インテルの x86 プラットフォーム向け Unity* 最適化ガイド: パート 1

この記事は、インテル® デベロッパー・ゾーンに公開されている「Unity* Optimization Guide for Intel x86 Platforms: Part 1」の日本語参考訳です。

目次

x86 プラットフォームのパフォーマンスを最大限に引き出すため、プロジェクトに適用可能ないくつかの最適化があります。このガイドでは、Unity* プロジェクトのパフォーマンスの向上に役立つさまざまなツールと Unity* ソフトウェアの機能を紹介し、テクスチャー品質、バッチ処理、カリング、ライト焼き付け、HDR エフェクトなどの使用法についても述べます。

このガイドをお読みになることで、Unity* でのパフォーマンスの問題とその原因の特定方法、主な最適化手法、優れたゲーム開発手法を習得することができます。最初に、アプリケーションの hotspot を特定するのに役立ついくつかのツールを見てみましょう。

ツール

このガイドでは、3 つの主要ツール (Unity* Profiler、インテル® GPA System Analyzer、インテル® GPA Frame Analyzer) について説明します。それぞれ強力なツールであり、個別に使用した場合でも、堅実なゲーム開発を支援しますが、3 つを組み合わせて使用することで、ゲームを大幅に合理化および最適化できます。

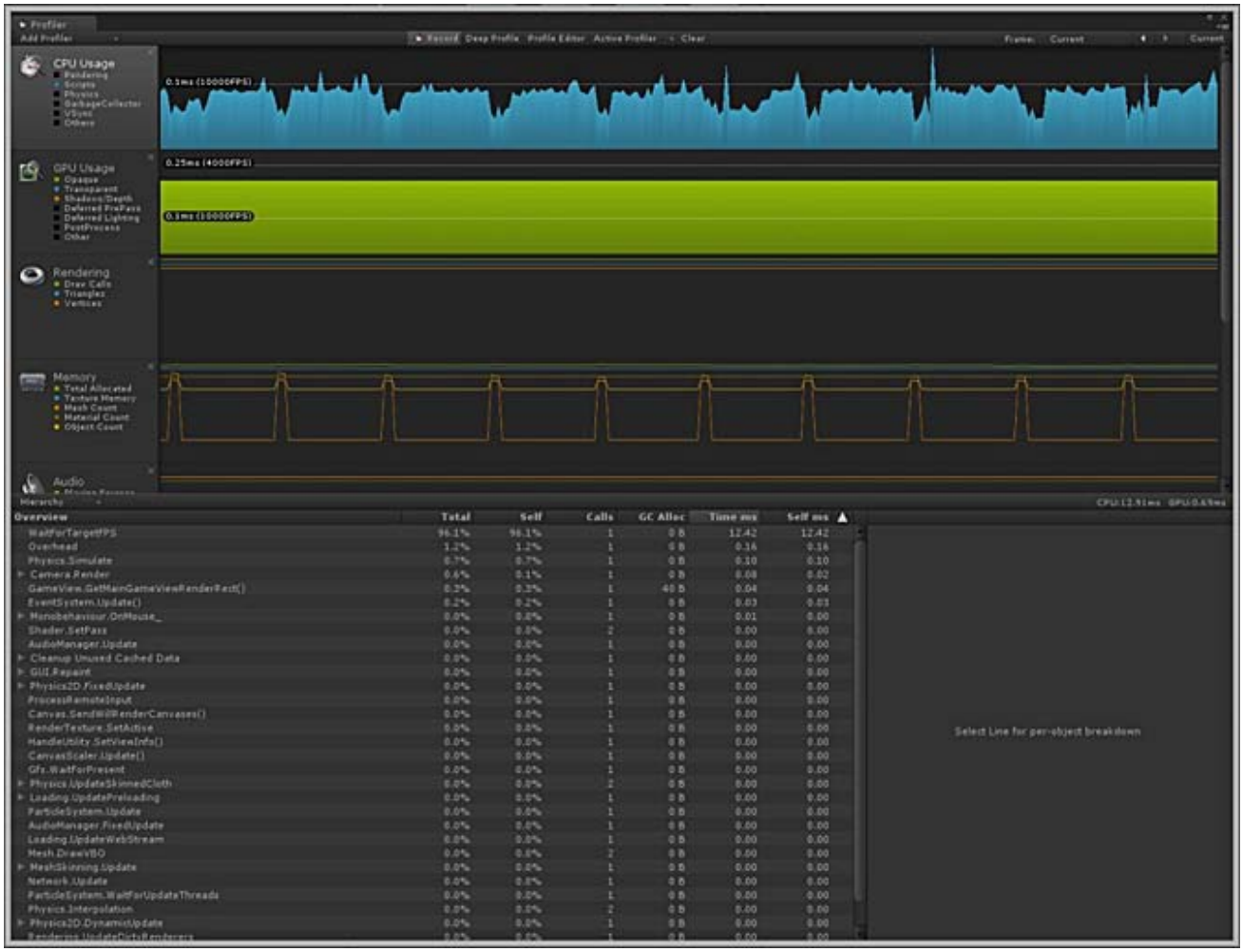


図 1. Unity* Profiler メイン画面

Unity* Profiler

Unity* Profiler (図 1) は、非常に強力な Unity* ツールで、プロジェクトのサブシステムの問題を特定するのに役立ちます。プロファイラー・グラフにはさまざまなサブプロファイラーが表示され、それぞれ特定のハードウェアのメトリックを示します。現在利用可能なサブプロファイラーは、CPU Usage (CPU 使用状況)、GPU Usage (GPU 使用状況)、Rendering (レンダリング)、Memory (メモリー)、Audio (オーディオ)、Physics (物理演算)、Physics 2D (2D 物理演算) です。各サブプロファイラーは、さらにドリルダウン可能な関連コンポーネントのセクションに分割されます。例えば、CPU Usage には、Rendering (レンダリング)、Scripts (スクリプト)、Physics (物理演算)、GarbageCollector (ガベージコレクター)、Vsync (垂直同期)、Others (その他) セクションが含まれます。

グラフセクションの下にある Overview (概要) ウィンドウでは、さまざまな Unity* サブシステムのタイミング情報、メモリー割り当て状況を含むメトリックの一覧を確認できます。レンダリングからガベージ・コレクションまで、あらゆるものがここに表示されます。アプリケーションで最も時間を費やしているセクションに注目して最適化の可能性を確認するとよいでしょう。グラフの任意のセクションをクリックすると、プロファイラーの更新が中断し、ハイライトしたフレームを調査できます。

Unity* Profiler は、エディターまたはスタンドアロン・ビルドで、実行中のアプリケーションにアタッチできます。正確なタイミングを取得するには、エディターのオーバーヘッドを避けるため、常にスタンドアロン・ビルドにアタッチすることを推奨します。ウィンドウ上部にある [Active Profiler] ボタンから、ADB (Android™ Debug Bridge) により検出された、またはネットワーク上の利用可能な [Android Player] インスタンスを選択します。

別のオプションとして、アプリケーションの詳細なプロファイルを行う [Deep Profile] があります。このオプションは、実際にすべてのモノコードをインストルメントし、プロファイルのオーバーヘッドが大きくなるため、一般には推奨しません。幸い、Unity* では、インストルメントするコードセグメントを明示的に指定することができます。図 2 は、指定したラベルとともにプロファイラーに表示するコードをインストルメントする方法を示します。

```
// Update is called once per frame
0 references
void Update () {
    if(mMyState == STATE.GOOD)
    {
        Profiler.BeginSample("Object fetching the good way");
        for(int i = 0; i < PlainCubeManager.Singleton.mPlainCubes.Count; ++i)
        {
            PlainCubeManager.Singleton.mPlainCubes[i].DoSomething();
        }
        Profiler.EndSample();
    }
    else
    {
        Profiler.BeginSample("Object fetching the bad way");
        PlainCube[] objects = FindObjectsOfType<PlainCube>();
        for(int i = 0; i < objects.Length; ++i)
        {
            objects[i].DoSomething();
        }
        Profiler.EndSample();
    }
}
```

図 2. Unity* Profiler で使用するコードセグメントの設定

インテル® GPA System Analyzer

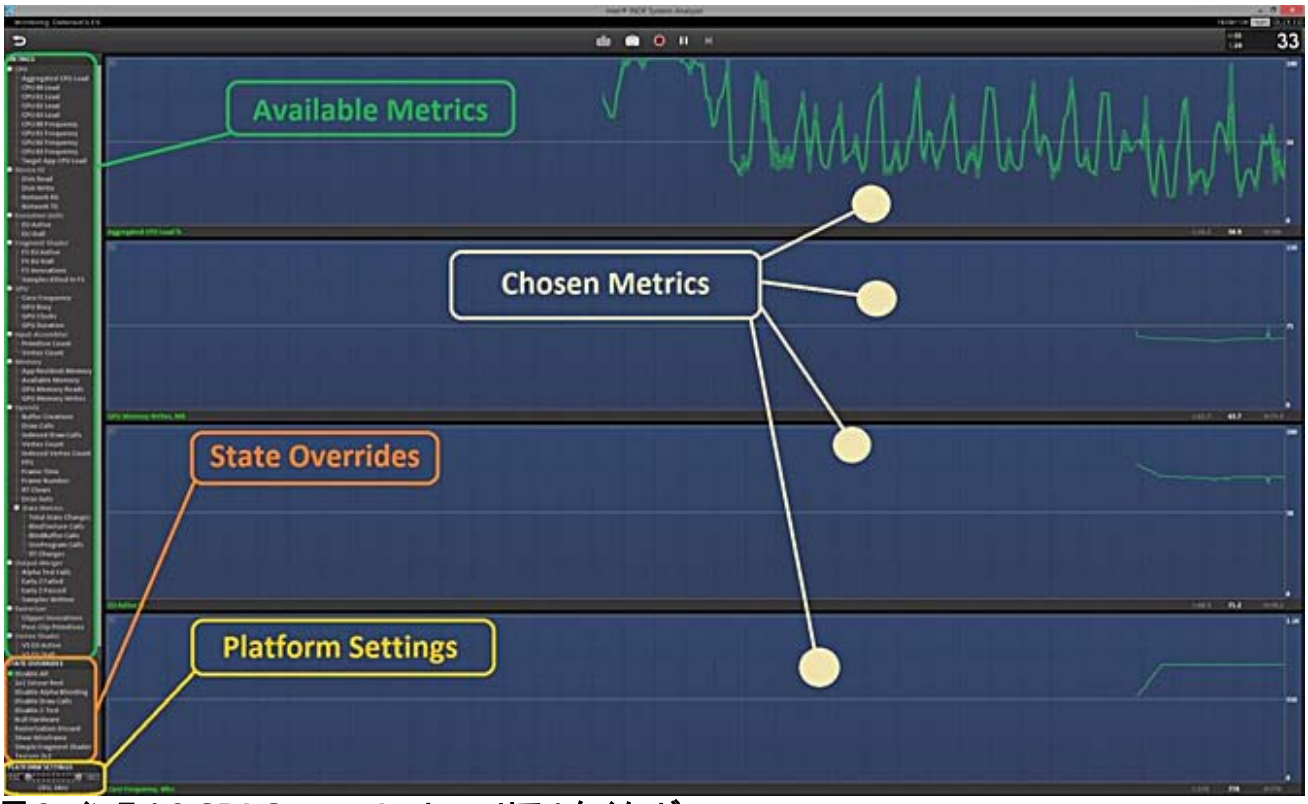


図 3. インテル® GPA System Analyzer リアルタイム・ビュー

インテル® Graphics Performance Analyzers (インテル® GPA) は、ゲームやグラフィックスを多用するアプリケーションの高速化を支援する、グラフィックス解析と最適化のツールスイートです。開発者がグラフィックス API 呼び出しの詳細な解析を実行し、パフォーマンスの問題がどこで発生しているかを特定できるように、幅広い機能を提供します。このガイドで紹介するテスト結果とメトリックは、インテル® GPA からのものです。インテル® GPA を利用することで、Windows® 上で DirectX® アプリケーションのグラフィックス・ワークロードを確認したり、一部のインテル® プロセッサー・ベースの Android™ システムで OpenGL* ES アプリケーションのグラフィックス・ワークロードを確認することができます。OpenGL* API 呼び出しを直接監視することはできませんが、インテル® GPA System Analyzer を利用して、OpenGL* ゲームを実行中に GPU と CPU メトリックを確認できます。グラフィックス API に関係なく、OpenCL* アクティビティーを含む詳細な CPU 負荷の確認にインテル® GPA Platform Analyzer を利用することもできます。インテル® GPA には、インストルメンテーションを追加する API も用意されているため、さらに詳しい調査も可能です。インテル® GPA のツールセットは、Android™ システムおよびデスクトップで利用できます。インテル® GPA の詳細とダウンロードについては、www.isus.jp/intel-gpa/ を参照してください。

インテル® GPA を利用する場合、最初にリアルタイムのパフォーマンス情報を収集します。リアルタイム・データの表示モードは 2 つあります: アプリケーションと一緒に実行する HUD (Heads-Up Display) とネットワーク経由でテストシステムに接続する System Analyzer。どちらも DirectX® パイプライン (および一部のインテル® プロセッサー上では OpenGL* ES パイプライン) からのメトリック、CPU 使用状況、システムの電力状態を表示します。サポートされるインテル® プロセッサー・グラフィックス・システムでは、詳細な GPU ハードウェア・メトリックも確認できます。HUD と System Analyzer で簡単なテストを行うことで、パフォーマンスの問題を素早く検出できます。HUD と System Analyzer の機能については、インテル® GPA ドキュメントを参照してください。

図 4. インテル® GPA System Analyzer HUD

メトリックの値を解析に含めるには、左のサイドバーからドラッグしてメインの描画領域にドロップします。ツールは、ARM* デバイスでも動作しますが、インテル® プロセッサー・ベースのハードウェアで利用可能なすべてのメトリックを使用できません。詳細は、インテル® GPA チュートリアル (英語) [Windows® | OS X*] を参照してください。インテル® プロセッサー・ベースのハードウェアでは、次のメトリックを利用できます。

- CPU
- Device IO (デバイス I/O)
- Execution Units (実行ユニット)
- Fragment Shader (フラグメント・シェーダー)
- GPU
- Input-Assembler (入力アセンブラー)
- Memory (メモリー)
- OpenGL*/DX
    - State Metrics (ステートメトリック)
- Output-Merger (出力マージャー)
- Power (電力)
- Rasterizer (ラスタライザー)
- Vertex Shader (頂点シェーダー)

CPU ボトルネックの場合、DirectX® と OpenGL* ワークロードの解析に Platform Analyzer が役立ちます。CPU アクティビティーのキャプチャー・トレースが表示されます。コードにインストルメンテーションを追加すると、CPU で実行中の個々のタスクを関連付け、DirectX®、ドライバー、GPU を通過する状態を確認できます。ボトルネックの特定を支援するため、インテル® GPA には、状況の変化に対するフレームレートの変動をチェックできる [State Overrides] セクション (図 5) があります。以下にいくつかの例を示します。

図 5. 利用可能なオーバーライド

- **Texture 2x2 (テクスチャー 2x2)**
  - 高画質のテクスチャーからのデータフェッチには時間がかかります。このオプションは、シーン中のすべてのテクスチャーを 2x2 テクスチャーに置換します。このオプションをオンにすることでパフォーマンスが大きく変わる場合、いくつかのテクスチャーのサイズを落とすことでフレームレートを向上できる可能性があります。
- **Null Hardware (Null ハードウェア)**
  - このオプションは、GPU の速度制限がない場合をシミュレーションします。このオプションをオンにすることでフレームレートが向上する場合、コードはドライバーまたは CPU に依存している可能性が高いと思われます。
- **Disable Draw Calls (描画呼び出しの無効化)**
  - このオプションは、ドライバーが非常に高速な場合をシミュレーションします。このオプションをオンにすることでフレームレートが変動する場合、コードはドライバーに依存している可能性があります。
- **Simple Fragment Shader (単純なフラグメント・シェーダー)**
  - このオプションは、すべてのシェーダーを非常に単純なフラグメント・シェーダーに置換します。このオプションをオンにすることで変動が見られる場合、シェーダーのパフォーマンスを最適化すべきです。

[State Overrides] セクションの下にある [Platform Settings] スライダーを利用すると、CPU をさまざまな周波数で実行することができます。ゲームやアプリケーションが任意のデバイス上で最大フレームレートで実行している場合であっても、この機能を使用してボトルネックがないかどうかテストすることができます。また、ゲームやアプリケーションが幅広いデバイスで実行可能かどうか検証することもできます。さらに、特定の周波数でテストすることで、インテル® ターボ・ブースト・テクノロジーなどによる影響を排除できます。

最後に、ウィンドウ上部のカメラアイコンをクリックすると、フレームをキャプチャーできます。System Analyzer は、ゲーム/アプリケーションの 1 フレームの生成に関係するすべてのもの (状態変更、タイミング、テクスチャーなど) を記録し、ファイルに保存します。このファイルを Frame Analyzer で開き、詳しく調べることができます。

インテル® GPA Frame Analyzer

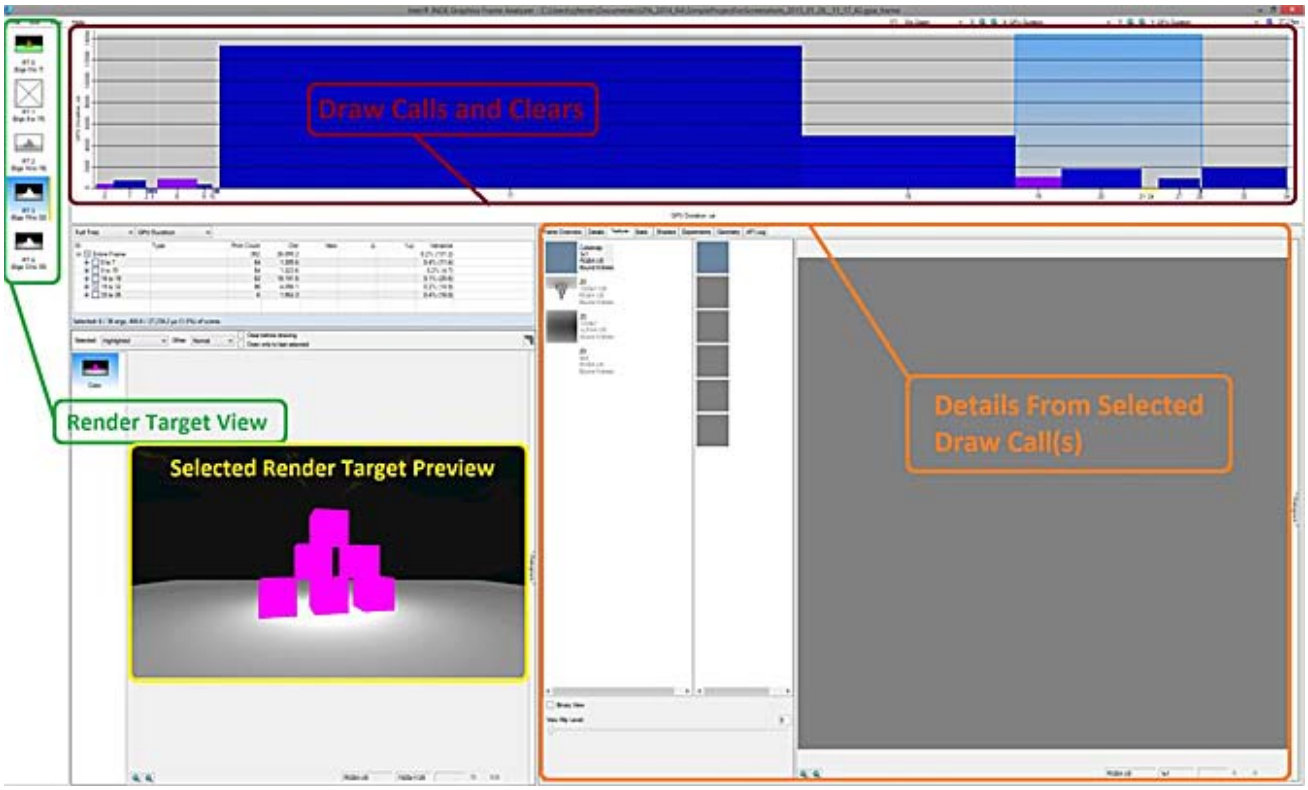
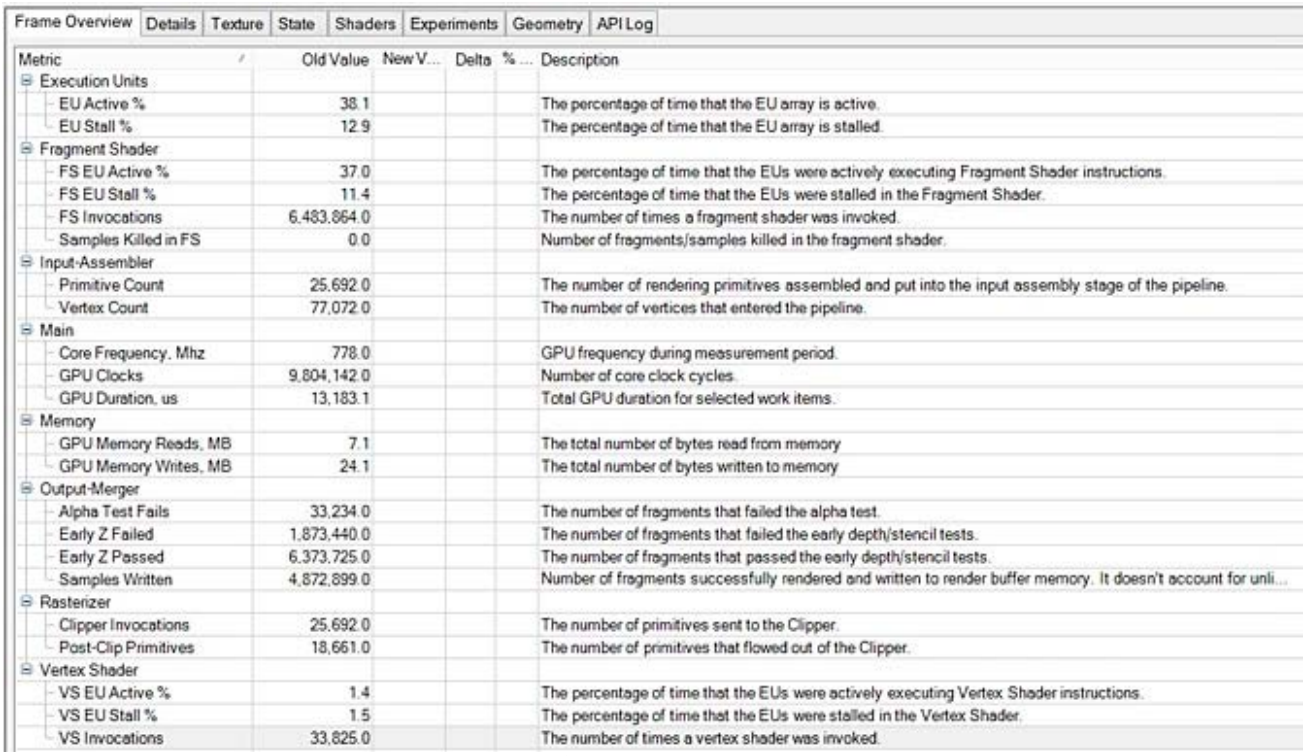


図 6. インテル® GPA Frame Analyzer に表示されたレコードの変更と関連フレーム情報

Frame Analyzer (図 5) では、1 つのフレーム・キャプチャーを開くことができます。キャプチャーされたフレームには、すべての状態変更、リソース、タイミング情報、その他のレコードが含まれています。ウィンドウ上部のグラフは、フレームで記録された個々の描画呼び出しを示します。これらの描画呼び出しは、分かりやすいように、レンダーターゲットごとに表示されています。グラフの X 値と Y 値は、左上のドロップダウン・メニューから変更できます。左側には、レンダーターゲットのリストがあります。左下では、現在ハイライトされている描画呼び出しのプレビューとフレーム表示を確認できます。各種オプションを利用して、描画済みのピクセルをハイライトしたり、それらを通常表示のままにしたり、ビューをカスタマイズできます。選択されていないもののプレビューへの影響も調整できます (非表示/表示)。右下のタブを利用して、現在選択されている描画呼び出しに関する次のような詳細を得られます。

- **Frame Overview (フレームの概要)**
    - o フレーム全体のタイミング/状態値は、GPU パイプラインのステージごとに表示されます。



Frame Overview | Details | Texture | State | Shaders | Experiments | Geometry | API Log

Metric	Old Value	New V...	Delta	% ...	Description
Execution Units					
EU Active %	38.1				The percentage of time that the EU array is active.
EU Stall %	12.9				The percentage of time that the EU array is stalled.
Fragment Shader					
FS EU Active %	37.0				The percentage of time that the EUs were actively executing Fragment Shader instructions.
FS EU Stall %	11.4				The percentage of time that the EUs were stalled in the Fragment Shader.
FS Invocations	6,483,864.0				The number of times a fragment shader was invoked.
Samples Killed in FS	0.0				Number of fragments/samples killed in the fragment shader.
Input-Assembler					
Primitive Count	25,692.0				The number of rendering primitives assembled and put into the input assembly stage of the pipeline.
Vertex Count	77,072.0				The number of vertices that entered the pipeline.
Main					
Core Frequency, Mhz	778.0				GPU frequency during measurement period.
GPU Clocks	9,804,142.0				Number of core clock cycles.
GPU Duration, us	13,183.1				Total GPU duration for selected work items.
Memory					
GPU Memory Reads, MB	7.1				The total number of bytes read from memory
GPU Memory Writes, MB	24.1				The total number of bytes written to memory
Output-Merger					
Alpha Test Fails	33,234.0				The number of fragments that failed the alpha test.
Early Z Failed	1,873,440.0				The number of fragments that failed the early depth/stencil tests.
Early Z Passed	6,373,725.0				The number of fragments that passed the early depth/stencil tests.
Samples Written	4,872,899.0				Number of fragments successfully rendered and written to render buffer memory. It doesn't account for unli...
Rasterizer					
Clipper Invocations	25,692.0				The number of primitives sent to the Clipper.
Post-Clip Primitives	18,661.0				The number of primitives that flowed out of the Clipper.
Vertex Shader					
VS EU Active %	1.4				The percentage of time that the EUs were actively executing Vertex Shader instructions.
VS EU Stall %	1.5				The percentage of time that the EUs were stalled in the Vertex Shader.
VS Invocations	33,825.0				The number of times a vertex shader was invoked.

図 7. Frame Overview セクションに表示される値

- **Details (詳細)**
    - o グラフ/ツリーで現在選択されている描画呼び出しのタイミング/状態値は、GPU パイプラインのステージごとに表示されます。

- **Texture** (テクスチャー) (図 8)
    - o 現在バインドされているテクスチャーのリスト
    - o [Texture] タブの左にあるサイドバーを利用して、圧縮、フォーマット、mip レベルなどを確認できます。

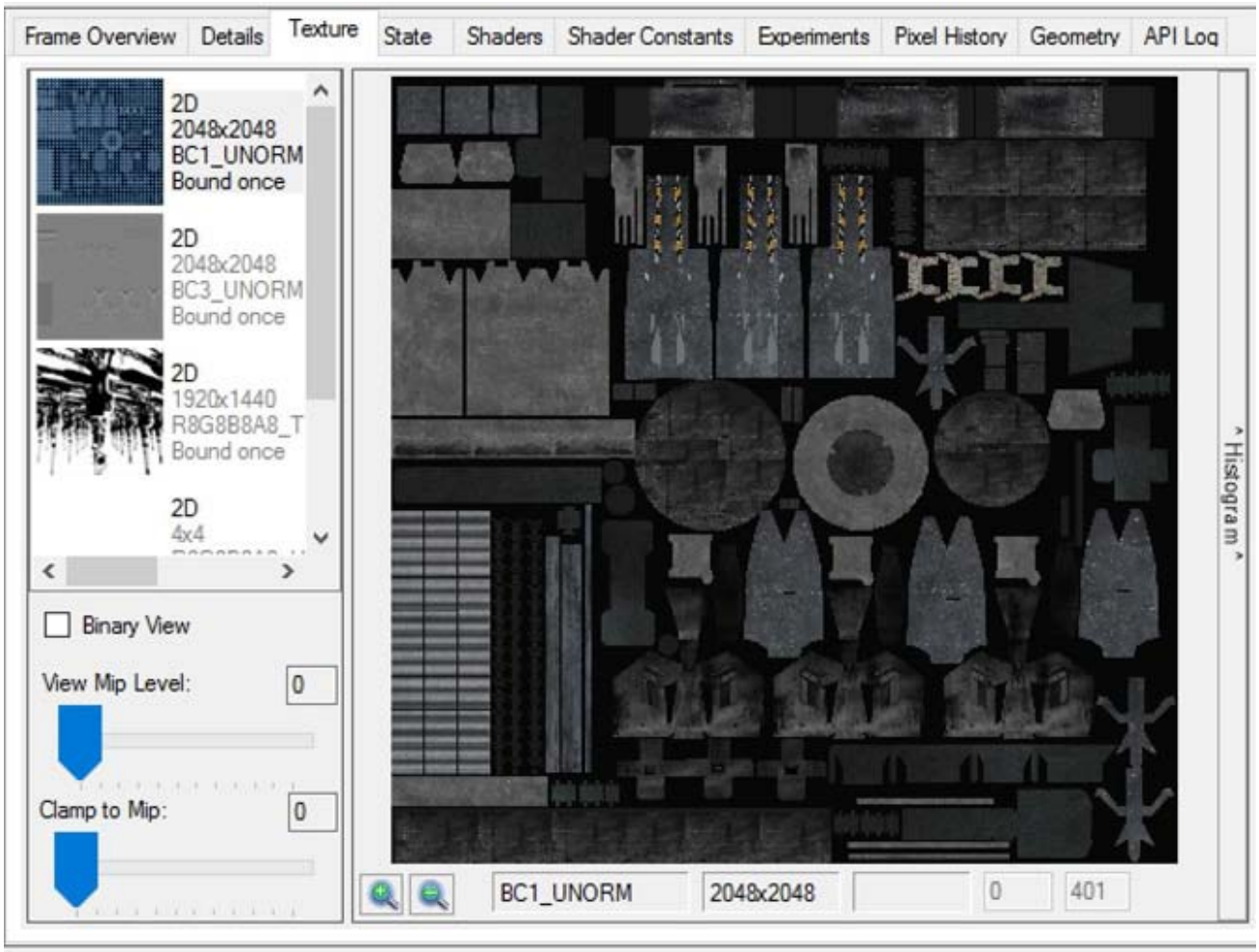


図 8. 描画呼び出しで使用されたテクスチャーのビュー

- **State (状態)**
    - o 選択されている描画呼び出しの状態設定
    - o 編集して、レンダーターゲットのプレビューとタイミングへの影響を確認できます。

- **Shaders (シェーダー)**
    - o 選択されている描画呼び出しで使用されているシェーダーが表示されます。
    - o シェーダーコードを編集して、シーンプレビューへの影響を視覚的に確認できます。シェーダーコードの変更は、タイミング情報にも反映されるため、特定の最適化によるフレームレートへの影響も分かります。

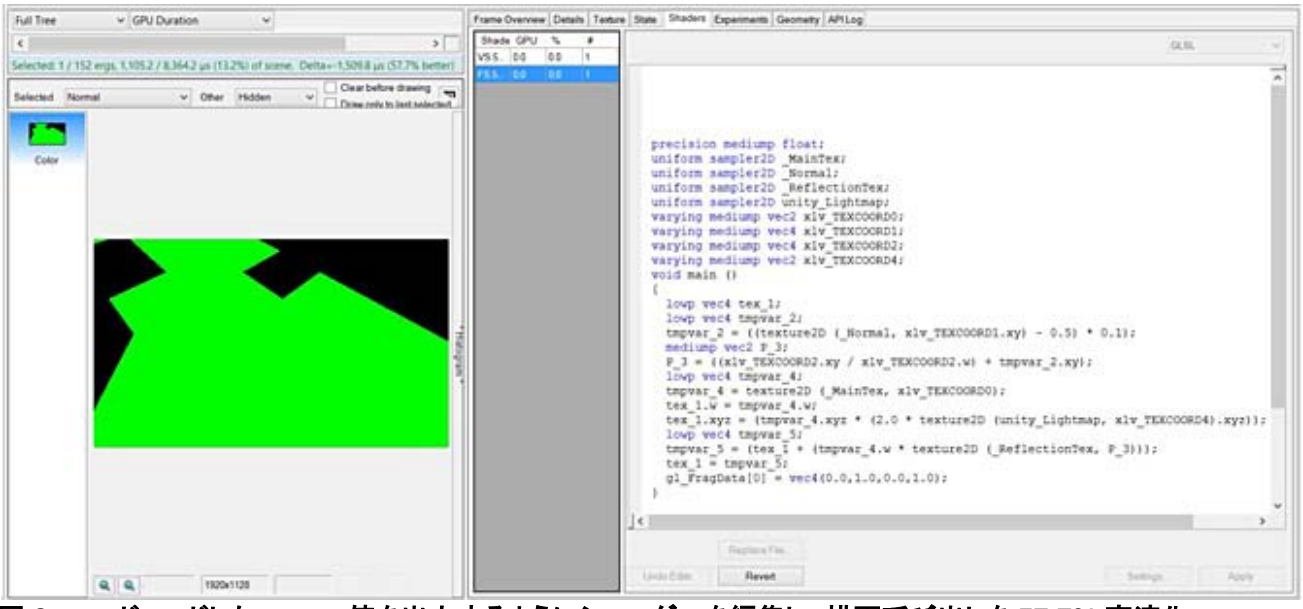


図 9. ハードコードした green 値を出力するようにシェーダーを編集し、描画呼び出しを 57.7% 高速化

- **Experiments (テスト)**
  - o System Analyzer のテストセクションと似ていますが、描画呼び出し単位で使用できます。

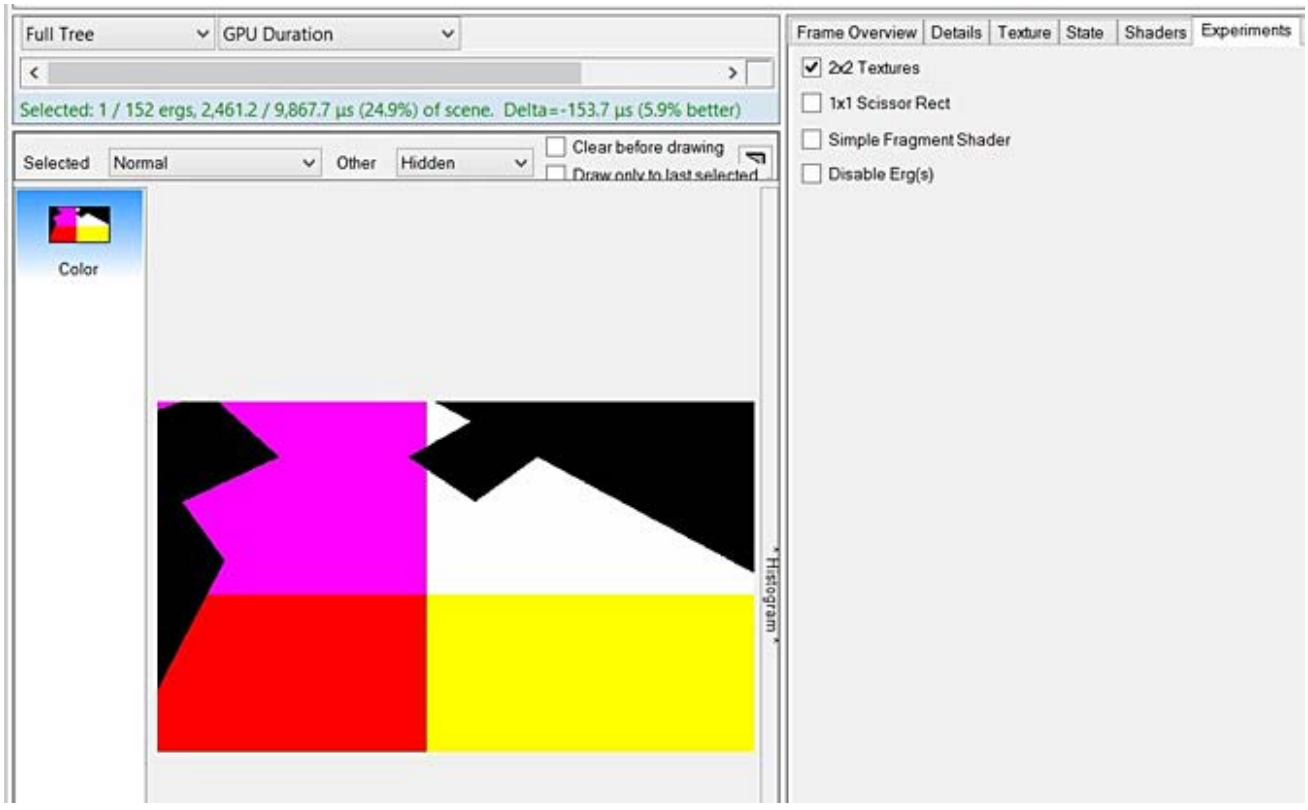


図 10. 2x2 テクスチャーに置換することで描画呼び出しを 5.9% 高速化できることを示すテスト。[Details] タブのグラフィックス・パイプラインの個々のポイントで変更を確認できます。

- **Geometry (形状)**
  - 選択されている描画呼び出しの形状データをウィンドウに 3D 表示します。

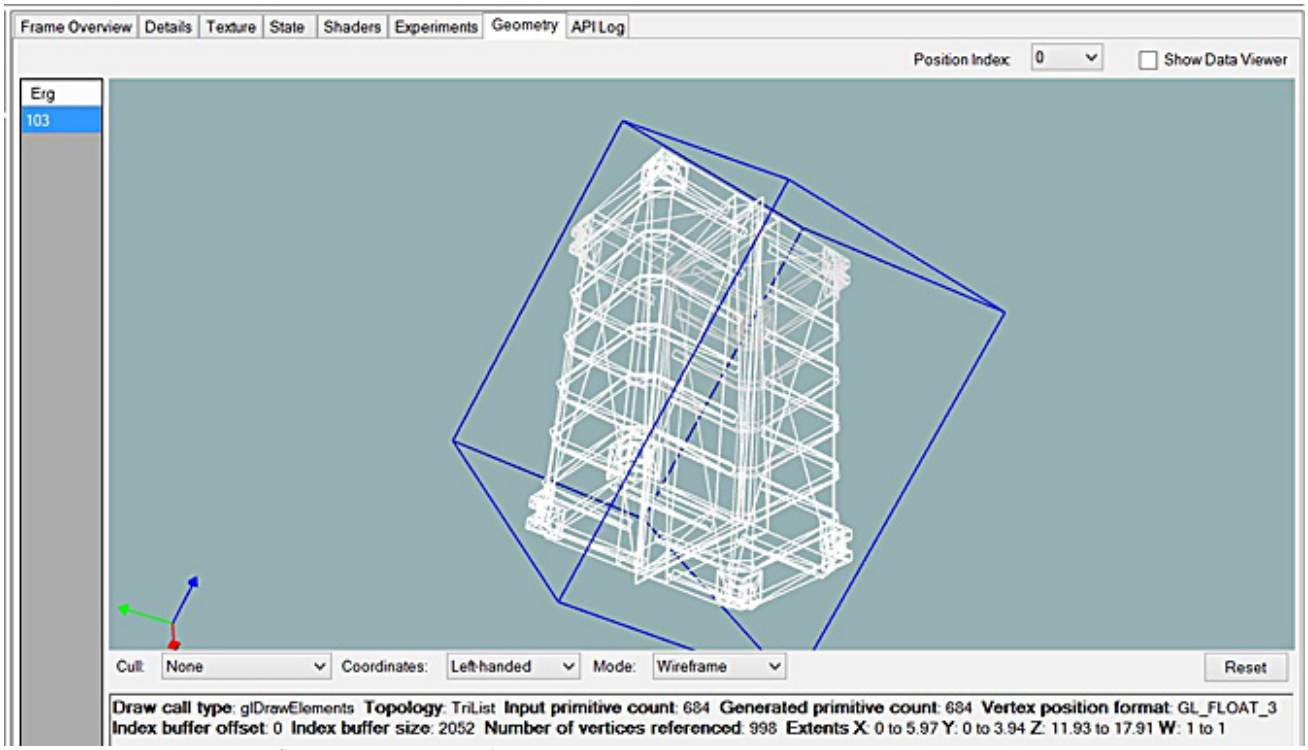


図 11. [Geometry] タブに表示されたモデル形状

- **API Log (API ログ)**
  - 選択されている描画呼び出しで使用されたすべての API 呼び出しを表示します。これは、パフォーマンスに影響する不要な状態変更を追跡するのに非常に便利です。

パート 2 に続く
インテル x86 プラットフォーム向け Unity* 最適化ガイド: パート 2 (英語)