomentInit 635  
 MomentInitAlloc 633  
 Moments 635  
 moments, of an image 632  
 MorphAdvFree 406  
 MorphAdvGetSize 408  
 MorphAdvInit 407  
 MorphAdvInitAlloc 405  
 MorphBlackhatBorder 412  
 MorphCloseBorder 410  
 MorphGradientBorder 413  
 MorphGrayFree 415  
 MorphGrayGetSize 417  
 MorphGrayInit 416  
 MorphGrayInitAlloc 414  
 morphological functions  
   DilateBorderReplicate 402  
   dilation 393  
   ErodeBorderReplicate 404  
   erosion 394  
   GrayDilateBorder 417, 418  
   MorphAdvFree 406  
   MorphAdvGetSize 408  
   MorphBlackhatBorder 412  
   MorphCloseBorder 410  
   MorphGradientBorder 413  
   MorphGrayFree 415

morphological functions (*continued*)

- MorphGrayGetSize 417
- MorphologyFree 400
- MorphOpenBorder 408
- MorphReconstructErode 423
- MorphReconstructGetBufferSize 420
- MorphTophatBorder 411
- MorphologyFree 400
- MorphologyGetSize 401
- MorphologyInit 400
- MorphologyInitAlloc 399
- MorphOpenBorder 408
- MorphReconstructErode 423
- MorphReconstructGetBufferSize 420
- MorphTophatBorder 411
- motion Analysis and object tracking 845
- motion analysis and object tracking functions
  - OpticalFlowPyrLK 849
  - OpticalFlowPyrLKFree 849
  - OpticalFlowPyrLKInitAlloc 848
  - UpdateMotionHistory 846
- motion compensation adding functions 1132, 1133, 1134
- motion compensation averaging functions 1134
- motion compensation copying functions 1130
- motion compensation interpolating functions 1131
- motion estimation subtracting functions 1150, 1151
- MPEG4 video decoder functions
  - CalcGlobalMV\_MPEG4 1291
  - ChangeSpriteBrightness\_MPEG4 1291
  - Copy16x16QP\_MPEG4 1285
  - Copy16x8QP\_MPEG4 1285
  - Copy8x8QP\_MPEG4 1285
  - DecodeCoeffsInter\_MPEG4 1297
  - DecodeCoeffsInterRVLCBack\_MPEG4 1298
  - DecodeCoeffsIntra\_MPEG4 1295
  - DecodeCoeffsIntraRVLCBack\_MPEG4 1296
  - DecodeDCIntra\_MPEG4 1294
  - EncodeCoeffsInter\_MPEG4 1308
  - EncodeCoeffsIntra\_MPEG4 1307
  - EncodeDCIntra\_MPEG4 1306
  - FilterDeblocking8x8HorEdge\_MPEG4 1300
  - FilterDeblocking8x8VerEdge\_MPEG4 1300
  - FilterDeringingSmooth8x8\_MPEG4 1301
  - FilterDeringingThreshold\_MPEG4 1301
  - OBMC16x16HP\_MPEG4 1286
  - OBMC8x8HP\_MPEG4 1286
  - OBMC8x8QP\_MPEG4 1286
  - QuantInter\_MPEG4 1305
  - QuantInterGetSize\_MPEG4 1304
  - QuantInterInit\_MPEG4 1304
  - QuantIntra\_MPEG4 1305
  - QuantIntraGetSize\_MPEG4 1304
  - QuantIntraInit\_MPEG4 1304
  - QuantInvInter\_MPEG4 1293
  - QuantInvInterGetSize\_MPEG4 1293
  - QuantInvInterInit\_MPEG4 1292
  - QuantInvIntra\_MPEG4 1293
  - QuantInvIntraGetSize\_MPEG4 1293
  - QuantInvIntraInit\_MPEG4 1292
  - ReconstructCoeffsInter\_MPEG4 1299
  - WarpChroma\_MPEG4 1290

MPEG4 video decoder functions (*continued*)

- WarpGetSize\_MPEG4 1289
- WarpInit\_MPEG4 1288
- WarpLuma\_MPEG4 1289
- Mul 118
- MulC 120
- MulCScale 124
- MulPack 562
- MulPackConj 566
- MulScale 123

## N

- naming conventions 31
- neighborhood operation 48
- Norm\_Inf 642
- Norm\_L1 644
- Norm\_L2 646
- NormDiff\_Inf 649
- NormDiff\_L1 651
- NormDiff\_L2 654
- NormRel\_Inf 656
- NormRel\_L1 658
- NormRel\_L2 661
- norms, of an image 641
- Not 164

## O

- object detection functions
  - HaarClassifierFree 896
  - HaarClassifierInitAlloc 894
  - TiltedHaarClassifierInitAlloc 895
- OBMC16x16HP\_MPEG4 1286
- OBMC8x8HP\_MPEG4 1286
- OBMC8x8QP\_MPEG4 1286
- operation models 48
- optical flow 847
- OpticalFlowPyrLK 849
- OpticalFlowPyrLKFree 849
- OpticalFlowPyrLKInitAlloc 848
- Or 157
- OrC 159

## P

- PackToCplxExtend 573
- pattern recognition 893
- PCTFwd\_HDP 1073
- PCTFwd\_JPEGXR 1076
- PCTFwd16x16\_HDP 1073
- PCTFwd16x16\_JPEGXR 1076
- PCTFwd8x16\_HDP 1073
- PCTFwd8x16\_JPEGXR 1076
- PCTFwd8x8\_HDP 1073
- PCTFwd8x8\_JPEGXR 1076
- PCTInv\_HDP 1074
- PCTInv\_JPEGXR 1077
- PCTInv16x16\_HDP 1074

PCTInv16x16\_JPEGXR 1077  
 PCTInv8x16\_HDP 1074  
 PCTInv8x16\_JPEGXR 1077  
 PCTInv8x8\_HDP 1074  
 PCTInv8x8\_JPEGXR 1077  
 Phase 570  
 PhasePack 571  
 photo core transform 1073  
 Photo YCC color model 187  
 pixel order images 47  
 planar order images 47  
 pOffset 1098  
 point operation 48  
 ppBitStream 1098  
 PredictIntra\_16x16\_H264 1374  
 PredictIntra\_4x4\_H264 1371  
 PredictIntraChroma8x8\_H264 1375  
 purple line 186  
 PutIntraBlock 1238  
 PutNonIntraBlock 1239  
 PyramidFree 860  
 PyramidInitAlloc 859  
 PyramidLayerDown 866  
 PyramidLayerDownFree 862  
 PyramidLayerUp 868  
 PyramidLayerUpFree 864  
 PyramidLayerUpInitAlloc 862  
 pyramids function  
     PyramidFree 860  
     PyramidLayerDownFree 862  
     PyramidLayerUp 868  
     PyramidLayerUpFree 864  
     PyramidLayerUpInitAlloc 862  
 PyrDown 856  
 PyrDownGetBufSize 855  
 PyrUp 857  
 PyrUpGetBufSize 855

## Q

quality factor, in JPEG compression 954  
 QualityIndex 665  
 Quant\_MPEG2 1235  
 QuantFwd8x8\_JPEG 955  
 QuantFwdRawTableInit\_JPEG 954  
 QuantFwdTableInit\_JPEG 955  
 QuantInter\_H263 1344  
 QuantInter\_MPEG4 1305  
 QuantInterGetSize\_MPEG4 1304  
 QuantInterInit\_MPEG4 1304  
 QuantInterNonuniform\_VC1 1554  
 QuantInterUniform\_VC1 1554  
 QuantIntra\_H263 1343  
 QuantIntra\_MPEG2 1235  
 QuantIntra\_MPEG4 1305  
 QuantIntraGetSize\_MPEG4 1304  
 QuantIntraInit\_MPEG4 1304  
 QuantIntraNonuniform\_VC1 1552  
 QuantIntraUniform\_VC1 1552  
 QuantInv\_DV 1257  
 QuantInv\_MPEG2 1226

QuantInv8x8\_JPEG 959  
 QuantInvInter\_H263 1328  
 QuantInvInter\_MPEG4 1293  
 QuantInvInterGetSize\_MPEG4 1293  
 QuantInvInterInit\_MPEG4 1292  
 QuantInvInterNonuniform\_VC1 1542  
 QuantInvInterUniform\_VC1 1542  
 QuantInvIntra\_H263 1326  
 QuantInvIntra\_MPEG2 1226  
 QuantInvIntra\_MPEG4 1293  
 QuantInvIntraGetSize\_MPEG4 1293  
 QuantInvIntraInit\_MPEG4 1292  
 QuantInvIntraNonuniform\_VC1 1539  
 QuantInvIntraUniform\_VC1 1539  
 QuantInvTableInit\_JPEG 958  
 quantization functions, for JPEG codec 954  
 quantization table 955  
 QuantizeResidual4x4Fwd\_H264 1462  
 QuantLuma8x8\_H264 1465  
 QuantLuma8x8Inv\_H264 1479  
 QuantWeightBlockInv\_DV 1258  
 QuantWeightBlockInv\_DV100 1262

## R

RangeMapping\_VC1 1545  
 RCTFwd\_JPEG2K 1055  
 RCTInv\_JPEG2K 1057  
 ReconstructChroma422Inter4x4\_H264High 1417  
 ReconstructChroma422Intra4x4\_H264High 1425  
 ReconstructChroma422IntraHalf4x4\_H264High 1423  
 ReconstructChromaInter\_AVS 1492  
 ReconstructChromaInter4x4\_H264High 1414  
 ReconstructChromaInter4x4MB\_H264 1414  
 ReconstructChromaInterMB\_H264 1409  
 ReconstructChromaIntra\_AVS 1491  
 ReconstructChromaIntra4x4MB\_H264 1421  
 ReconstructChromaIntraHalf4x4High\_H264 1419  
 ReconstructChromaIntraHalves4x4MB\_H264 1419  
 ReconstructChromaIntraHalvesMB\_H264 1411  
 ReconstructChromaIntraMB\_H264 1412  
 ReconstructCoeffsInter\_H261 1314  
 ReconstructCoeffsInter\_H263 1338  
 ReconstructCoeffsInter\_MPEG4 1299  
 ReconstructCoeffsIntra\_H261 1313  
 ReconstructCoeffsIntra\_H263 1337  
 ReconstructDCTBlock\_MPEG1 1220  
 ReconstructDCTBlock\_MPEG2 1223  
 ReconstructDCTBlockIntra\_MPEG1 1222  
 ReconstructDCTBlockIntra\_MPEG2 1224  
 ReconstructLumaInter\_AVS 1490  
 ReconstructLumaInter4x4\_H264High 1430  
 ReconstructLumaInter4x4MB\_H264 1430  
 ReconstructLumaInter8x8\_H264High 1435  
 ReconstructLumaInter8x8MB\_H264 1435  
 ReconstructLumaInterMB\_H264 1427  
 ReconstructLumaIntra\_AVS 1489  
 ReconstructLumaIntra16x16\_H264High 1440  
 ReconstructLumaIntra16x16MB\_H264 1439, 1440  
 ReconstructLumaIntra4x4\_H264High 1434  
 ReconstructLumaIntra4x4MB\_H264 1434

ReconstructLumaIntra8x8\_H264High 1438  
ReconstructLumaIntra8x8MB\_H264 1438  
ReconstructLumaIntraHalf4x4\_H264High 1432  
ReconstructLumaIntraHalf4x4MB\_H264 1432  
ReconstructLumaIntraHalf8x8\_H264High 1436  
ReconstructLumaIntraHalf8x8MB\_H264 1436  
ReconstructLumaIntraHalfMB\_H264 1428  
ReconstructLumaIntraMB\_H264 1429  
ReconstructPredFirstRow\_JPEG 1011  
ReconstructPredRow 1011  
RectStdDev 608  
ReduceBits 346  
region of interest 48  
Remap 728  
Resample\_H263 1330  
ResampleRow 102  
ResampleRowGetBorderWidth 102  
ResampleRowGetSize 100  
ResampleRowInit 101  
ResampleRowReplicateBorder 102  
Resize 697  
ResizeCCRotate 1191  
ResizeCenter 701  
ResizeFilter 725  
ResizeFilterGetSize 724  
ResizeFilterInit 724  
ResizeGetBufSize 712  
ResizeShift 716  
ResizeSqrPixel 705  
ResizeSqrPixelGetBufSize 714  
ResizeYUV420 722  
ResizeYUV420GetBufSize 723  
ResizeYUV422 720  
RGB color model 187  
RGB image formats 199  
RGB555ToYCbCr\_JPEG 920  
RGB565ToRGB 300  
RGB565ToYCbCr\_JPEG 920  
RGB565ToYUV420 214  
RGB565ToYUV422 209  
RGBToCbYCr422 236  
RGBToCbYCr422Gamma 236  
RGBToGray 294  
RGBToHLS 278  
RGBToHSV 283  
RGBToLUV 270  
RGBToRGB565 299  
RGBToXYZ 268  
RGBToY\_JPEG 915  
RGBToYCbCr 220  
RGBToYCbCr\_JPEG 918  
RGBToYCbCr411\_JPEG 928  
RGBToYCbCr411LS\_MCU 938  
RGBToYCbCr420 246, 250  
RGBToYCbCr422 229  
RGBToYCbCr422\_JPEG 925  
RGBToYCbCr422LS\_MCU 936  
RGBToYCbCr444LS\_MCU 935  
RGBToYCC 276  
RGBToYCoCg 285  
RGBToYCrCb422 231

RGBToYUV 203  
RGBToYUV420 210  
RGBToYUV422 206  
RLE coding functions 1061, 1062, 1063, 1064  
JoinRow\_TIFF 1064  
PackBitsRow\_TIFF 1062  
SplitRow\_TIFF 1064  
UnpackBitsRow\_TIFF 1063  
ROI 48  
ROI processing, in geometric transforms 695  
Rotate 733  
RotateCenter 741  
rounding mode 37  
RShiftC 165

## S

S3 Texture Compression 1065  
SAD16x16 1163  
SAD16x16Blocks4x4 1173  
SAD16x16Blocks8x8 1172  
SAD16x16xN 1164  
SAD16x16xNI 1164  
SAD16x8 1165  
SAD4x4 1170  
SAD4x4xN 1171  
SAD4x4xNI 1171  
SAD4x8 1169  
SAD8x16 1166  
SAD8x4 1169  
SAD8x8 1167  
SAD8x8xN 1168  
SAD8x8xNI 1168  
SampleDown411LS\_MCU 978  
SampleDown422LS\_MCU 978  
SampleDown444LS\_MCU 977  
SampleDownH2V1\_JPEG 969  
SampleDownH2V2\_JPEG 970  
SampleDownRowH2V1\_Box\_JPEG 970  
SampleDownRowH2V2\_Box\_JPEG 973  
SampleLine 96  
SampleUp411LS\_MCU 980  
SampleUp444LS\_MCU 979  
SampleUpH2V1\_JPEG 973  
SampleUpH2V2\_JPEG 975  
SampleUpRowH2V1\_Triangle\_JPEG 976  
SampleUpRowH2V2\_Triangle\_JPEG 976  
sampling functions, for JPEG codec 968  
SAT8x8D 1181  
SATD16x16 1174  
SATD16x8 1175  
SATD4x4 1180  
SATD4x8 1179  
SATD8x16 1176  
SATD8x4 1178  
SATD8x8 1177  
SBGRToYCoCg 287  
SBGRToYCoCg\_Rev 291  
Scale 64  
scaling of output results 38  
ScanFwd 1188

ScanInv 1187  
scanline padding 695  
scanning  
    ScanFwd 1188  
    ScanInv 1187  
segment 1240  
SegmentGradient 881  
SegmentGradientGetBufferSize 880  
SegmentWatershed 877  
SegmentWatershedGetBufferSize 877  
Set 67  
Shear 745  
SmoothingChroma\_HorEdge\_VC1 1534  
SmoothingChroma\_VerEdge\_VC1 1532  
SmoothingLuma\_HorEdge\_VC1 1529  
SmoothingLuma\_VerEdge\_VC1 1527  
SpatialInterpolation\_H263 1334  
spectrum locus 186  
Split422LS\_MCU 981  
Sqr 143  
SqrDiff16x16 1152  
SqrDiff16x16B 1153  
SqrDistanceFull\_Norm 670  
SqrDistanceSame\_Norm 673  
SqrDistanceValid\_Norm 675  
SqrIntegral 599  
Sqrt 145  
SRHNInitAlloc\_PSF2x2 909  
SRHNInitAlloc\_PSF3x3 909  
SSD4x4 1155  
SSD8x8 1154  
statistics functions  
    computing image moments 635  
    context initializing 633  
    deallocation 634  
    HistogramEven 613  
    HistogramRange 611  
    image norms 642, 644, 646, 649, 651, 654, 656  
    integral of pixel squares 599  
    integral representation 597  
    maximum 619  
    maximum value for a pair of pixels 626  
    maximum with index 621  
    mean and stdev value 606  
    mean value 603  
    minimum 616  
    minimum and maximum 622  
    minimum and maximum with indexes 624  
    minimum value for a pair of pixels 627  
    minimum with index 618  
    pixels count 615  
    retrieving central moments 638  
    retrieving Hu moments 640  
    retrieving spatial moments 637  
    standard deviation of integral images 608  
    standard deviation of tilted integral images 609  
    sum 595  
    tilted integral of pixel squares 602  
    tilted integral representation 601  
status codes 38  
status information 55

status information function 54  
Structures in IPPi 41  
Sub 127  
Sub128\_JPEG 967  
Sub16x16 1150, 1151  
Sub8x8 1150  
SubC 129  
SubSAD8x8 1151  
Sum 595  
SumsDiff16x16Blocks4x4 1183  
SumsDiff8x8Blocks4x4 1184  
SumWindowColumn 434  
SumWindowRow 433  
superblock 1240  
SuperSampling 718  
SuperSamplingGetBufSize 720  
support functions 53  
SwapChannels 88

## T

texture compression 1065  
TextureDecodeBlock 1071  
TextureDecodeBlockFromRGB 1070  
TextureDecodeBlockFromRGBA 1069  
TextureDecodeBlockToRG 1070  
TextureEncodeBlock 1067  
TextureEncodeBlock4x4FromRGBA 1068  
TextureEncodeBlockFromRGB 1067  
TextureEncodeBlockFromRGBA 1066  
TextureEncodeBlockFromYCoCg 1072  
Threshold 364  
threshold functions 364  
Threshold\_GT 366  
Threshold\_GTVal 372  
Threshold\_LT 368  
Threshold\_LTVal 374  
Threshold\_LTValGTVal 376  
Threshold\_Val 370  
tiled image processing 51  
Tilted RectStdDev 609  
TiltedHaarClassifierInitAlloc 895  
TiltedIntegral 601  
TiltedSqrIntegral 602  
ToneMapLinear 361  
ToneMapMean 361  
Transform4x4Fwd\_VC1 1547  
Transform4x4Inv\_VC1 1511  
Transform4x8Fwd\_VC1 1547  
Transform4x8Inv\_VC1 1511  
Transform8x4Fwd\_VC1 1547  
Transform8x4Inv\_VC1 1511  
Transform8x8Fwd\_VC1 1547  
Transform8x8Inv\_VC1 1511  
TransformDequantChromaDC\_H264 1365  
TransformDequantChromaDC\_SISP\_H264 1370  
TransformDequantLumaDC\_H264 1364  
TransformLuma8x8Fwd\_H264 1464  
TransformLuma8x8InvAddPred\_H264 1480  
TransformPrediction\_H264 1369  
TransformQuant8x8Fwd\_AVS 1501

TransformQuantChromaDC\_H264 1459  
TransformQuantLumaDC\_H264 1461  
TransformQuantResidual\_H264\_16s\_C1I 1463  
TransformResidual4x4Fwd\_H264 1462  
TransformResidual4x4Inv\_H264 1365  
Transpose 86  
TrueDistanceTransform 832  
TrueDistanceTransformGetBufferSize 831

## U

UndistortGetSize 902  
UndistortRadial 903  
UnidirWeight\_H264 1403  
UniDirWeightBlock\_H264 1403  
uniform chromaticity scale diagram 187  
universal pyramids 858  
UpdateMotionHistory 846  
UpsampleFour\_H263 1332  
UpsampleFour8x8\_H263 1333

## V

Variance16x16 1159, 1162  
VarMean8x8 1156  
VarMeanDiff16x16 1157  
VarMeanDiff16x8 1158  
VC-1 decoder functions  
    bicubic interpolation 1522, 1525  
    bilinear interpolation 1520, 1523  
    deblocking filtering 1536, 1537, 1538  
    dequantization 1539, 1542  
    range reduction 1545  
    smoothing filtering 1527, 1529, 1532, 1534  
    bicubic interpolation 1522, 1525  
    bilinear interpolation 1520, 1523  
    intensity compensation 1523, 1525  
VC-1 encoder functions  
    quantization 1552, 1554  
VC1 decoder functions  
    inverse transform 1511  
VC1 encoder functions  
    forward transform 1547  
version information function 53  
version information, for JPEG library 914  
vertical sampling, in a JPEG codec 968  
video coding color conversion 1189  
video coding general functions 1088  
video encoder functions, motion estimation  
    MeanAbsDev16x16 1161  
    MeanAbsDev8x8 1160  
video processing functions  
    general color conversion 1189, 1191  
video processing functions, general  
    DeinterlaceBlend 1204  
    DeinterlaceBlendFree 1205  
    DeinterlaceBlendGetSize 1199  
    DeinterlaceBlendInit 1200  
    DeinterlaceBlendInitAlloc 1202  
    DeinterlaceEdgeDetect 1197

video processing functions, general (*continued*)

    DeinterlaceFilterCAVT 1194  
    DeinterlaceFilterTriangle 1193  
    DeinterlaceMedianThreshold 1196  
    DeinterlaceMotionAdaptive 1198  
    FilterDenoiseAdaptive 1210  
    FilterDenoiseAdaptiveFree 1209  
    FilterDenoiseAdaptiveInitAlloc 1209  
    FilterDenoiseCAST 1207  
    FilterDenoiseCASTInit 1206  
    FilterDenoiseMosquito 1213  
    FilterDenoiseMosquitoFree 1212  
    FilterDenoiseMosquitoInitAlloc 1212  
    FilterDenoiseSmooth 1208  
    Median 1195

VLCAdapt\_JPEGXR 1079  
VLCDecode4x4\_JPEGXR 1082  
VLCDecodeDC420\_JPEGXR 1082  
VLCDecodeDC422\_JPEGXR 1082  
VLCDecodeDC444\_JPEGXR 1082  
VLCEncode4x4\_JPEGXR 1081  
VLCEncode4x4Flex\_JPEGXR 1081  
VLCEncodeDC420\_JPEGXR 1081  
VLCEncodeDC422\_JPEGXR 1081  
VLCEncodeDC444\_JPEGXR 1081  
VLCGetStateSize\_JPEGXR 1078  
VLCInit\_JPEGXR 1079  
VLCScan4x4DC\_JPEGXR 1080  
VLCScanReset\_JPEGXR 1080  
VLCScanSet\_JPEGXR 1080  
VOP, video object plane 1284

## W

WarpAffine 751  
WarpAffineBack 754  
WarpAffineQuad 756  
WarpBilinear 775  
WarpBilinearBack 778  
WarpBilinearQuad 780  
WarpChroma\_MPEG4 1290  
WarpGetSize\_MPEG4 1289  
WarpInit\_MPEG4 1288  
WarpLuma\_MPEG4 1289  
WarpPerspective 763  
WarpPerspectiveBack 766  
WarpPerspectiveQuad 768  
wavelet transform functions  
    buffer size calculation 795, 801  
    deallocation 794, 800  
    forward transform 795  
    initialization 793, 799  
    inverse transform 802  
wavelet transforms, for lossless compression 1017  
WeightedAverage\_H264 1402  
WeightPrediction\_AVS 1487  
WinBartlett 576  
WinBartlettSep 576  
windowing functions  
    Bartlett window 576  
    Hamming window 577

WinHamming 577  
 WinHummingSep 577  
 WTFwd 795  
 WTFwd\_B53\_JPEG2K 1034  
 WTFwdCol\_B53\_JPEG2K 1021  
 WTFwdCol\_D97\_JPEG2K 1028  
 WTFwdColLift\_B53\_JPEG2K 1022  
 WTFwdColLift\_D97\_JPEG2K 1030  
 WTFwdFree 794  
 WTFwdGetBufSize 795  
 WTFwdInitAlloc 793  
 WTFwdRow\_B53\_JPEG2K 1018  
 WTFwdRow\_D97\_JPEG2K 1026  
 WTIInv 802  
 WTIInv\_B53\_JPEG2K 1036  
 WTIInv\_D97\_JPEG2K 1039  
 WTIInvCol\_B53\_JPEG2K 1023  
 WTIInvCol\_D97\_JPEG2K 1031  
 WTIInvColLift\_B53\_JPEG2K 1025  
 WTIInvColLift\_D97\_JPEG2K 1032  
 WTIInvFree 800  
 WTIInvGetBufSize 801  
 WTIInvInitAlloc 799  
 WTIInvRow\_B53\_JPEG2K 1020  
 WTIInvRow\_D97\_JPEG2K 1027

## X

Xor 161  
 XorC 162  
 XYZToRGB 269

## Y

YCbCr411 329  
 YCbCr411ToBGR 265  
 YCbCr411ToBGR\_JPEG 931  
 YCbCr411ToBGR555 267  
 YCbCr411ToBGR555LS\_MCU 951  
 YCbCr411ToBGR565 267  
 YCbCr411ToBGR565LS\_MCU 951  
 YCbCr411ToBGR565LS\_MCU 950  
 YCbCr411ToBGR565LS\_MCU 947  
 YCbCr411ToBGR565LS\_MCU 945  
 YCbCr411ToBGR 222  
 YCbCr411ToBGR\_709CSC 223  
 YCbCr411ToBGR\_JPEG 922  
 YCbCr411ToBGR444 226  
 YCbCr411ToBGR444Dither 228  
 YCbCr411ToBGR555 226  
 YCbCr411ToBGR555\_JPEG 924  
 YCbCr411ToBGR555Dither 228  
 YCbCr411ToBGR565 226  
 YCbCr411ToBGR565\_JPEG 924  
 YCbCr411ToBGR565Dither 228  
 YCbCr411ToRGB 221  
 YCbCr411ToRGB\_JPEG 919  
 YCbCr411ToRGB444 224  
 YCbCr411ToRGB444Dither 225  
 YCbCr411ToRGB555 224  
 YCbCr411ToRGB555\_JPEG 921  
 YCbCr411ToRGB555Dither 225  
 YCbCr411ToRGB565 224  
 YCbCr411ToRGB565\_JPEG 921  
 YCbCr411ToRGB565Dither 225  
 YCbCr411ToYCbCr420 332  
 YCbCr411ToYCbCr422 330  
 YCbCr411ToYCrCb420 334  
 YCbCr411ToYCrCb422 331  
 YCbCr420 315  
 YCbCr420To422\_Interlace 319  
 YCbCr420ToBGR 256  
 YCbCr420ToBGR\_709CSC 257  
 YCbCr420ToBGR\_709HDTV 258  
 YCbCr420ToBGR444Dither 262  
 YCbCr420ToBGR555Dither 262  
 YCbCr420ToBGR565Dither 262  
 YCbCr420ToCbYCr422 320  
 YCbCr420ToCbYCr422\_Interlace 320  
 YCbCr420ToRGB 247  
 YCbCr420ToRGB444Dither 249

YCbCr420ToRGB555Dither 249  
 YCbCr420ToRGB565Dither 249  
 YCbCr420ToYCbCr411 324  
 YCbCr420ToYCbCr422 316  
 YCbCr420ToYCbCr422\_Filter 317  
 YCbCr420ToYCrCb420 321  
 YCbCr420ToYCrCb420\_Filter 322  
 YCbCr422 301  
 YCbCr422To420\_Interlace 305  
 YCbCr422ToBGR 234  
 YCbCr422ToBGR\_JPEG 928  
 YCbCr422ToBGR444 244  
 YCbCr422ToBGR444Dither 245  
 YCbCr422ToBGR555 244  
 YCbCr422ToBGR555Dither 245  
 YCbCr422ToBGR555LS\_MCU 950  
 YCbCr422ToBGR565 244  
 YCbCr422ToBGR565Dither 245  
 YCbCr422ToBGR565LS\_MCU 950  
 YCbCr422ToBGR565LS\_MCU 949  
 YCbCr422ToCbYCr422 302  
 YCbCr422ToRGB 230  
 YCbCr422ToRGB\_JPEG 926  
 YCbCr422ToRGB444 241  
 YCbCr422ToRGB444Dither 242  
 YCbCr422ToRGB555 241  
 YCbCr422ToRGB555Dither 242  
 YCbCr422ToRGB565 241  
 YCbCr422ToRGB565Dither 242  
 YCbCr422ToRGB565LS\_MCU 946  
 YCbCr422ToYCbCr411 307  
 YCbCr422ToYCbCr420 303  
 YCbCr422ToYCrCb420 306  
 YCbCr444ToBGR555LS\_MCU 948  
 YCbCr444ToBGR565LS\_MCU 948  
 YCbCr444ToBGR565LS\_MCU 947  
 YCbCr444ToRGB565LS\_MCU 945  
 YCbCrToBGR 222  
 YCbCrToBGR\_709CSC 223  
 YCbCrToBGR\_JPEG 922  
 YCbCrToBGR444 226  
 YCbCrToBGR444Dither 228  
 YCbCrToBGR555 226  
 YCbCrToBGR555\_JPEG 924  
 YCbCrToBGR555Dither 228  
 YCbCrToBGR565 226  
 YCbCrToBGR565\_JPEG 924  
 YCbCrToBGR565Dither 228  
 YCbCrToRGB 221  
 YCbCrToRGB\_JPEG 919  
 YCbCrToRGB444 224  
 YCbCrToRGB444Dither 225  
 YCbCrToRGB555 224  
 YCbCrToRGB555\_JPEG 921  
 YCbCrToRGB555Dither 225  
 YCbCrToRGB565 224  
 YCbCrToRGB565\_JPEG 921  
 YCbCrToRGB565Dither 225  
 YCCK color model 187  
 YCCK444ToCMYKLS\_MCU 952  
 YCCKToCMYK\_JPEG 933

YCCKToCMYK411LS\_MCU 953  
YCCKToCMYK422LS\_MCU 953  
YCCToRGB 277  
YCrCb color model 187  
YCoCg color model 187  
YCoCgToBGR 288  
YCoCgToBGR\_Rev 292  
YCoCgToRGB 286  
YCoCgToSBGR 289  
YCoCgToSBGR\_Rev 293  
YCrCb411ToYCbCr422\_16x4x5MB\_DV 1273  
YCrCb411ToYCbCr422\_5MBDV 1272  
YCrCb411ToYCbCr422\_EdgeDV 1274  
YCrCb411ToYCbCr422\_ZoomOut2\_5MBDV 1272  
YCrCb411ToYCbCr422\_ZoomOut2\_EdgeDV 1274  
YCrCb411ToYCbCr422\_ZoomOut4\_5MBDV 1272  
YCrCb411ToYCbCr422\_ZoomOut4\_EdgeDV 1274  
YCrCb411ToYCbCr422\_ZoomOut8\_5MBDV 1272  
YCrCb411ToYCbCr422\_ZoomOut8\_EdgeDV 1274  
YCrCb420ToCbYCr422 327  
YCrCb420ToRGB 251  
YCrCb420ToYCbCr411 328  
YCrCb420ToYCbCr420 327  
YCrCb420ToYCbCr422 325  
YCrCb420ToYCbCr422\_5MBDV 1275  
YCrCb420ToYCbCr422\_8x8x5MB\_DV 1276  
YCrCb420ToYCbCr422\_Filter 326  
YCrCb420ToYCbCr422\_ZoomOut2\_5MBDV 1275  
YCrCb420ToYCbCr422\_ZoomOut4\_5MBDV 1275  
YCrCb420ToYCbCr422\_ZoomOut8\_5MBDV 1275  
YCrCb422ToRGB 232  
YCrCb422ToYCbCr411 310

YCrCb422ToYCbCr420 309  
YCrCb422ToYCbCr422 308  
YCrCb422ToYCbCr422\_10HalvesMB16x8\_DV100 1279  
YCrCb422ToYCbCr422\_5MBDV 1277  
YCrCb422ToYCbCr422\_8x4x5MB\_DV 1279  
YCrCb422ToYCbCr422\_ZoomOut2\_5MBDV 1277  
YCrCb422ToYCbCr422\_ZoomOut4\_5MBDV 1277  
YCrCb422ToYCbCr422\_ZoomOut8\_5MBDV 1277  
YUV color model 187  
YUV420ToBGR 213  
YUV420ToBGR444 218  
YUV420ToBGR444Dither 219  
YUV420ToBGR555 218  
YUV420ToBGR555Dither 219  
YUV420ToBGR565 218  
YUV420ToBGR565Dither 219  
YUV420ToRGB 211  
YUV420ToRGB444 215  
YUV420ToRGB444Dither 216  
YUV420ToRGB555 215  
YUV420ToRGB555Dither 216  
YUV420ToRGB565 215  
YUV420ToRGB565Dither 216  
YUV422ToRGB 207  
YUVToRGB 205

## Z

zigzag order of elements 955  
ZigzagFwd8x8 97  
ZigzagInv8x8 99