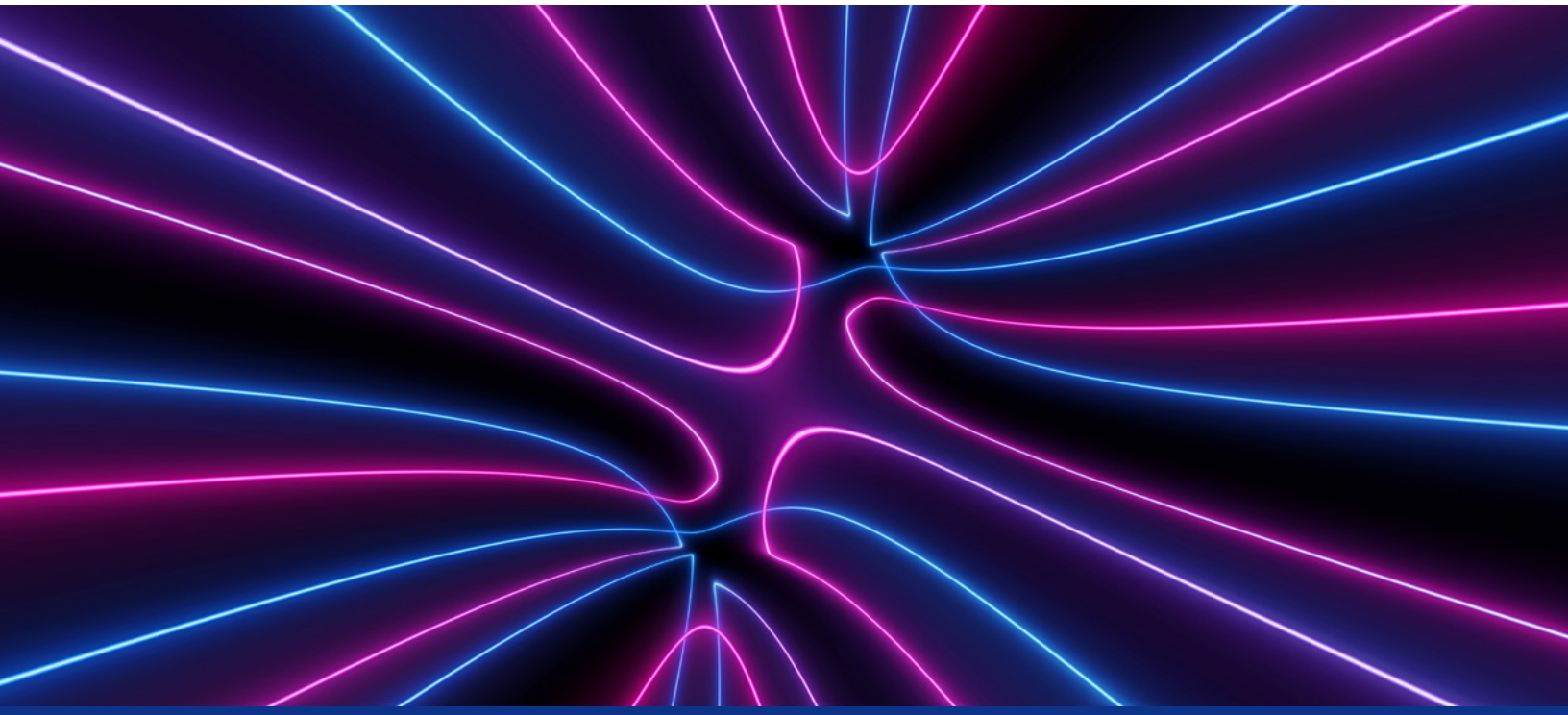


ゲームが GPU 依存ではありませんか？

GPU およびデバイス・コンテキスト・キュー解析により容易に判断

Oleg Fedyaev インテル コーポレーション グラフィックス・ソフトウェア・エンジニア

コンピューター・グラフィックスは、コンピューターでの作業、映画の視聴、スマートフォンの使用、さらには車の運転まで、日常生活において重要な役割を果たしています。グラフィックス・プロセッサーのパフォーマンスは、この 10 年間で劇的に向上しました。そして、その過程におけるビデオゲーム業界の影響は小さくありません。同時に、GPU 機能の継続的な成長は、ゲーム開発者に新たな可能性をもたらし、ハードウェアを最大限に活用する画期的なレンダリング手法やエフェクトの考案を促してきました。しかし、GPU ハードウェア開発者とゲーム開発者のこの競争には課題があります。多くの場合、革新的なレンダリング手法はハードウェアの制限に直面します。

この記事では、高レベルのシステム概要を使用して、ゲームが GPU 依存かどうかを素早く簡単に判断する方法を紹介します。

ゲーム・パフォーマンスの基本

ビデオゲーム制作には多額の費用がかかります。パフォーマンス最適化への投資は、ゲーム・プロジェクトの収益性に影響する重要な工程です。通常、ゲームのジャンル—アクション、アドベンチャー、戦略、その他—によりパフォーマンス要件は異なります。アーティファクトの描画に顕著な停滞や遅延が生じ、ゲームが目に見えて遅い場合は、対処が必要なパフォーマンスの問題が存在します。

ゲームのパフォーマンスを測定するメトリックは、1 秒あたりにレンダリングされるフレーム数 (FPS) を示すフレームレートです。FPS はさまざまなアプリケーションのベンチマークとランク付けに使用され、FPS が高いほうが優れています。ほとんどの場合、このアプローチを使用できます。例えば、FPS が低いと動きが多いアクションゲームは見栄えが良くありません。

最近のゲームは、複数のコンポーネントで構成されており複雑です。

- グラフィックスのレンダリング
- 物理特性の計算
- サウンドの再生
- スクリプトの実行
- ネットワークのホスティング
- その他

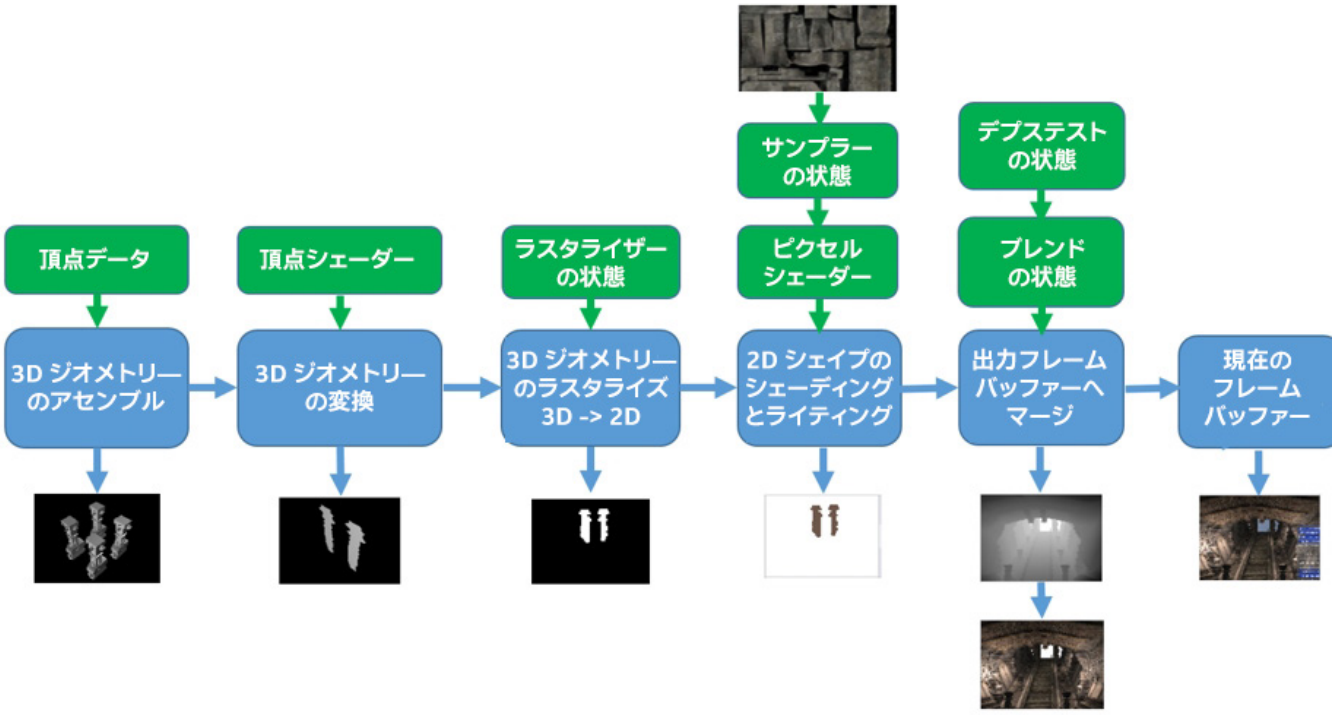
各コンポーネントは、個別にまたは連動して、ゲームのパフォーマンスに影響します。そのため、アプリケーションが GPU 依存か CPU 依存かを特定することは容易ではありません。

レンダリングはゲームを構成するコンポーネントの 1 つですが、グラフィックスはゲームの独特な作風、スピリット、雰囲気を作り出す上で重要であるため、レンダリングから解析を開始することは合理的です。

従来のレンダリング・パイプライン

グラフィックス・レンダリング・パイプライン、グラフィックス・プログラミング・モデル、およびこの処理におけるグラフィックス・ドライバーの役割を理解せずに、GPU がパフォーマンス・ボトルネックかどうかを正確に判断することはできません。アプリケーション・コードからハードウェアにまで、スタック全体の GPU アクティビティーを徹底的に解析するには、かなりの専門知識が必要です。幸いにも、基本的なパフォーマンス解析を実行すれば、詳しい調査を行わなくても GPU 全体の利用状況を確認できます。

レンダリング・パイプラインの動作にはリソースと状態が伴います。パイプラインにバインドされたリソースは、何を、どこでレンダリングすべきかを指定します。リソースは、適切な形式で記述されたジオメトリー、テクスチャー、およびレンダリング・ターゲットで構成されます。ラスタライズ・パラメーター、デプス条件のテスト、属性のブレンド、およびその他の状態は、画面上にイメージを生成するためこれらのリソースをどのように解釈して処理するかを指定します。レンダリング・リソースと状態は、シェーダーと呼ばれる GPU プログラムと緊密に連携して、レンダリング・パイプラインのさまざまなステージで実行されます (**図 1**)。



1 **レンダリング・パイプライン**

従来のレンダリング・パイプラインは、ソースデータを受け取り、各ステージで順次処理します。最終的な結果が得られるまで、これを繰り返します。レンダリングされたオブジェクトは、最初に仮想空間で変換され、次にスクリーンサーフェスに投影されます。その後、投影の可視部分が色付けされ、フレームバッファー内のほかのレンダリング・オブジェクトとマージされます。

グラフィックス・プログラミング・モデルは単純です。レンダリング・コマンドを正しい順序で送信するため、描画コンテキストを構成して開き、使用します。パイプラインにジョブの実行を指示する描画コマンドを呼び出す前に、必要なリソース、状態、およびプログラムをパイプラインにバインドする必要があります。この処理は、最終的なシーンがフレームバッファーに形成されるまで、オブジェクトのレンダリングに必要な回数だけ繰り返されます。最終的なシーンは、バッファー・スワップ・コマンドでスクリーンに出力されます。OpenGL*、DirectX*、Vulkan*、またはその他のグラフィックス API など、アプリケーションが使用する API に関係なく、この概念は同じです。

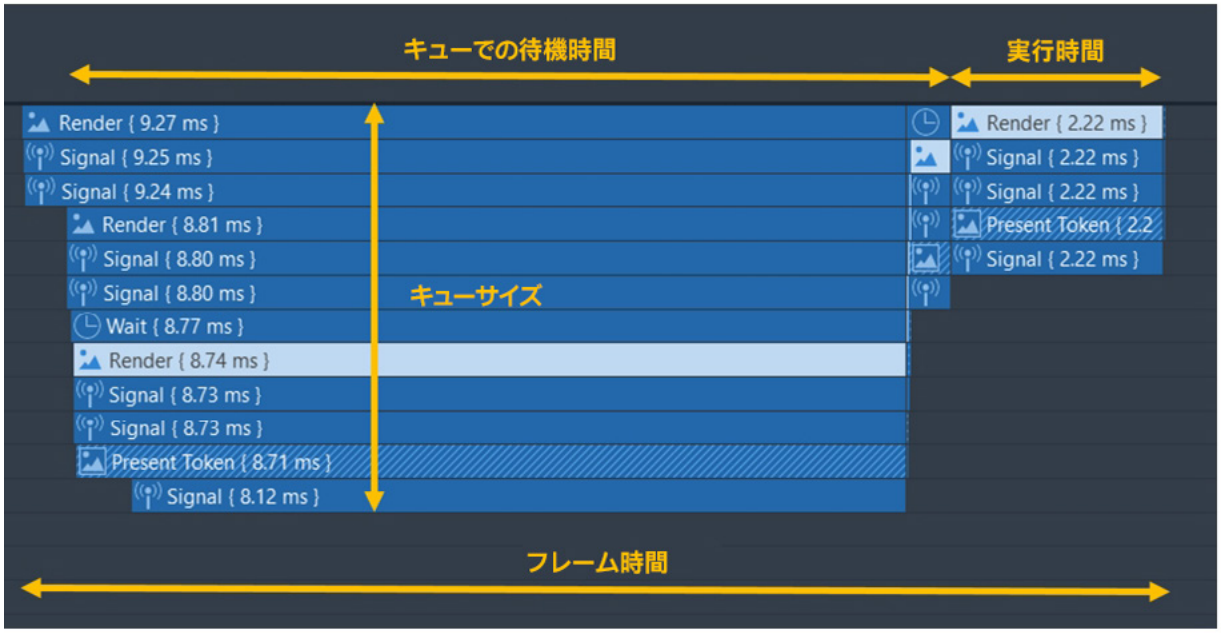
1 つの描画に多くの複雑な処理が含まれていることは明らかです。描画時間には、各処理にかかった時間が含まれます。個々の描画にかかる時間はさまざまで、合計フレーム・レンダリング時間に影響します。長いフレーム時間は GPU 依存を示す可能性があります。これは、グラフィックス・ドライバーのパフォーマンス・メトリックを基に GPU ロードを予測することで確認できます。

グラフィックス・ドライバー・アクティビティー

一般的なグラフィックス・プログラムは、グラフィックス・ドライバーを操作し、直接 GPU を操作することはありません。アプリケーションで描画コンテキストを開くと、ドライバー・デバイス・コンテキストと呼ばれるグラフィックス・ドライバーへのインターフェイスが暗黙的に作成されます。レンダリングを可能にするため、ドライバーは多くの処理を実行する必要があります。

- GPU 上でのメモリーブロックの解放と割り当て
- レンダリングに必要なリソースの CPU から GPU へのアップロード
- GPU 実行ユニットのレジスターの設定
- GPU プログラムのアップロード
- CPU への結果の転送
- その他

アプリケーション・コード内で一連のグラフィックス API を呼び出すと、ドライバーはそれらを GPU が処理可能な一連のコマンドに変換します。コマンドはすぐには GPU で実行されません。代わりに、コマンドバッファーに蓄積されます。ドライバーは、一連のコマンドをパケットにまとめて、実行デバイス・コンテキスト・キューにプッシュして、実行をスケジュールします (**図 2**)。



2 **コマンドバッファー**

デバイス·コンテキスト·キューには、さまざまなコマンドタイプからなるさまざまなパケットタイプが含まれます。パケットタイプは、パケット内で多数のコマンドタイプにより決まります。各パケットは、前のパケットの最後のコマンドが GPU で実行を完了するまでキューで待機します (例えば、**図 2** で選択されている Render パケットを参照)。

デバイス・コンテキスト・キューを調査することで、パフォーマンスに関する有用な情報が得られます。例えば、大きなキューサイズは、多くの場合 GPU に送信されるグラフィックス処理量が多いことを示します。長いパケット実行時間は、計算負荷の高い描画処理が原因の可能性があります。長いパケット待機時間は、非効率なレンダリング・アルゴリズムや同期が原因の可能性があります。

1 つのフレームに関連するすべてのパケットが判明すると、キューに最初のパケットを送信してから、同じフレームを最後に送信したパケットの最後のコマンドが実行を完了するまでの時間を計算することで、おおよそのフレーム時間を予測できます。

しかし、フレーム時間が長くても、レンダリングを実行するグラフィックス·プロセッサーに関連付けられている GPU ハードウェア·キューを調査するまで、アプリケーションが GPU 依存かどうか判断することはできません。GPU は共有リソースであり、複数のアプリケーションがグラフィックスをレンダリングするため同時に利用できます。長いレンダリング時間は、GPU コンテキストを取得した別のアプリケーションとの同時実行の結果であるかもしれません。

ハードウェア GPU キュー (**図 3**) は、GPU 全体の利用状況を明確にします。このキューを使用して、GPU のビジー状態と、その時点でレンダリングされていたアプリケーションを特定できます。

3 **ハードウェア GPU キュー**

図 3 の GPU キューのスナップショットは、コマンドパケットの色で区別された少なくとも 2 つのアプリケーションが同時にレンダリングしていることを示しています。どちらのアプリケーションも GPU 依存ではありません。青色で示されたアプリケーションのフレーム時間は約 11 ミリ秒であり、これは約 80FPS に相当します。80FPS は通常十分に高い値です。緑色は非常に小さなフレーム (それぞれ約 5 ミリ秒) のバックグラウンド・プロセスのようです。さらに、実行コマンドパケット間に GPU がアイドル状態であった期間を示す複数のギャップがあるため、GPU はそれほどビジーではありません。

ソフトウェアとハードウェア・キューの解析は、信頼性の観点から非常に有望です。また、必要なパフォーマンス・データの収集方法が分かっているため、これらのキューは簡単に構築できます。

システム・イベント・トレース

Windows*、Linux*、macOS* など、使用するオペレーティング・システムに関係なく、システムのイベント・トレース・レイヤーに接続することができます。このレイヤーは、異なるシステムモジュール内の主要な実行ポイントに関連付けられたさまざまなイベントを記録します。一部のイベントは、パフォーマンス解析の対象です。グラフィックス・ドライバーも例外ではありません。ドライバーはコマンドパケットをデバイス・コンテキスト・キューにプッシュしたり、コマンドパケットを GPU へアップロードしたり、またはコマンドパケットの最後のコマンドを実行するたびに、対応するイベントをシステム・トレース・レイヤーへ送信するため、それらを簡単に収集できます。例えば、Windows* でデバイス・コンテキスト・キューと GPU コマンド・パケット・キューを構築する場合、Windows* イベント・トレーシング (ETW) システムの Microsoft-Windows-DxgKrnl プロバイダーからいくつかのイベントを収集する必要があります。

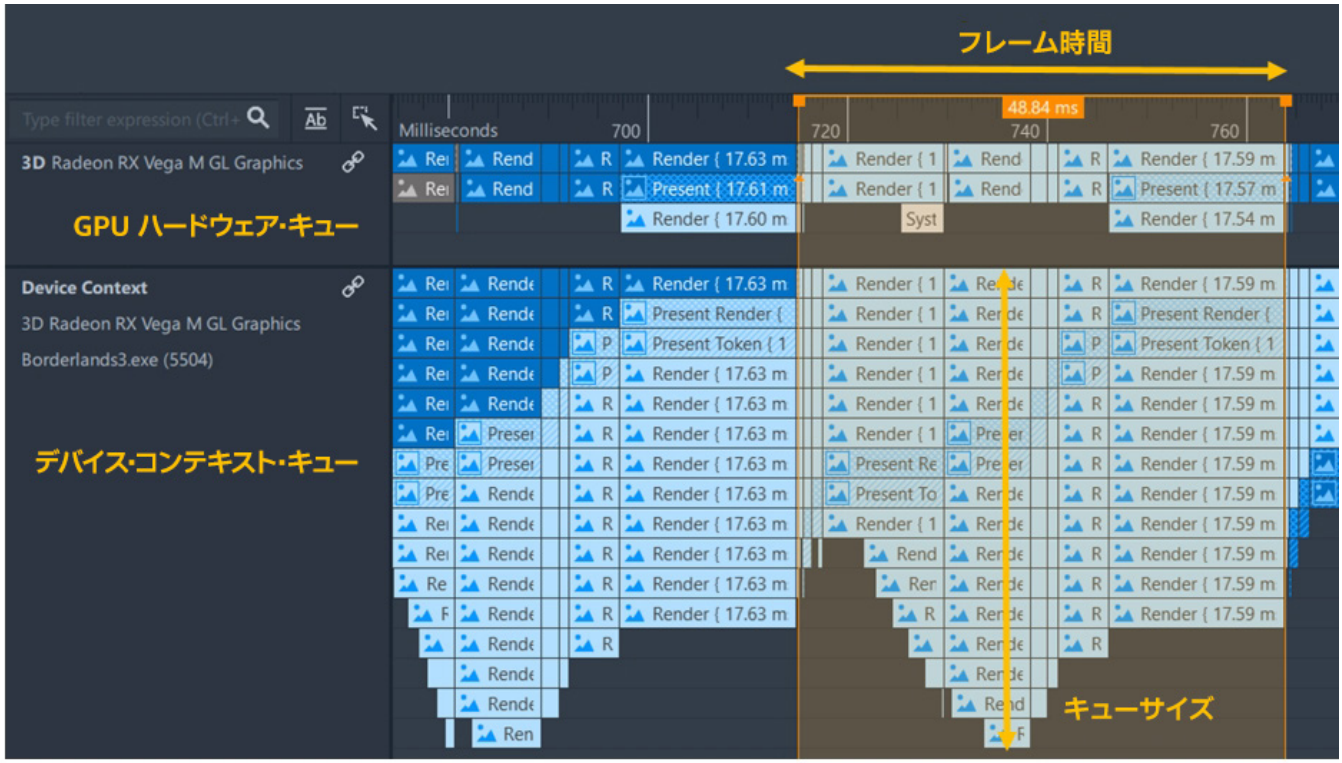
イベントデータにエンコードされているさまざまな属性により、異なるイベントを組み合わせてキュー内の各パケットの現在の状態をいつでも区別することが可能です。

グラフィックス・アプリケーション解析

システム・イベント・トレースは、ドキュメントが充実しており、どのプラットフォームでも使用できます。システム・トレース・データの収集または視覚化には、多くのツールを利用できます。しかし、デバイス・コンテキスト・キューと GPU ハードウェア・キューを適切に、同時に解析できるツールは限られています、**インテル® GPA グラフィックス・トレース・アナライザー** (英語) はその 1 つで、高レベルのシステム解析から単一フレームの描画ごとの解析まで、さまざまな詳細レベルでグラフィックス・アプリケーションのパフォーマンスを解析できるように設計されています。

ここで紹介したことを、実際のグラフィックスを多用するゲームに適用してみましょう。実験を予測可能なものにするため、中程度の性能のグラフィックス・プロセッサーを搭載したワークステーションでテストします。2019 年 9 月にリリースされた大人気のアクション RPG シューターゲームの最新作である **Borderlands* 3** (英語) を使用します。これをインテル® HD グラフィックス 630 内蔵グラフィックスと AMD Radeon* RX Vega M GL ディスクリート・グラフィックスの 2 つのグラフィックス・プロセッサーが搭載された **Windows* 10 搭載ミニ PC、インテル® NUC エンスージアスト - NUC8i7HVKVA** で実行します。このゲームをワイドスクリーン・モニターに適した 2560x1440 解像度で実行し、ゲームのすべてのグラフィックス・オプションをハイプロファイルに設定すると、ゲームエンジンはレンダリングに最適なグラフィックス・プロセッサーを選択します。このデバイスでは、Radeon* Vega が選択されました。

5 分間ゲームをプレイして最初に気になったことは、マウスやゲームパッドなどの入力デバイスの状態が変わってから、スクリーンのシーンが切り替わるまでの遅延です。一部の移動オブジェクトのアニメーションがやや不自然に見えます。インテル® GPA グラフィックス・トレース・アナライザーでトレースを収集して開くと、デバイス・コンテキスト・キューと GPU ハードウェア・キューのタイムライン・トラックから GPU 依存シナリオのすべての属性が一目で確認できます (**図 4**)。



4 **デバイス・コンテキスト・キューと GPU ハードウェア・キュー**

GPU キューにはギャップ (アイドル時間) がありません。完全にビジーで、ゲームから送信されたコマンドを継続的に実行しています。デバイス・コンテキスト・キューのサイズは十分に大きく、多くのグラフィックス処理が準備され、レンダリングを待機しています。単一フレーム内の GPU で実行されたすべてのコマンドパケットを選択すると、フレーム時間が分かります。フレーム時間は約 48.8 ミリ秒 (約 21FPS) であり、このアクションゲームには十分ではありません。このようなゲームで最適なゲーム体験を実現するには、通常 60FPS 以上が必要です。

高レベルのシステム概要を使用して GPU 依存シナリオを解析し、ソフトウェアとハードウェア・キューの詳細を調査することでいくつかの利点が得られます。この解析は簡単で正確です。グラフィックスの専門知識を必要とせず、レンダリングに使用されるグラフィックス API のタイプに依存しません。また、どのプラットフォームでも実行することがき、システムトレースから対応するパフォーマンス・イベントを収集できます。

関連情報

- **インテル® グラフィックス・パフォーマンス・アナライザー (インテル® GPA)**
- **インテル® グラフィックス・トレース・アナライザー (英語)**
- **インテル® グラフィックス・パフォーマンス・アナライザー・クックブック (英語)**