Unreal Engine* 4.19 の最適化にインテルのソフトウェア・エンジニアが協力

Garret Romaine、Jeff Rous (Intel)

この記事は、2018 年 6 月 8 日時点の、インテル® デベロッパー・ゾーンに公開されている「Intel Software Engineers Assist with Unreal Engine* 4.19 Optimizations」の日本語訳です。

Epic* の Unreal Engine* 4.19 のリリースは、インテル® テクノロジー向けの最適化、特にマルチコア・プロセッサーの最適化の歴史に新しいページを刻みました。これまで、ゲームエンジンは、グラフィックス機能およびパフォーマンスの点から、伝統的にコンソールの設計よりも後回しにされていました。一般に、ほとんどのゲームは最新のプロセッサー向けに最適化されず、PC のパフォーマンスの多くはアイドル状態のまま活用されていませんでした。インテルは、開発者が Unreal Engine* 4 でワークを実行する際に、PC プラットフォームが提供するプロセッサーの計算能力をすべて利用し、ただちにゲームの性能を引き出せるように取り組みました。

Unreal Engine* バージョン 4.19 では、インテルの協力により、次の拡張が行われました。

- ユーザーが利用しているプロセッサーと一致するようにワーカースレッドの数を増加。
- クロス (布) フィジックス・システムのスループットを向上。
- インテル® VTune™ Amplifier の統合をサポート。

Unreal Engine* ユーザーは、これらの拡張により、インテル® アーキテクチャーを活用して、マルチコアシステムの能力を引き出すことができます。クロス・フィジックス、動的破壊、プロセッサー・パーティクルなどのシステム、およびインテル® VTune™ Amplifier やインテル® C++ コンパイラーなどのインテル® ソフトウェア開発ツールとの操作性の強化はすべて利点です。この記事では、重要な向上点について詳細に説明した後、開発者が次の PC タイトルで Unreal Engine* を考慮すべき理由を述べます。

Unreal Engine* の歴史

1991 年に、Tim Sweeney 氏はメリーランド大学在籍中に Epic MegaGames (現在の Epic Games*) を設立しました。同じ年に、最初の製品としてシェアウェアのパズルゲーム *ZZT* (英語) を発売しました。ゲームはオブジェクト指向モデルを使用して Turbo Pascal で記述されて

おり、ユーザーはゲームのコードを実際に変更することができました。レベルエディターはすでに一般的でしたが、これは大きな進歩でした。

その後、Epic* は *Epic* Pinball*、*Jill of the Jungle*、*Jazz Jackrabbit** などのゲームを発売しました。1995 年に、Sweeney 氏は *DOOM**、*Wolfenstein**、*Quake**、*Duke Nukem** などのゲームに代表されるファーストパーソン・シューター (本人視点シューティング・ゲーム) の研究を始めました。1998 年に、Epic* は *Unreal** を発売しました。このゲームは、発売時点で最高のファーストパーソン・シューターの 1 つであり、詳細なグラフィックスは業界の注目を集めました。その後まもなく、ほかの開発者から Unreal Engine* を自分たちのゲームで利用したいという問い合わせが寄せられるようになりました。

2010 年の IGN の記事 (英語) で、Sweeney 氏は、チームは多くの問い合わせに興奮したこと、これらのパートナーとの初期のコラボレーションがその後のエンジンビジネスのスタイルを定義したことを述べています。現在も使用されているそのスタイルとは、「コミュニティー主導のアプローチ、およびライセンシーとエンジンチーム間のオープンかつ直接的なコミュニケーション」です。目標は常に、統一されたツールを作成して技術的障害を取り除くことに注目し、ゲーム・コミュニティーの創造性を解き放つことでした。初期のエンジンでは提供されないことが多い、大量のドキュメントとサポートも提供しました。

今日では、Unreal Engine* (英語) は、ゲーム業界で上位の収益を上げている多くのタイトルで利用されています。2017 年 3 月の VentureBeat のインタビュー (英語) で、Sweeney 氏は、開発者は Unreal Engine* を利用することにより 100 億ドルを超える収益を上げていると述べています。「Unreal Engine* の市場シェアは、収益で見ると競合他社の 2 倍あります。ユーザー数では Unity* のほうが多いにもかかわらず収益の市場シェアが高いという事実は、Unreal* がハイエンドに集中しているおかげです。収益で Steam の上位 100 位を占めるゲームの多くは Unreal* であり、ライセンス可能なエンジンを提供している競合他社の合計を上回っています。」

インテルとのコラボレーションによる Unreal Engine* の機能の向上

Unreal Engine* を現在ライセンスしているゲーム開発者は、ここに記述されている最適化を活用できます。最適化により、統合グラフィックスを備えたラップトップやタブレットからディスクリート・グラフィックス・カードを備えたハイエンド・デスクトップまで、利用可能なプラットフォームの範囲が拡大され、ゲームの市場シェアの拡大に貢献するでしょう。最適化は、プラットフォームでダイナミック・クロスやインタラクティブ・フィジックスなどのハイエンドな効果の実現を保証することにより、多くの PC ベースのシステムのエンドユーザーに役立ちます。さらに、最適化されたインテルのツールを利用することで、インテル® アーキテクチャーの能力を最大限に引き出すことができます。

インテル・デベロッパー・リレーションズのエンジニア Jeff Rous によれば、インテルと Epic Games* のチームは 1990 年代後半から協力して作業を行っています。Rous は、Email および電話会議により Epic* のエンジニアとの広範な共同作業や活発なコミュニケーションを行うだけでなく、密度の高い作業を行うために年に数回ノースカロライナにある Epic* 本社をたびたび訪問し、約 6 年間 Unreal Engine* の最適化に個人的に取り組んできました。Rous は、

Unreal Engine* コードの最適化に加えて、Epic* の *Fortnite**(英語) *Battle Royale* などのタイトルにも取り組みました。

現在の作業に先立って、インテルは以前の Unreal Engine* 4 のリリースでも Unreal* と協力して作業を行っていました。インテル® デベロッパー・ゾーンには、「Unreal Engine* 4 最適化チュートリアル、パート 1」(英語) から始まる一連の最適化チュートリアルが掲載されています。チュートリアルには、開発者がエンジンの内部および外部で使用できるツール、エディターのベスト・プラクティス、プロジェクトのフレームレートと安定性の向上に役立つスクリプトの説明が含まれます。

インテル® C++ コンパイラーの拡張

インテルは、Unreal Engine* 4.12 のパブリック・エンジン・リリースに、インテル® C++ コンパイラーのサポートを追加しました。インテル® C++ コンパイラーは、アプリケーションのパフォーマンス向上に役立つ標準規格に基づく C/C++ ツールです。主要なコンパイラー、開発環境、オペレーティング・システムとのシームレスな互換性を提供し、優れた最適化と SIMD ベクトル化、インテル® パフォーマンス・ライブラリーとの統合、最新の OpenMP* 5.0 並列プログラミング・モデルの活用により、アプリケーションのパフォーマンスを向上できます。

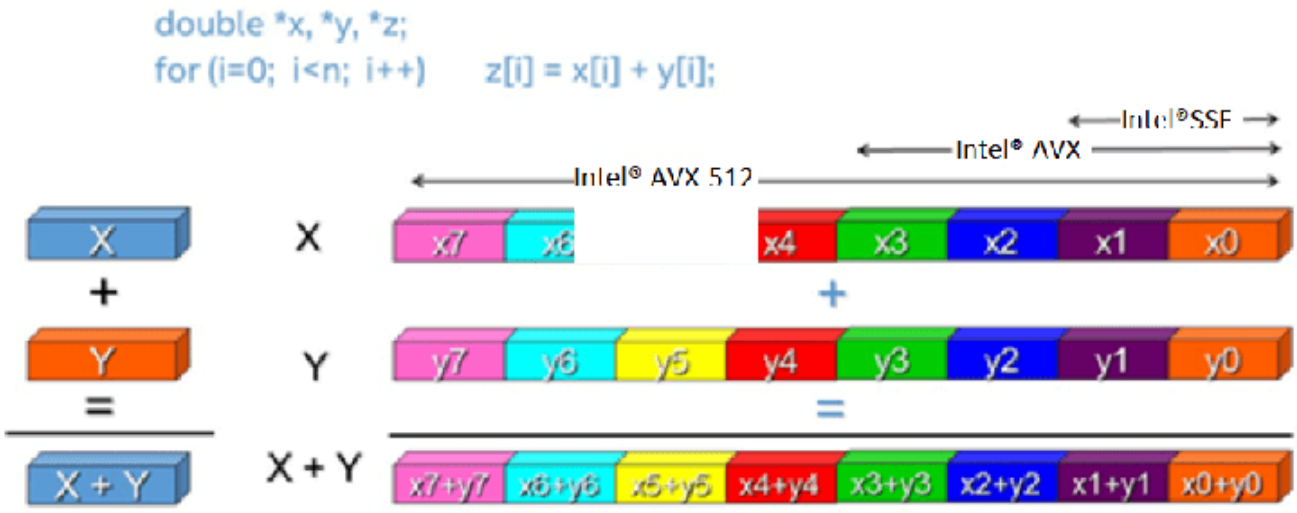

図 1: ループのスカラーバージョンとインテル® ストリーミング SIMD 拡張命令 (インテル® SSE)、インテル® アドバンスト・ベクトル・エクステンション (インテル® AVX)、インテル® AVX-512 を利用してベクトル化されたバージョン。

Unreal Engine* 4.12 以降、インテルはコードベースを最新に保ち、Infiltrator ワークロードのテストのフレームレートを大幅に高めました。

テクスチャー圧縮の向上

Unreal Engine* 4 では、インテルの高速テクスチャー圧縮のサポートも追加されました。ISPC はインテル® SPMD (Single Program Multiple Data) プログラム・コンパイラーを表します。開発者は、コード・ライブラリーを使用して、マルチコアおよび新しい/将来の命令セットをターゲットにできます。ISPC テクスチャー・ライブラリーを統合する前は、最新かつ最も先進的なテクス

チャー圧縮形式である ASTC (Adaptive Scalable Texture Compression) でも、圧縮には長い時間がかかっていました。*Sun Temple* のデモ (Unreal Engine* 4 サンプル・シーン・パックの一部) では、以前使用していたリファレンス・エンコーダーよりも品質が向上しているにもかかわらず、すべてのテクスチャーの圧縮にかかる時間を 68 分から 35 秒に短縮できました。コンテンツ開発者はプロジェクトをすばやく構築して、作業時間を大幅に節約できます。

Unreal Engine* 4.19 の最適化

Unreal Engine* 4.19 におけるインテルの取り組みは、開発者に多くの利点を提供します。エンジンレベルでは、最適化によりスケーリング・メカニズムとタスク処理が向上します。エンジンレベルのほかの作業では、レンダリング・プロセスが CPU 使用率によるボトルネックでないことが保証されます。

また、ゲーム開発者が使用する多くのミドルウェア・システムも最適化の恩恵を受けます。フィジックス、人工知能、ライティング、オクルージョン・カリング、バーチャル・リアリティー (VR) アルゴリズム、ベジテーション (植物化)、オーディオ、非同期計算はすべて恩恵を受けます。

4.19 のタスクシステムに対する変更の恩恵を理解するには、Unreal Engine* のスレッド化モデルの概要を知っておくと良いでしょう。

Unreal Engine* 4 のスレッド化モデル

図 2 は、時間 (左から右へ進む) を表しています。ゲームスレッドはほかのスレッドよりも先に実行され、レンダースレッドはゲームスレッドの 1 フレーム後になります。つまり、表示はすべて 2 フレーム後に実行されます。

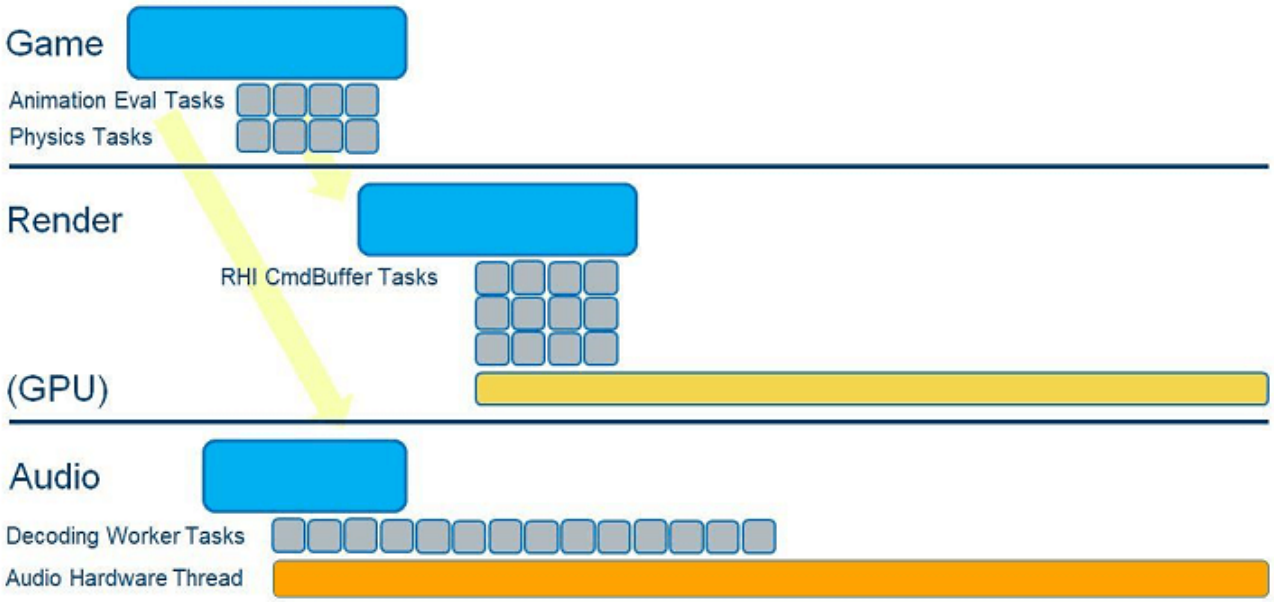

図 2: Unreal Engine* 4 のスレッド化モデル。

フィジックス・ワークはゲームスレッド上で生成され、並列に実行されます。アニメーションも並列に評価されます。アニメーションの並列評価は、最近の VR タイトル、*Robo Recall* で効果的に使用されました

図 3 で示されているゲームスレッドは、ゲームプレイ、アニメーション、フィジックス、ネットワーク、そして最も重要なアクターのティックアクションの更新を制御します。

開発者は、ティックグループを使用して、オブジェクト・ティックの順序を制御できます。ティックグループは並列処理を提供しませんが、並列のワークのスケジュールに依存する動作を開発者が制御できるようにします。並列のワークが後でゲームスレッドのボトルネックの原因とならないことを保証するには、この制御が不可欠です。

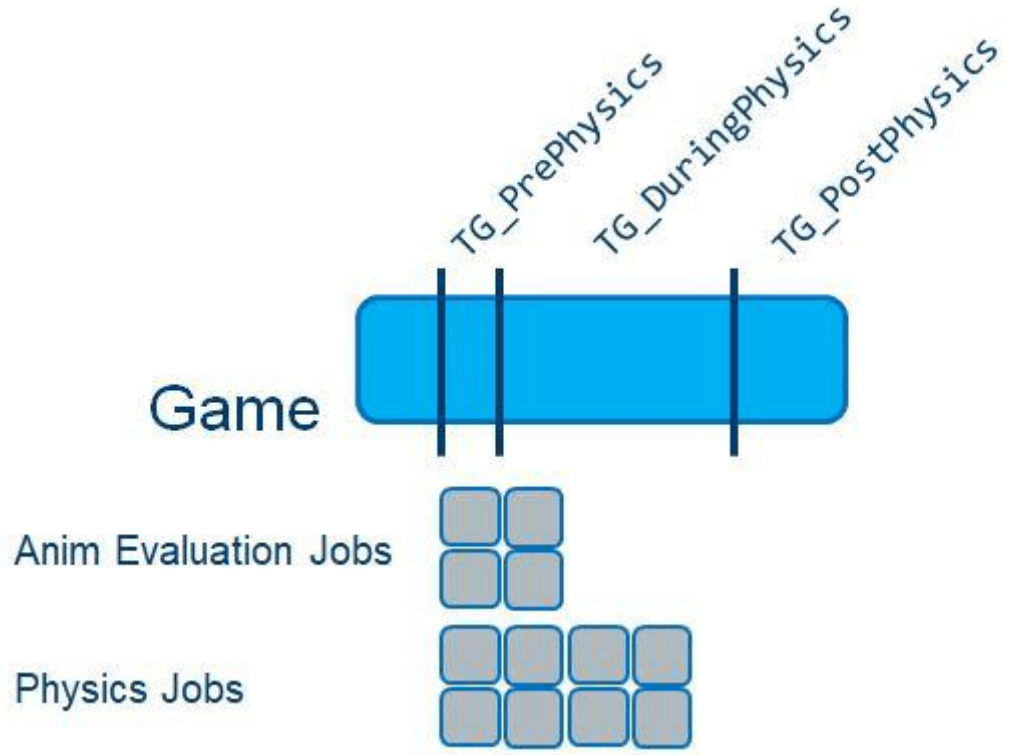

図 3: ゲームスレッドと関連するジョブ。

図 4 で示されているように、レンダースレッドは GPU に送るレンダーコマンドの生成を制御します。基本的に、シーンがスキャンされた後、GPU に送るコマンドバッファーが生成されます。コマンドバッファーの生成は、シーン全体のコマンドの生成にかかる時間を減らし、ワークをより早く GPU に送るため、並列に行われます。

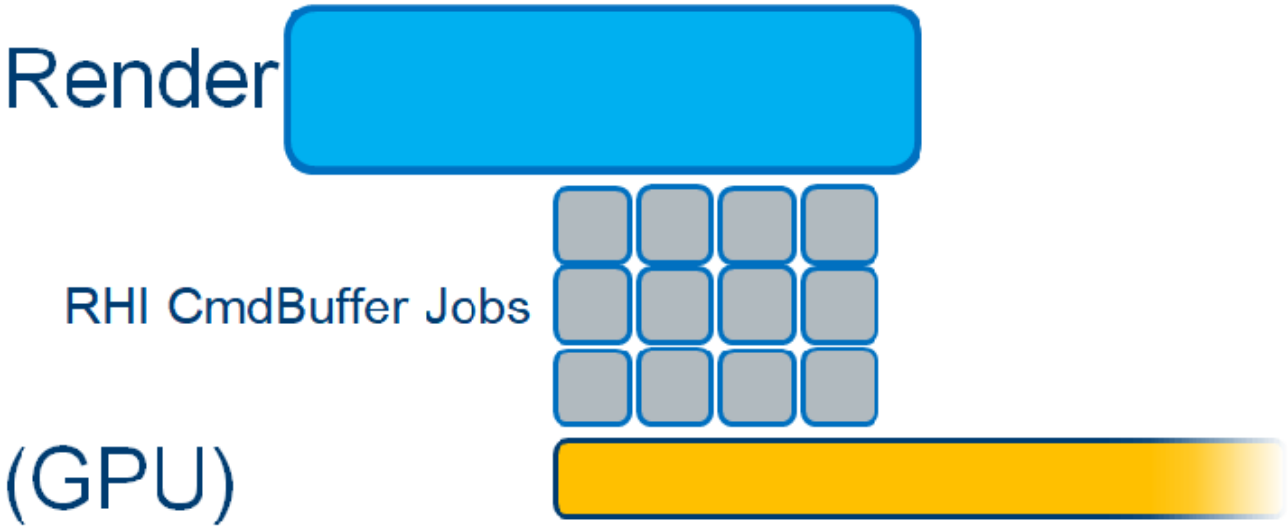

図 4: レンダー・スレッド・モデルは描画呼び出しをチャンクに分割することに依存します。

各フレームは、次々に実行されるフェーズに分割されます。各フェーズ内で、レンダースレッドはそのフェーズのコマンドリストを生成します。

- 深度プリパス
- ベースパス
- 半透明性
- 速度

フレームをチャンクに分割すると、タスクの結果が得られる並列コマンドリストを使用して、ワーカータスクに移行されます。それらは再度シリアル化された後、描画呼び出しの生成に使用されます。エンジンは呼び出し位置でワーカースレッドをジョインせず、同期ポイント (フェーズの最後) でジョインするか、使用されるポイントでジョインします (すばやく実行できる場合)。

オーディオスレッド

メイン・オーディオ・スレッドはレンダースレッドと似ていて、次のタスクを実行することで低レベルのミキシング関数のインターフェイスの役割を果たします。

- サウンド・キュー・グラフの評価
- ウェーブ・インスタンスの作成
- 減衰などの制御

オーディオスレッドは、ユーザーが指定した API (Blueprint や Gameplay などの) がすべて対話するスレッドです。デコードおよびソースワーカーのタスクは、オーディオ情報をデコードし、空間配置や頭部伝達関数 (HRTF) のアンパックなどの処理も実行します。(アルゴリズムでユーザーが音の位置や距離で違いを把握できるようにするには、HRTF は VR のプレイヤーにとって非常に重要です。)

オーディオ・ハードウェア・スレッドは、出力ハードウェアに直接レンダリングしてミックスを消費する単一プラットフォーム依存のスレッドです (例えば、Microsoft* Windows* の XAudio2)。このスレッドは Unreal Engine* により作成および制御されませんが、最適化はスレッド使用の影響を受けます。

2 種類のタスク (デコードおよびソースワーカー) があります。

- デコード: 圧縮されたソースファイルのブロックをデコードします。ダブル・バッファリングを使用して、再生中に圧縮したオーディオをデコードします。
- ソースワーカー: サンプルレート変換、空間配置 (HRTF)、効果を含むソースに実際のソース処理を行います。ソースワーカーの数は INI ファイルで設定可能です。
  - 4 つのワーカーと 32 のソースがある場合、各ワーカーは 8 つのソースをミックスします。
  - ソースワーカーは高度に並列化できるため、プロセッサーの処理能力が高い場合は数を増やすことができます。

Robo Recall は、Unreal Engine* の新しいオーディオミキシングとスレッド化システムを利用する最初のタイトルでもあります。*Robo Recall* では、HRTF はオーディオ時間のほぼ半分かかります。

プロセッサー・ワーカー・スレッドのスケーリング

Unreal Engine* 4.19 以前は、タスクグラフに利用可能なワーカースレッドの数は制限されていて、インテル® ハイパースレッディング・テクノロジーを考慮していませんでした。その結果、6 コア以上のシステムではアイドル状態になっていました。タスクグラフ (Unreal Engine* の内部ワーク・スケジューラー) で利用可能な正しい数のワーカースレッドを正確に作成することにより、コンテンツ・クリエーターはアニメーション、クロス、破壊、パーティクルなどの視覚拡張システムをスケーリングできます。

Unreal Engine* 4.19 では、タスクグラフのワーカースレッドの数はユーザーのプロセッサーに基づいて計算されます (優先度レベルあたり 22 まで)。

```
if (NumberOfCoresIncludingHyperthreads > NumberOfCores)
{
   NumberOfThreads = NumberOfCoresIncludingHyperthreads - 2;
}
else
{
   NumberOfThreads = NumberOfCores - 1;
}
```

並列ワークの最初のステップは、ゲームが利用可能なコアをすべて利用できる可能性を残すことです。これはスケーリングを成功させる基本的な問題です。4.19 の変更により、コンテンツは、クロス・フィジックス、環境破壊、プロセッサー・ベースのパーティクル、高度な 3D オーディオなどのシステムを利用して、最新のプロセッサーの能力を最大限に引き出せるようになりました。

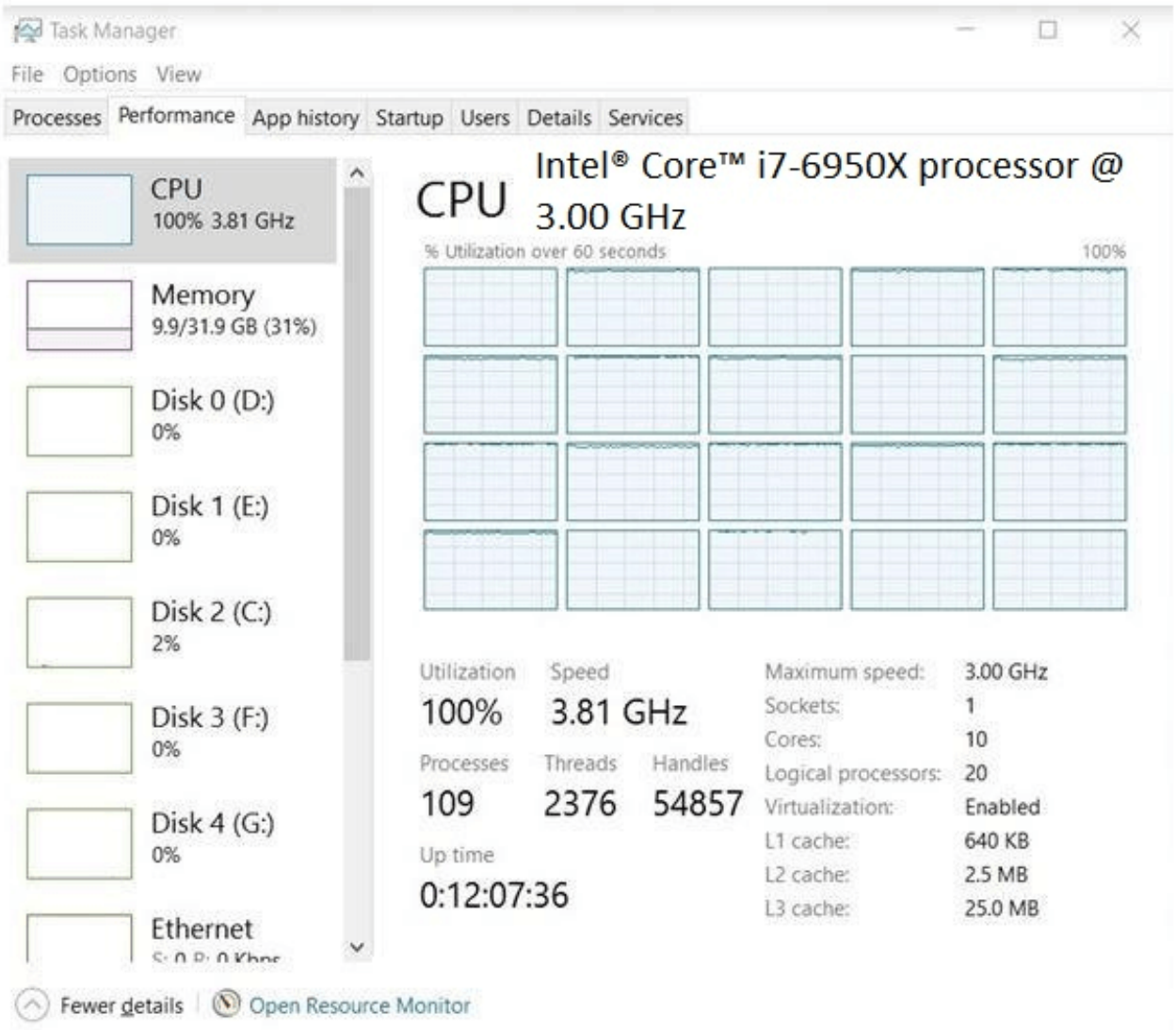

図 5: Unreal Engine* 4.19 では利用可能なハードウェア・スレッドをすべて利用する機会が提供されました。

上記のベンチマーク例では、同期ワークロードを使用したテストで、インテル® Core™ i7-6950X プロセッサー (3.00GHz) を搭載したシステムの CPU 使用率が 100% になっています。

破壊の利点

マルチコアシステムでの優れたスレッド利用による利点の 1 つは破壊です。破壊システムは、タスクグラフを使用してメッシュを小さな部分にする動的な破壊をシミュレートします。一般的な破壊ワークロードは、数秒の広範なシミュレーションから成り、続いてベースラインに戻ります。

コア数の多いプロセッサーは、より長く、より破壊された部分を保持できるため、リアル感が大幅に向上します。

Rous は、開発者が破壊を利用してさまざまなことを実行できると考えており、適切なコンテンツによるリアル感の向上の良い目標であると述べています。「より強力なプロセッサーで、長い時間の後にメッシュをさらに破壊して破壊されたチャンクをすると、破壊も簡単にスケールアップできます。破壊はワーカースレッドのフィジックス・エンジンで行われるため、非常に多くのシステムを同時に処理しない限り、プロセッサーはレンダリングのボトルネックになりません。」

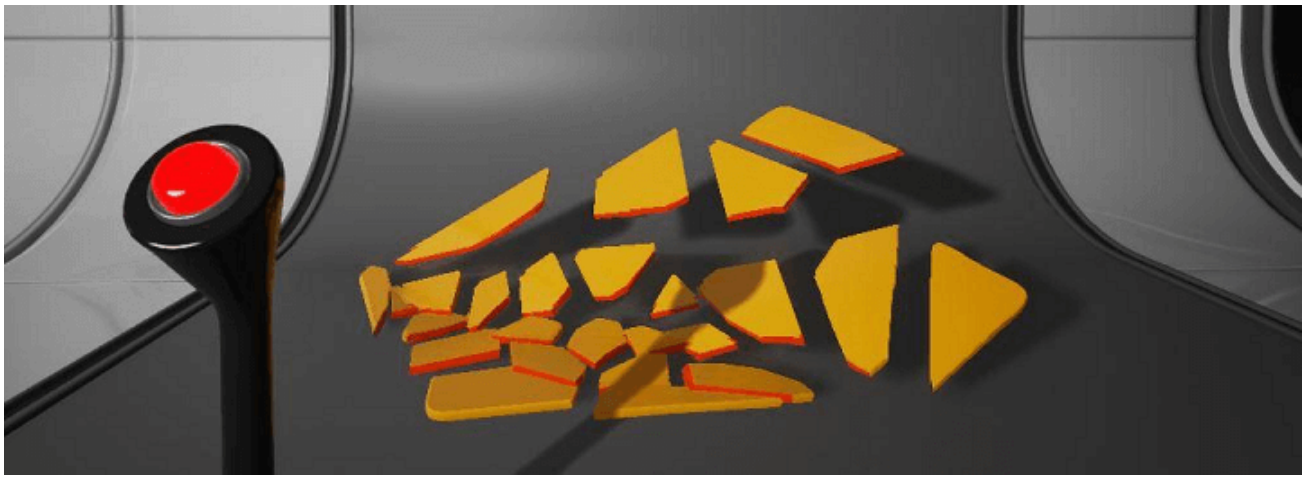

図 6: 破壊システムはメッシュを小さな部分にする動的な破壊をシミュレートします。

クロスシステムの最適化

クロスシステムは、プレイヤー、風、その他の環境要因に反応する動的 3D メッシュ・シミュレーションによりキャラクターやゲーム環境にリアル感を追加するために使用されます。ゲーム内の一般的なクロスシステムには、プレイヤーの衣類や旗が含まれます。

クロスシステムが現実的になるほど、ゲーム体験の臨場感も増します。一般的に言えば、クロスシステムが現実的になるほど、シーンはより現実的になります。

開発者は、クロスシステムを現実的にするという問題に長い間取り組んできました。この取り組みがなければ、キャラクターの服は張り付いたままで、服が風になびくような効果は実現していなかったでしょう。しかし、クロスシステムのモデル化は困難な課題でした。

クロスシステムの初期の取り組み

テキサス A&M 大学の Donald House 氏によれば、クロス・シミュレーションの最初の重要なコンピューター・グラフィックス・モデルは、1986 年に Jerry Weil 氏により示されました。House 氏ほかは、「Cloth and Clothing in Computer Graphics」でその全過程 (英語) を示し、Weil 氏の研究を詳細に記述しました。Weil 氏 (英語) は「拘束点で一時停止した編物のドレープ (ひだ) を模倣する純粋な幾何学的手法」を開発した、と House 氏は記述しています。Weil 氏のシミュレーション・プロセスには 2 つのフェーズがありました。最初に、拘束点が三角形になる懸垂線のカーブを含む布の表面を幾何学的に近似します。次に、緩和反復プロセスを適用し、

オリジナルの懸垂線の交点を補間して表面を滑らかにします。この静的ドレープモデルは、近似および緩和プロセスを完全に適用した後、連続的に拘束点をわずかに移動して緩和フェーズを再適用することにより、動的な動作を表すこともできます。

同時期に、布の動作のモデル化に物理学に基づいたアプローチを使用した連続体モデルが登場しました。これらの初期のモデルは、弾性シートとして布をモデル化した、連続体表現を使用していました。この分野の最初の論文は、1987 年に提出された、Carl R. Feynman 氏の修士論文 (英語) です。Feynman 氏は、グリッド表現に連続体弾性モデルを付け加えました。シミュレーションを行うメッシュサイズの問題により、連続体手法を使用したクロスのモデル化では、本物の布の複雑な折り畳みや座屈の動作を再現することは困難です。

パーティクル・モデルによるトラクションの獲得

パーティクル・モデルは、1992 年に David Breen および Donald House 両氏がクロスドレープの非連続体相互作用パーティクル・モデル (House 氏によれば「相互作用パーティクル・システムによりクロスのマイクロメカニカル構造の明示的な表現」) を開発 (英語) したときに適合性を獲得しました。彼は、このモデルは「クロスは物質ではなく相互に作用する機械的部品の構造と表すのが最も適切 (英語) であり、スレッド間のマイクロメカニカルな相互作用からマクロスケールの動力学的特性を引き出している。」という観察に基づくと説明しました。1994 年には、このモデルをどのように使用すれば特定のマテリアルのドレープを正確に再現できるかが示され、Breen/House モデルが拡張されました。これらのモデルの中で最も成功したモデルの 1 つは、1996 年に Eberhard、Weber、Strasser の各氏により提唱されたモデルです。彼らは、Breen/House モデルで示されている基本的なエネルギー方程式をラグランジュ力学により再定式化し、力学を計算できるようにした連立常微分方程式を使用しました。

動的メッシュ・シミュレーション・システムは、現在の一般的なモデルです。プレイヤー、風、その他の環境要因に反応して、プレイヤーのケープや旗などを現実的に表現します。

Unreal Engine* は、クロスシステムを強化するため複数のアップグレードを行ってきました。例えば、バージョン 4.16 では、APEX Cloth を NVIDIA* の NvCloth ソルバーに置換しました。この低レベルのクロスソルバーは、クロス処理を実行するパーティクル・シミュレーションを行い、軽量で、拡張性があります。開発者はデータに直接アクセスできます。

より多くの三角形、より優れたリアル感

Unreal Engine* 4.19 では、クロスシステムをさらに最適化して処理能力を向上するため、インテルのエンジニアは Unreal Engine* チームと協力して作業を行いました。クロス・シミュレーションは、ほかのフィジックス・オブジェクトのように扱われ、タスクグラフのワーカースレッド上で実行されるように変更され、開発者はマルチコア・プロセッサー上でコンテンツをスケーリングしてボトルネックを回避できるようになりました。この変更により、シーンで利用可能なクロス・シミュレーションの量は約 30% 増加しました。

プレイヤーが特定のポイントを見ていない場合でも、クロスはすべてのフレームでシミュレートされます。シミュレーションの結果は、クロスシステムがプレイヤーの視界に現れるかどうかで決定されます。クロス・シミュレーションは、システムが追加されないと仮定して、フレームごとに

ほぼ同じ量のプロセッサーを使用します。この量は簡単に予測可能であり、開発者は利用可能なヘッドルームに収まるように使用する量をチューニングできます。

図 7: Content Examples プロジェクトのクロスシステムの例。

この記事のグラフで使用されているクロスは、メッシュごとに 8,192 のシミュレートされた三角形を持ち、データがキャプチャーされたとき視界内にすべてありました。すべてのデータはインテル® Core™ i7-7820HK プロセッサー上でキャプチャーされました。

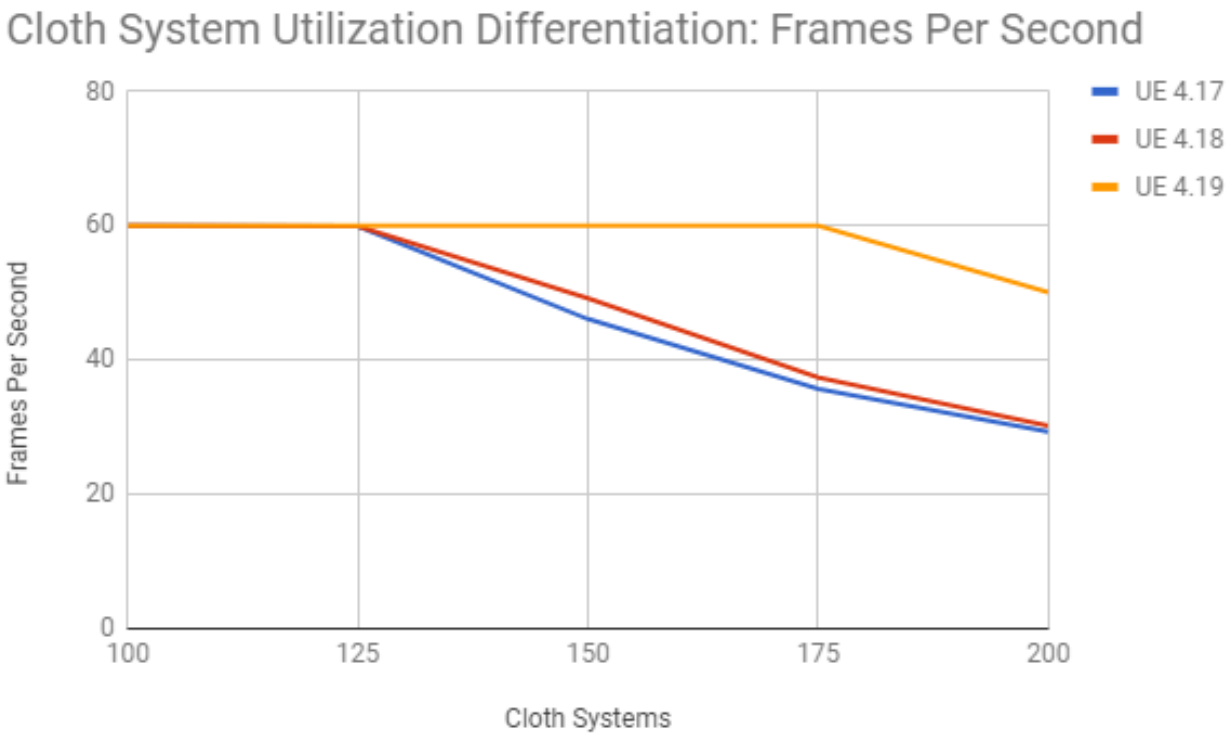

図 8: シーンのクロスシステム数に基づく Unreal Engine* 4 のバージョン間の CPU 使用率の違い。

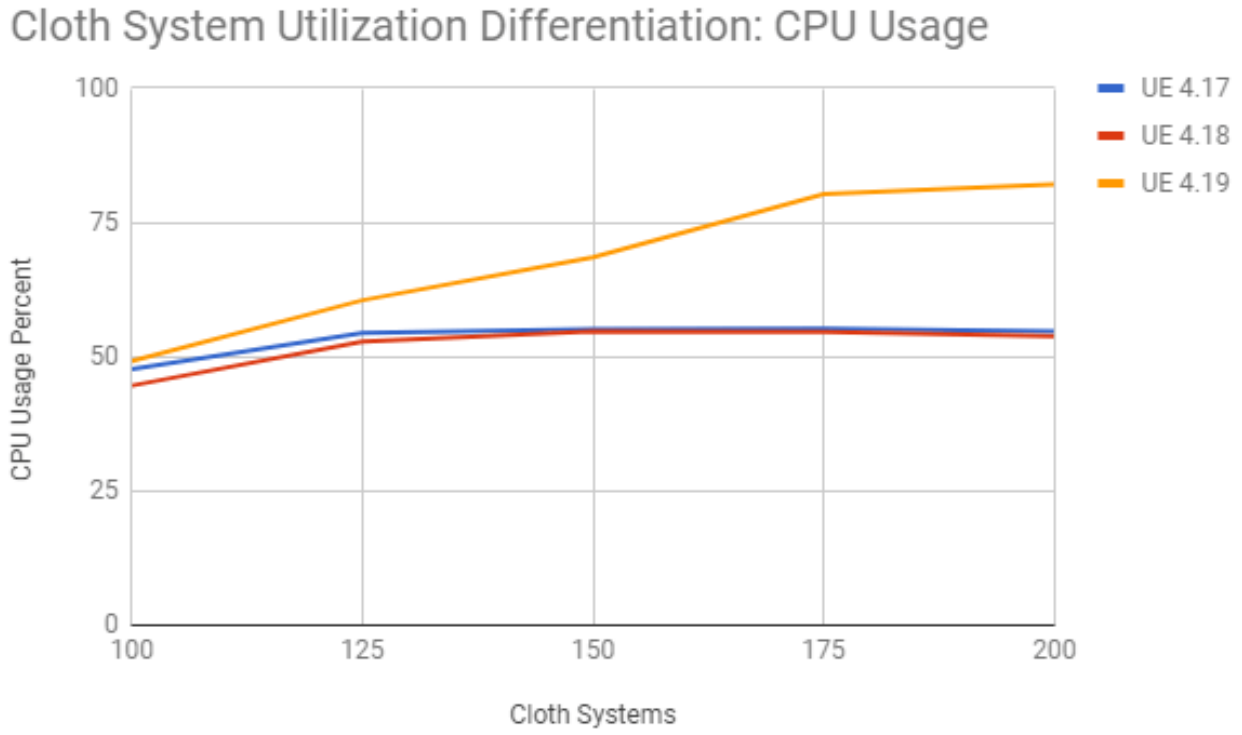

図 9: シーンのクロスシステム数に基づく Unreal Engine* 4 のバージョン間の fps の違い。

プロセッサー・パーティクルの拡張

パーティクル・システムは初期の頃から、コンピューター・グラフィックスとビデオゲームで使用されています。モーションは実生活の中心となる要素であるため、プレイヤーが完璧な没入感を得るには、パーティクルをモデル化して爆発、火の玉、雲系、その他のイベントを作成することが重要です。

プロセッサー・パーティクルには、次のような高品質の機能を利用可能です。

- 光の放射
- マテリアル・パラメーターの制御
- アトラクター・モジュール

マルチコアシステムのパーティクルは、プロセッサー・システムを GPU システムと平行して使用することにより拡張できます。このようなシステムでは簡単にスケーリングが可能です。ヘッドルームを使い切るまで、開発者はプロセッサー・ワークロードを追加し続けることができます。エンジニアは、プロセッサー・パーティクルと GPU パーティクルをペアにすると、光の放射が追加され、光が衝突するオブジェクトで反射されて、リアル感が向上することを見つけました。各システムには固有の制限があるため、ペアにすると合計よりも大きなシステムになります。

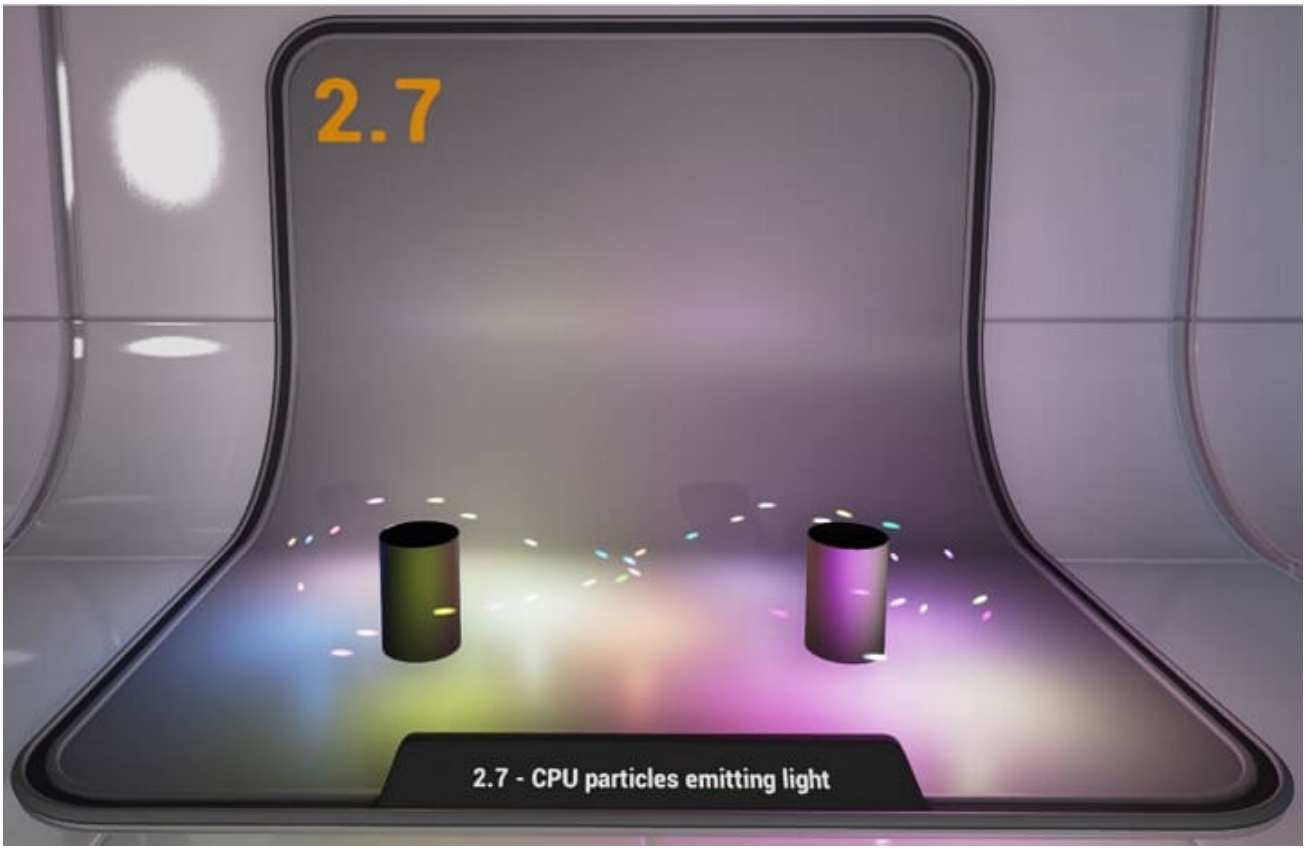

図 10: プロセッサー・パーティクルは利用可能なヘッドルームに基づいて簡単にスケーリングできます。

インテル® VTune™ Amplifier のサポート

インテル® VTune™ Amplifier は、スレッドのボトルネック、同期ポイント、プロセッサーの hotspot を特定できる業界標準のツールです。Unreal Engine* 4.19 では、インテル® VTune™ Amplifier ITT マーカーのサポートが追加されました。これにより、ユーザーは、常にエンジンが行っていることに対して詳細な情報を提供する、アノテーション付きのプロセッサー・トレースを生成できます。

ITT API には次の特徴があります。

- 収集するトレース量に基づいてアプリケーション・パフォーマンスのオーバーヘッドを制御します。
- アプリケーションを再コンパイルすることなくトレース収集が可能です。
- C/C++ および Fortran 環境でアプリケーションをサポートします。
- アプリケーション・コードをトレースするインストルメンテーションをサポートします。

ユーザーは、**-VTune** オプションを指定してインテル® VTune™ Amplifier を起動し、UI から Unreal Engine* のワークロードを実行することにより、この新しい機能を活用できます。ワーク

ロードの内部が表示されたら、コンソールで **Stat Namedevents** と入力して ITT マーカーをトレースに出力します。

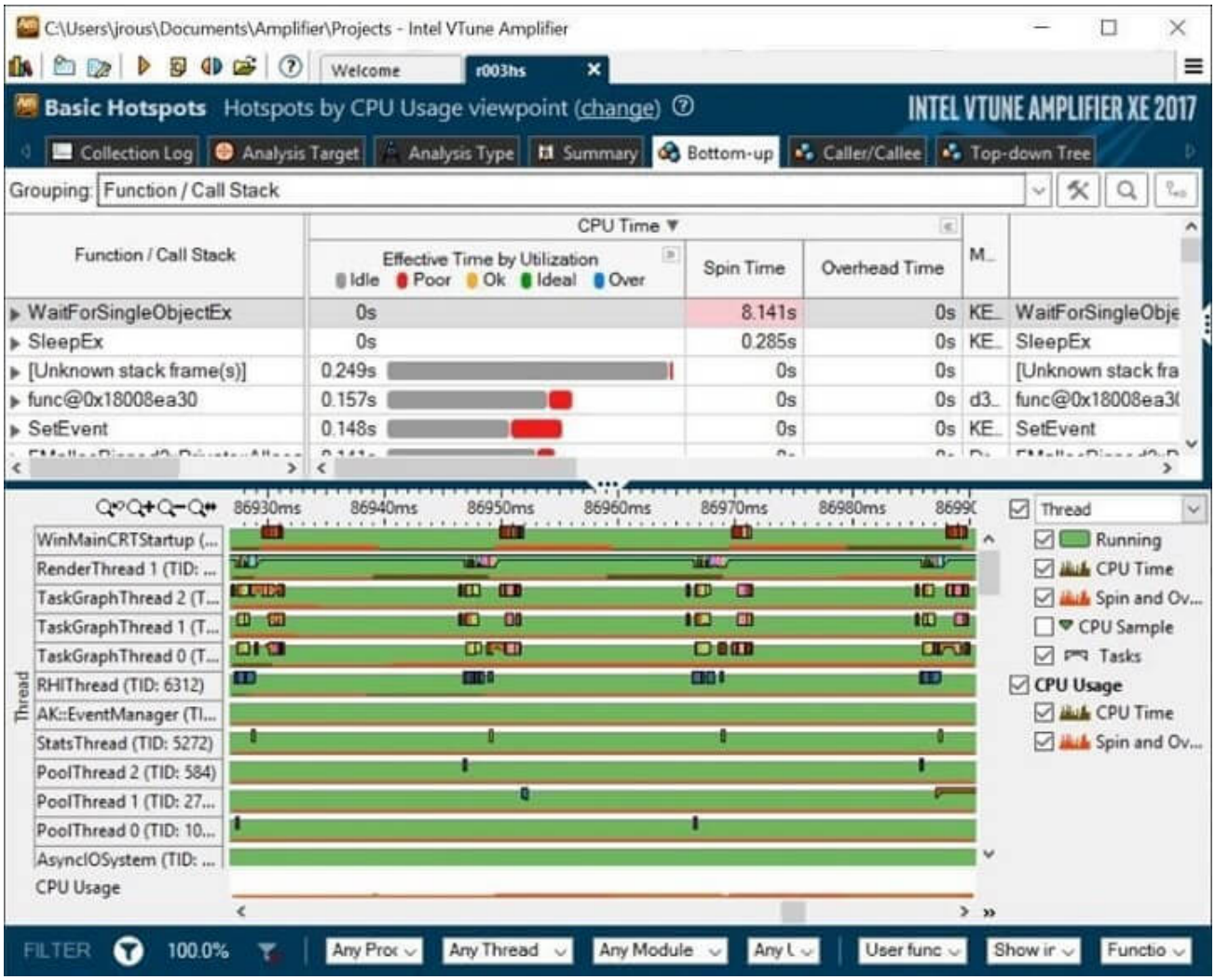

図 11: Unreal Engine* 4.19 でアノテーション付きのインテル® VTune™ Amplifier トレースを表示した例。

まとめ

改良には、すべての層 (エンジン、ミドルウェア、ゲームエディター、ゲームそのもの) の技術的な課題を解決することが含まれます。タイトルベースで作業を行うのではなく、エンジンを改良することは、Unreal* デベロッパー・エコシステム全体に役立ちます。4.19 での改良は、次の分野でエコシステム全体のプロセッサー・ワークロードの課題を解決するのに役立ちます。

- オブジェクトあたりのブレークポイントの増加による、破壊のリアル感の向上。
- パーティクルの増加による、ベジテーション、クロス、埃パーティクルなどのアニメーションの向上。
- 背景キャラクターのリアル感の向上。

- クロスシステムの向上。
- パーティクルの向上 (例えば、キャラクターや NPC との対話、環境の表現)。

多くのエンドユーザーは強力なマルチコアシステムに移行しているため、インテルでは多くのコアを活用するロードマップを引き続き採用することを計画しています。スレッド依存のシステムやボトルネックになる演算はすべて、チームの課題です。開発者は、Unreal Engine* の最新のバージョンをダウンロード (英語) し、インテル® デベロッパー・ゾーンに参加して、さまざまな情報を確認することを推奨します。

関連情報

Unreal Engine* 4 最適化ガイド (英語)

クロス・シミュレーションのためのプロセッサーの最適化 (英語)

破壊可能メッシュの設定 (英語)

CPU スケーリングのサンプル (英語)

コンパイラーの最適化に関する詳細は、最適化に関する注意事項を参照してください。