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 3

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 3」の日本語参考訳です。

目次

- エディターの最適化
- オクルージョン・カリング
- LOD: 詳細レベル
- 影
- 複数のカメラ
- レンダーキューの順序
- ライトマップ
- メッシュコライダーの代わりに単純なコライダーを複雑なモデルに使用する

チュートリアルのパート 2 に戻る:
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 2

エディターの最適化

オクルージョン・カリング

オクルージョン・カリングは、カメラでほかのオブジェクトにオクルードされる (隠される) オブジェクトをカリングできる、Unity* が提供する機能です。単純な例としてファンタジー MMORPG を使用しましょう。プレーヤーが無秩序に広がる都市を含む巨大な城に向かって歩く場合、システムリソースを費やして城壁内のオクルードされた店やプレーヤーをすべてレンダリングする必要はありません。そういう場合はオクルージョン・カリングの出番です (図 24)。

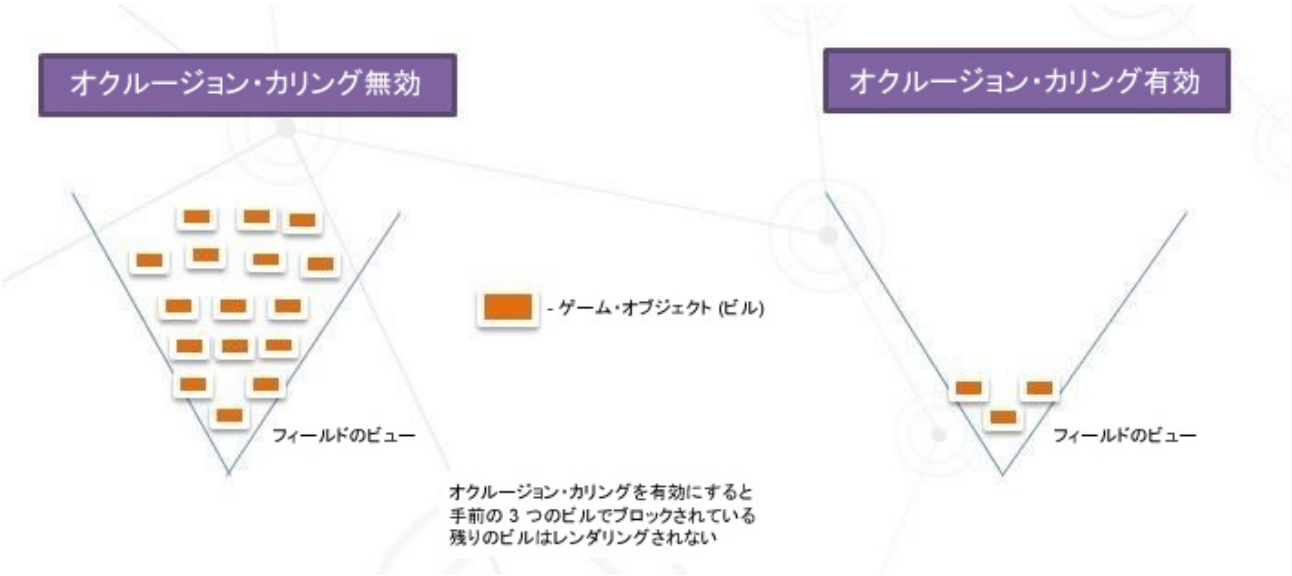

図 24. オクルージョンの説明

オクルージョン・カリングは、全体の描画呼び出し数を減らすことにより GPU 処理を (そして、オクルージョン・カリング計算にかかる時間が、ドライバー呼び出しで節約できる時間よりも短い場合は CPU 処理も) 軽減します。オクルージョン・カリングを設定するには、次の用語を知っておくと便利です。Unity* は、これらの用語を使用して、カリングを行うシーンを設定します。

- オクルーダー (Occluder) – ブロックされたオブジェクト (オクルーディー) がレンダリングされないようにバリアの役割を果たすオブジェクト。
- オクルーディー (Occludee) – オクルーダーによりブロックされ画面にレンダリングされないオブジェクト。

ほとんどのオブジェクトは、カメラの向きとゲーム境界に応じて、オクルーディーにもオクルーダーにもなります。シーン全体を確認して、オクルージョン・カリング計算に含めるオブジェクトを複数選択し、[Occluder Static (オクルーダー・スタティック)] および [Occludee Static (オクルーディー・スタティック)] としてマークすることを推奨します。

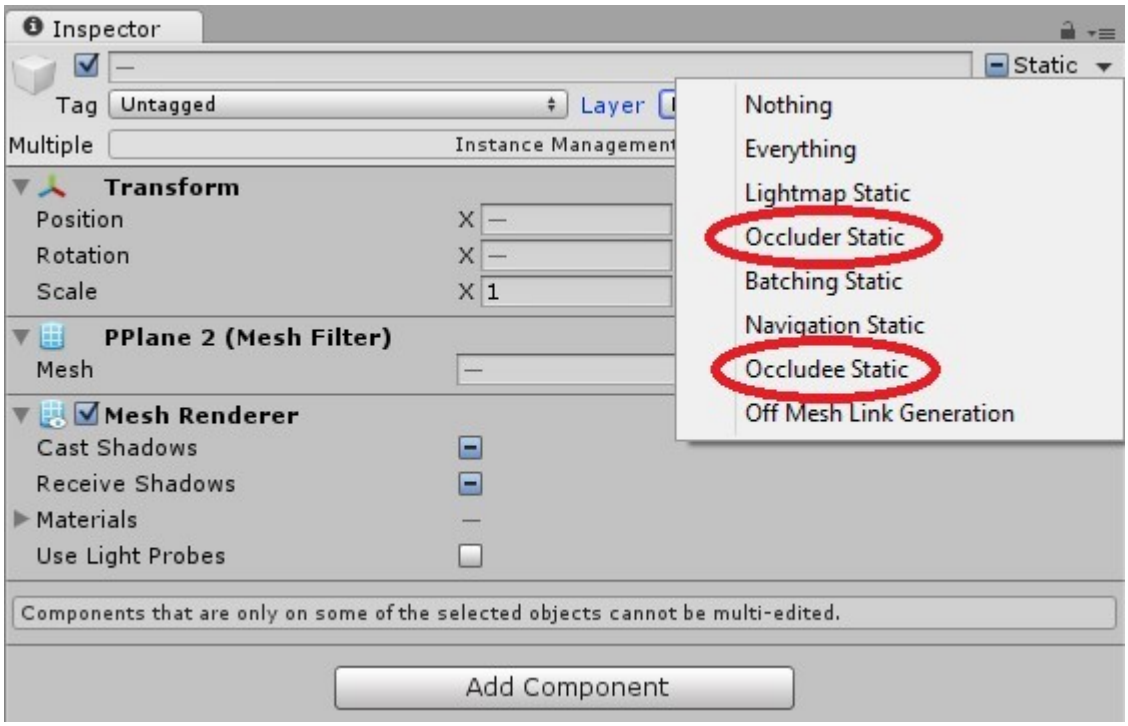

図 25. Inspector でのオクルーダーとオクルーディーの設定方法

オクルージョン・カリングを追加する最後のステップは、シーンのベイク (焼き付け) です。

[Window (ウィンドウ)] > [Occlusion Culling (オクルージョン・カリング)] を選択して、[Occlusion Culling (オクルージョン・カリング)] ウィンドウを開きます。高パフォーマンス/低精度から低パフォーマンス/高精度まで、異なるベイク手法を指定できる、図 26 のようなウィンドウが表示されます。アプリケーションでは「最小有効量」手法を使用すべきです。

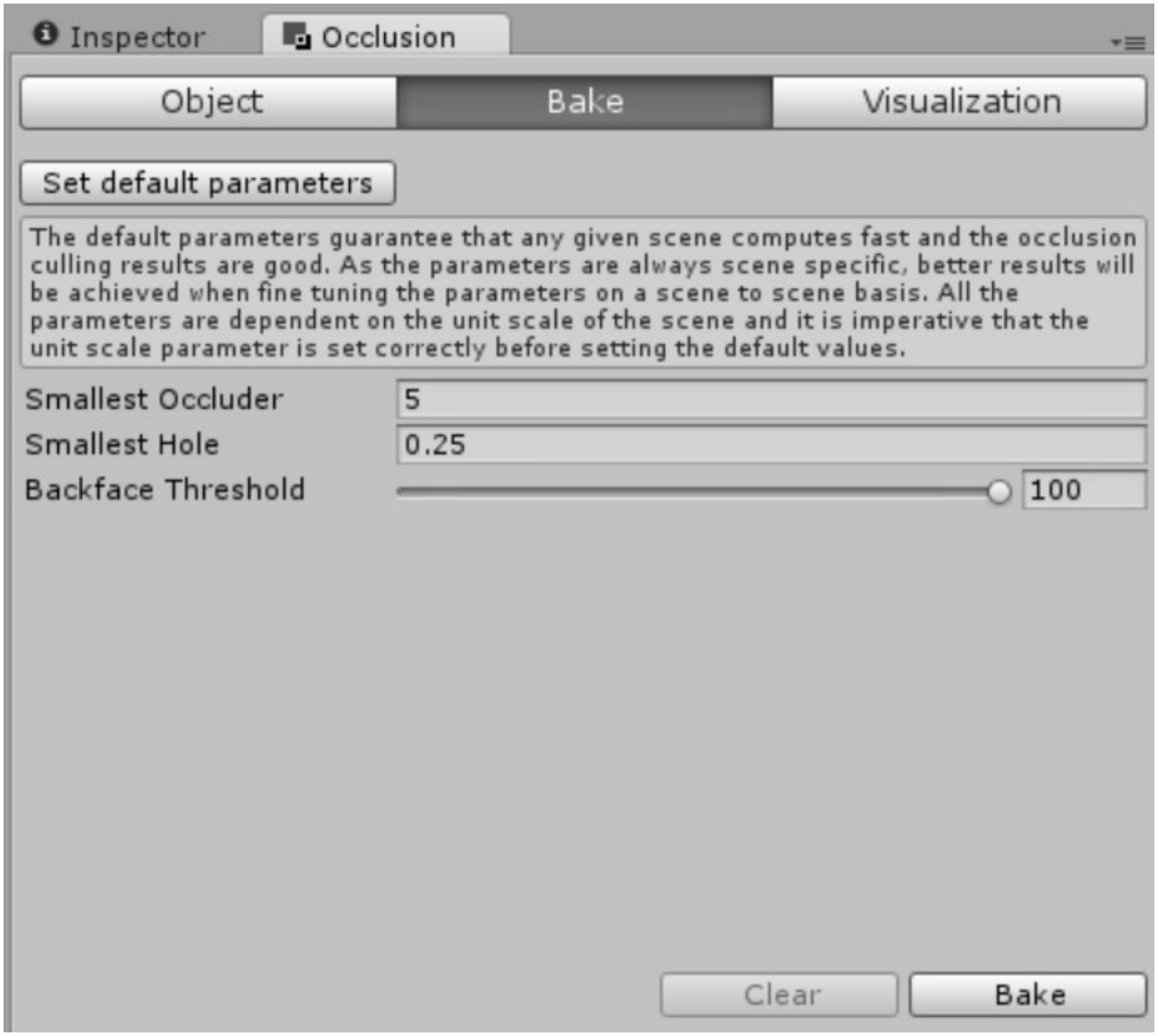

図 26. [Occlusion (オクルージョン)] ウィンドウと [Bake (ベイク)] ボタン

オクルージョン・カリング・システムを設定する場合は、**オクルージョン・エリア**を注意深く設定します。デフォルトでは、Unity* はシーン全体をオクルージョン・エリアとして使用するため、不要な計算が行われます。シーン全体に適用しないようにするには、オクルージョン・エリアを手動で作成し、計算に含めるエリアのみ囲みます。

Unity* は、オクルージョン・カリング・システムの各部分を視覚化できます。カメラのボリューム、可視の線、ポータルを表示するには、[Occlusion Culling (オクルージョン・カリング)] ウィンドウ ([Window (ウィンドウ)] > [Occlusion Culling (オクルージョン・カリング)]) を開いて、[Visualization (視覚化)] タブをクリックします。シーンビューのコンポーネントをすべて視覚化できます。

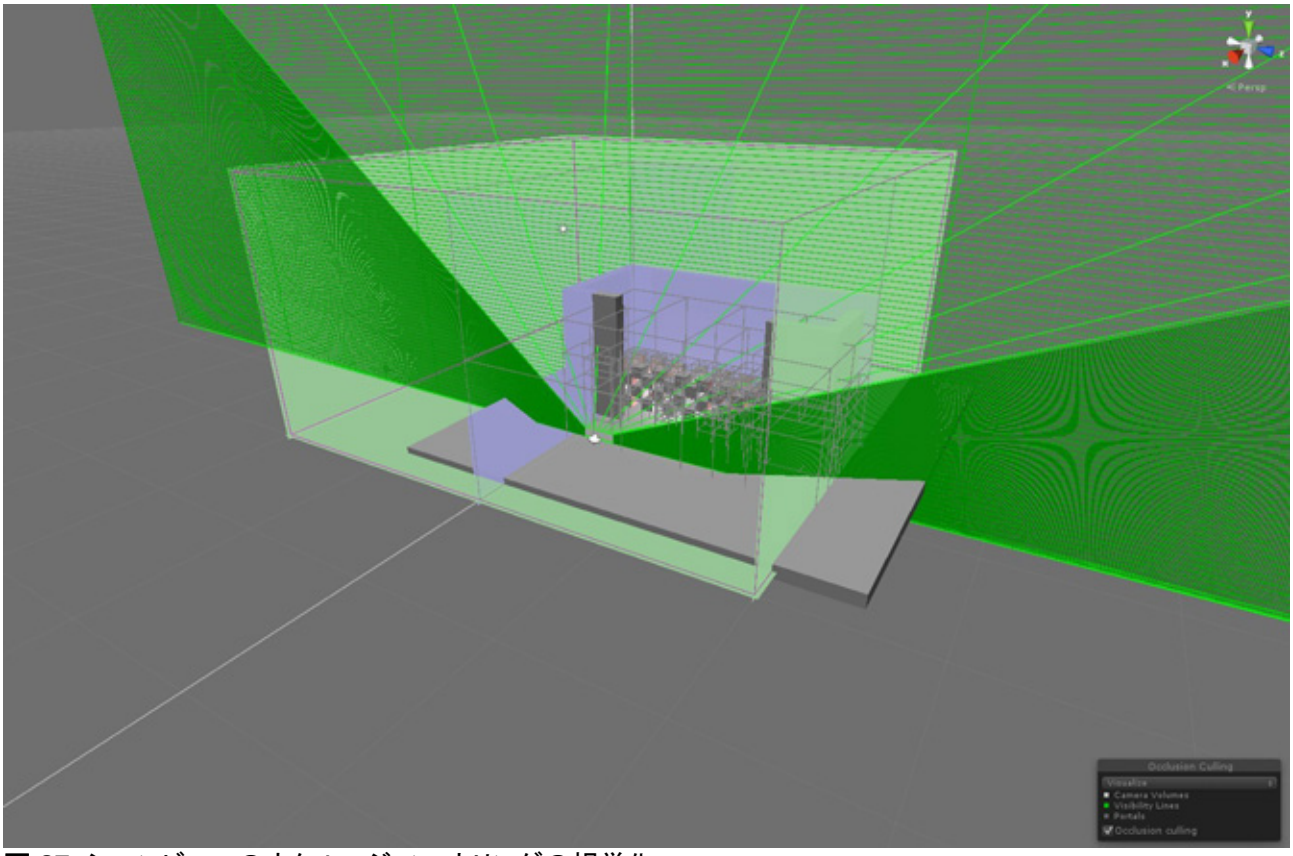

図 27. シーンビューのオクルージョン・カリングの視覚化

詳細については、http://docs.unity3d.com/ja/current/Manual/class-OcclusionArea.html を参照してください。

LOD: 詳細レベル (Level Of Detail)

詳細レベル (LOD) コンポーネントは、オブジェクトのカメラからの距離に依存する詳細レベルを変更して、ゲーム・オブジェクトがメッシュを切り替えられるようにします。この LOD 機能を使用すると、表示の正確さにほとんど影響を与えることなく、フレームのメモリー使用量を大幅に軽減できます。LOD を調整して、より低い詳細レベルでより少ないジオメトリーを提供すると、入力アセンブラーおよび頂点シェーダーの処理が軽減されます。

Unity* Profiler を利用して、LOD 機能が実際に使用されているかどうか確認できます。CPU 使用状況プロファイラーを開いて、[Camera.Render] > [Drawing (描画)] > [Culling (カリング)] を選択し、LOD.ComputeLOD が表示されているか確認します。

図 28. Unity* Profiler で LOD 使用状況を確認する

また、インテル® GPA を使用してフレームキャプチャーを行い、モデルの対応する描画呼び出しを選択した後、[Geometry (ジオメトリー)] タブをクリックして、正しいモデルが使用されていることを確認できます。頂点数のような統計とともに、実際のモデルのジオメトリーが視覚的に表現されます。頂点数がキャプチャーのカメラ距離で使用されるモデルと一致することを確認できます。

詳細レベルは通常、頂点に依存します。頂点あたりの計算量が多すぎるとボトルネックが発生することがあります。モバイルバージョンの Unity* シェーダーを使用すると、頂点あたりの計算量を減らすことができます。オブジェクトが小さいか非常に離れていて、オブジェクトの詳細が必要ない場合は、LOD グループの頂点数を制限します。

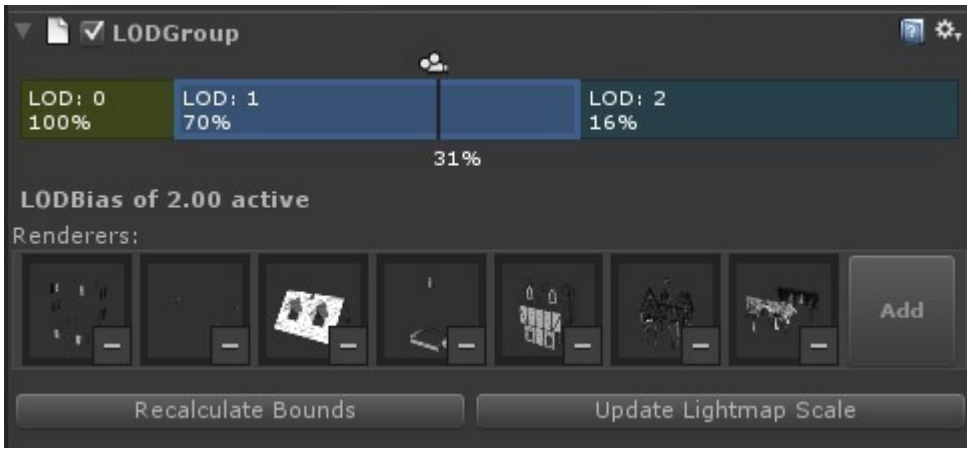

図 29. ゲーム・オブジェクトに LOD コンポーネントを追加するには、[Component (コンポーネント)] > [Rendering (レンダリング)] > [LOD Group (LOD グループ)] をクリックする

図 30. LOD グループでの調整

図 31. 高品質

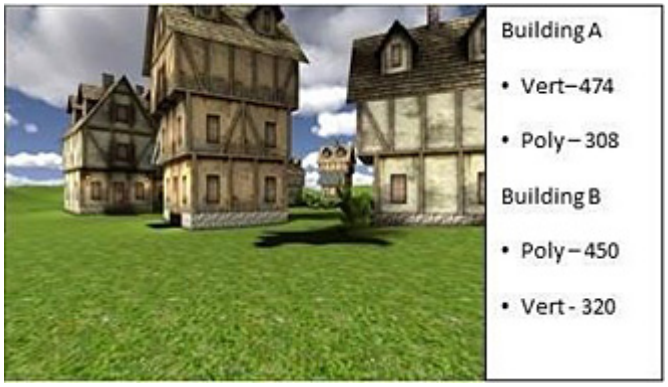

図 32. 低品質

影

影は、非常に多くの GPU パフォーマンスを消費します。影が使用しているシステムリソースの量は、[Profiler (プロファイラー)] > [GPU] > [Shadows (影)] セクションで確認できます。シーンのレイアウトに応じて、影のパフォーマンスを最大化できる、さまざまな最適化があります。例えば、シーンの影の多くがディレクショナル・ライトで生成されている場合、[Project Settings (プロジェクト設定)] > [Quality (品質)] の [Shadow Distance (影の距離)] の値を小さくすると、パフォーマンスが大幅に向上します。影の距離は、フラグメント・シェーダーのパフォーマンスに最も密接に関連しています。インテル® GPA のフレーム・キャプチャーでは、シャドウマップからサンプリングする描画呼び出しを選択すると、フラグメント・シェーダー実行ユニットのストール/アクティブやサンプラーの読み取り/書き込みメトリックが表示されます。影の距離の値はコードで設定することもできます。ポイントライトの場合、影の解像度を調整することで、モバイル上でコストのかかるメモリー帯域幅のボトルネックを軽減できます。

[Project Settings (プロジェクト設定)] > [Quality (品質)] で利用可能な影オプションの概要を次に示します (詳細については、Unity* マニュアルの画質設定を参照してください)。

- **Shadow Filtering (影のフィルター)** – 影をフィルターする方法を指定します。
  - Hard (ハード) - シャドウマップからサンプリングする場合、Unity* は最も近いシャドウマップのピクセルを処理します。
  - Soft (ソフト) – いくつかのシャドウマップのピクセルを平均して、より滑らかな影を作成します。Hard (ハード) よりもコストがかかりますが、より自然な影を作成します。
- **Shadow Resolution (影の解像度)** – シャドウマップの解像度を指定します。
  - 多くの**ポイントライト/スポットライト**を使用する場合、パフォーマンスに大きく影響します。
- **Shadow Projection (影の投影)** – 影の投影方法を指定します。
  - Stable Fit (ステーブルフィット) – カメラが動いても揺れない低解像度の影をレンダリングします。
  - Close Fit (クロースフィット) – カメラが動くと揺れる高解像度の影をレンダリングします。

- **Shadow Cascades (シャドウカスケード)** – シャドウマップのカスケードで使用される並列分割の数 (品質向上のためビューアに近いカスケードは解像度が高くなります)。
    - **ディレクショナル・ライト**のパフォーマンスに大きく影響します。

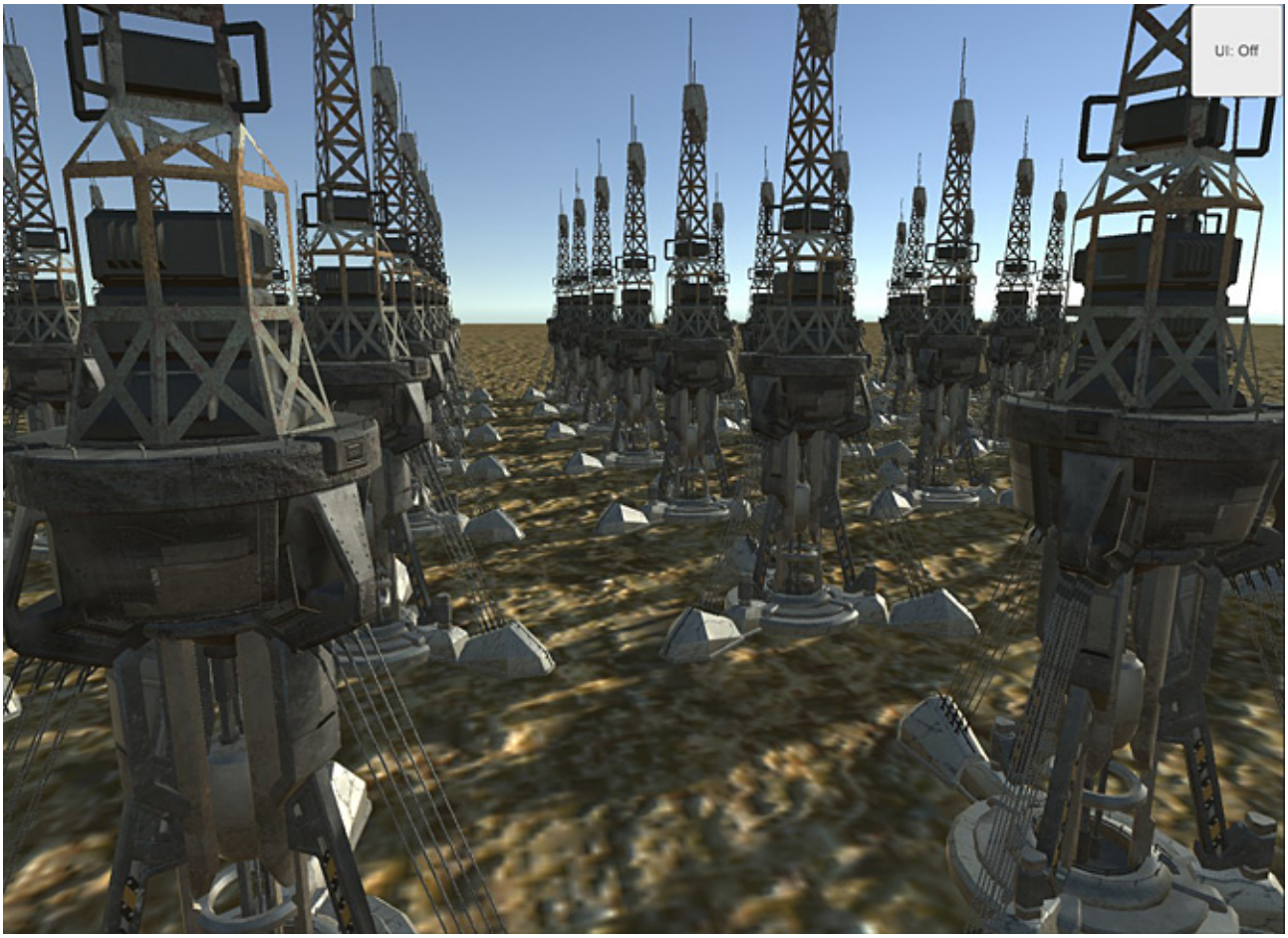

図 33. シャドウカスケードの数を 0 に設定

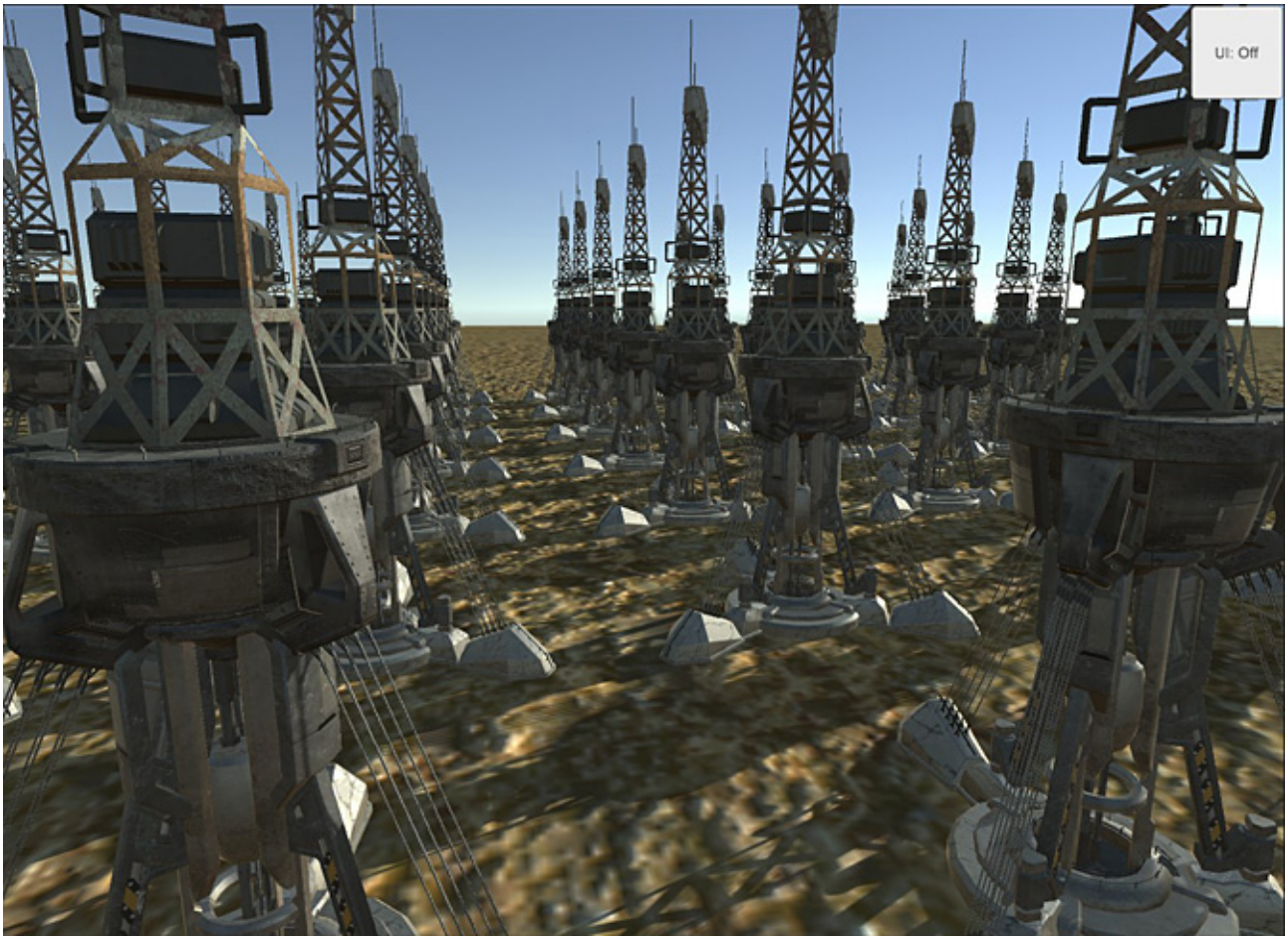

図 34. シャドウカスケードの数を 4 に設定

- **Shadow Distance (影の距離)** – 影を投影できるオブジェクトからの最大距離を指定します。
  - **ディレクショナル・ライト**を使用した場合、フラグメント・シェーダーのパフォーマンスに大きく影響します。
  - スクリプトを使用して変更できます。

パフォーマンスは GPU の使用状況により異なります (シーンおよび影をキャストしている/受け取っているオブジェクト数に依存します)。常に、希望する外観を達成するのに最低限の品質設定を使用することが重要です。一般に、影の距離をデフォルトより低い値に変更することを推奨します。

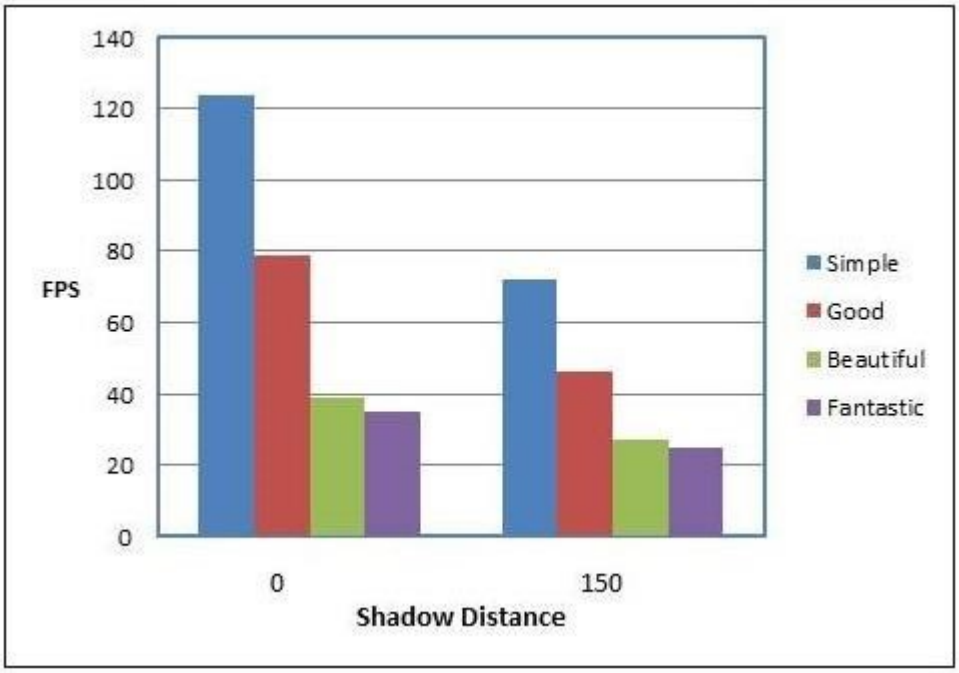

図 35. Unity* Bootcamp デモの影の距離に基づく FPS

シーンのフレームキャプチャーを行った後、インテル® GPA Frame Analyzer を使用することで、Unity* により生成されたシャドウマップを表示できます。最終的なレンダーターゲットに移動し、[Texture (テクスチャー)] タブでシャドウマップを表示します。

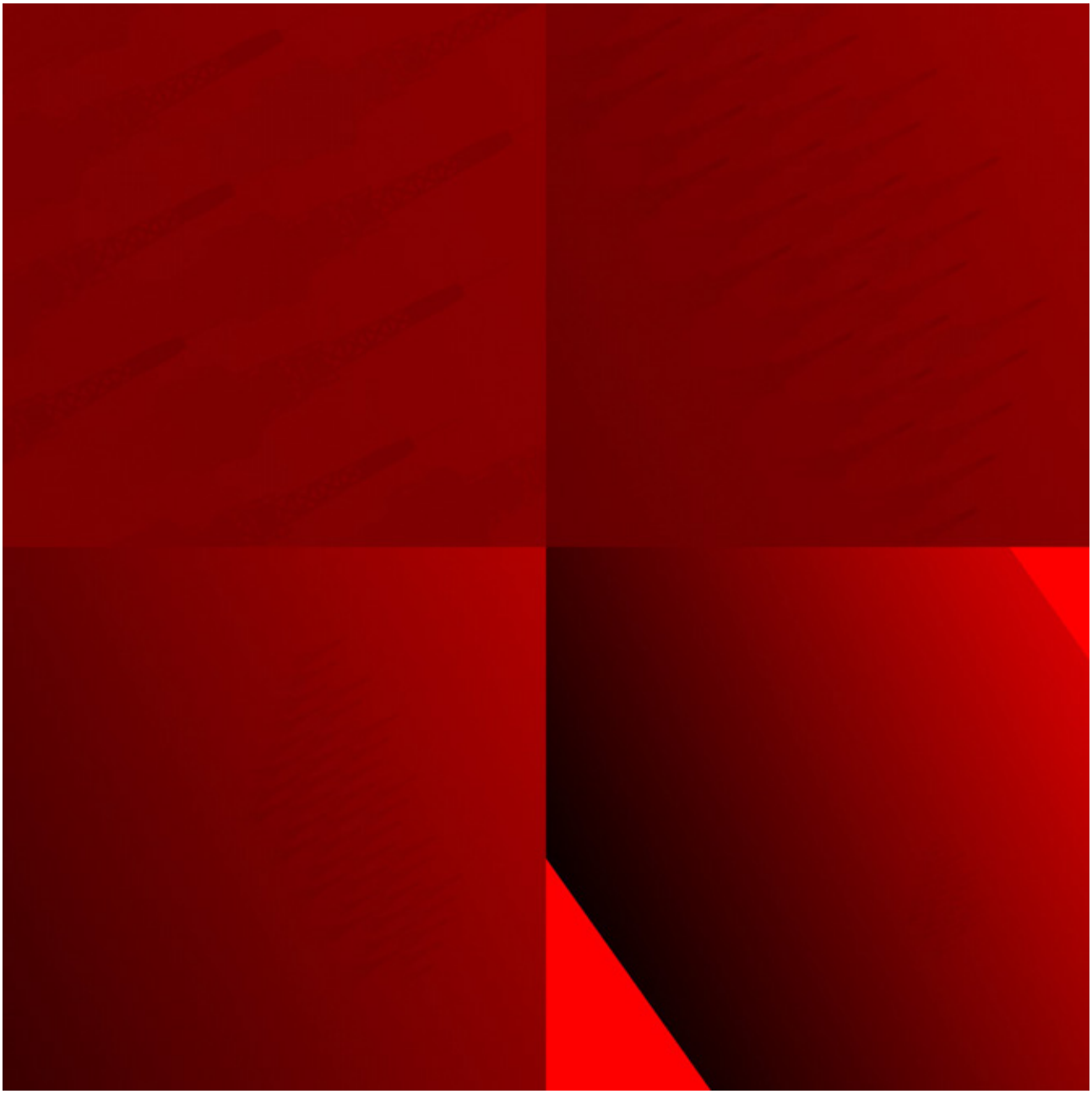

図 36. 生成されたカスケード・シャドウマップをインテル® GPA Frame Analyzer で表示

複数のカメラ

特定の効果を得るため、複数のカメラを使用することがあります。例えば、視差効果を作成する 1 つの方法は、異なるレートで移動する複数のカメラを使用することです。しかし、各カメラは、グラフィックス API のクリア呼び出しの個別のセットと新しいレンダーターゲットに関連付けられるため、注意が必要です。3 台のカメラと UI カメラ (キャンバス・オブジェクトに必要な別のカメラ) を使用する単純なシーンからフレームキャプチャーを行った場合、クリアはシーンの 5.4% を占めます。

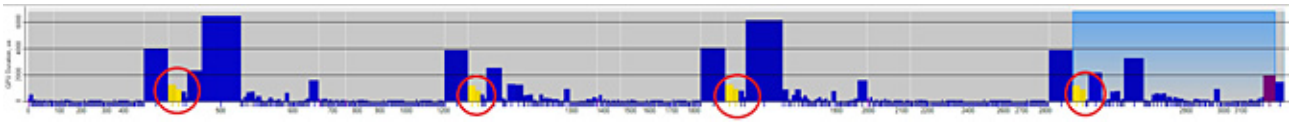

図 37. 合計 4 台のカメラを使用するシーンと関連付けられた描画呼び出しタイムライン。色/深度/ステンシルのクリアは赤い丸で囲まれています。

レンダーキューの順序

グラフィックス・プログラミングでは、不要に何回も重ねてピクセルを描画する (グラフィックス・リソースを浪費する) ことを**オーバードロー (Overdraw)** と呼びます。Unity* は、**[Render Queue (レンダーキュー)]** プロパティーと呼ばれる異なるモデルのレンダリング順序を定義する方法を提供しています。[Render Queue (レンダーキュー)] プロパティーは数値で、[Mesh Renderer (メッシュレンダラー)] の [Material (マテリアル) で設定できます。

レンダーキューの利点を確認するため、多くのオブジェクトがある床を描画してみましょう。まず最初に床をレンダリングします。シーンの半分ですべてのピクセルに触れています。次に、その上にすべてのオブジェクトをレンダリングします。この方法には、多くの不要な処理が含まれています。この例では、オブジェクトが触れているすべてのピクセルが 2 回描画されています。

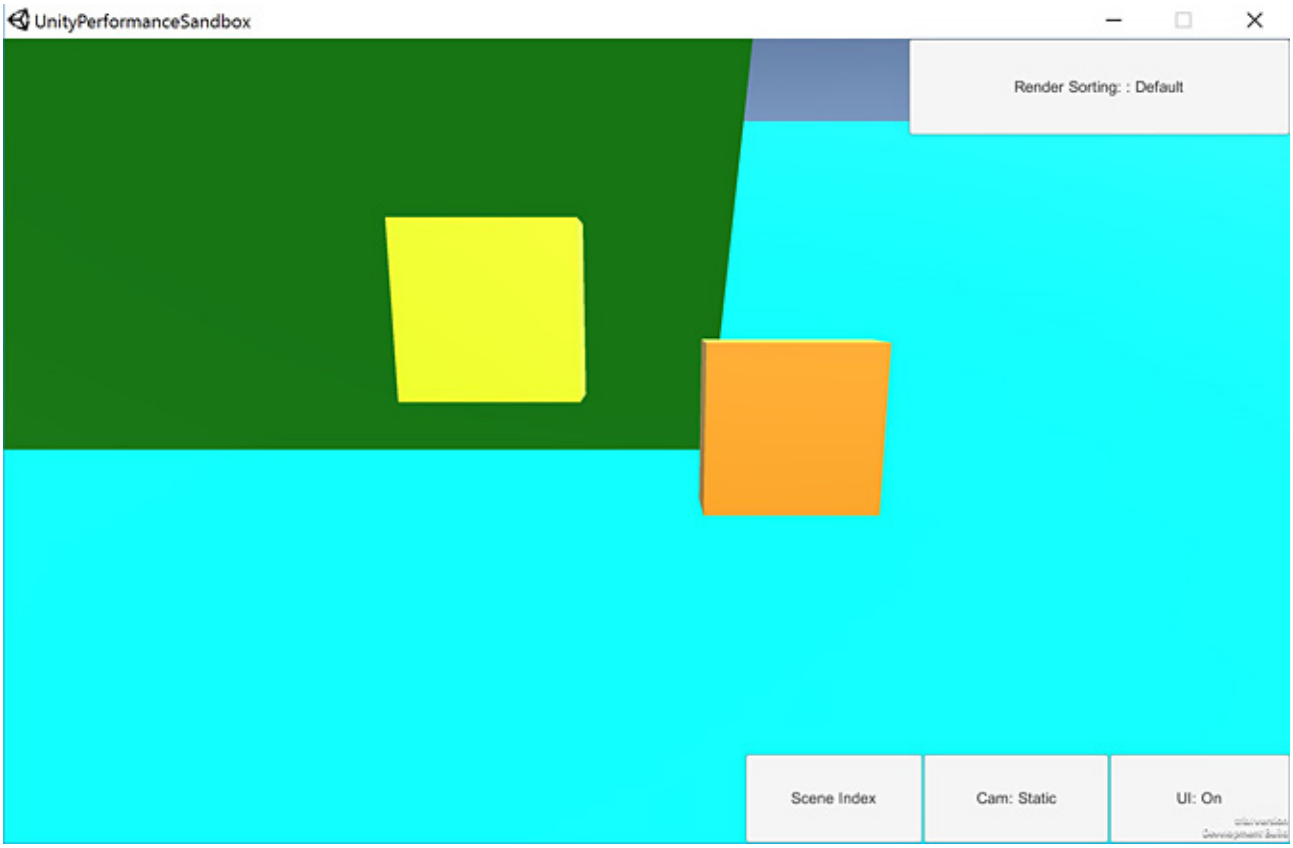

図 38. 切り替え可能なレンダーキュー順序メソッド (デフォルトまたはスマート) を使用した通常のシーン。

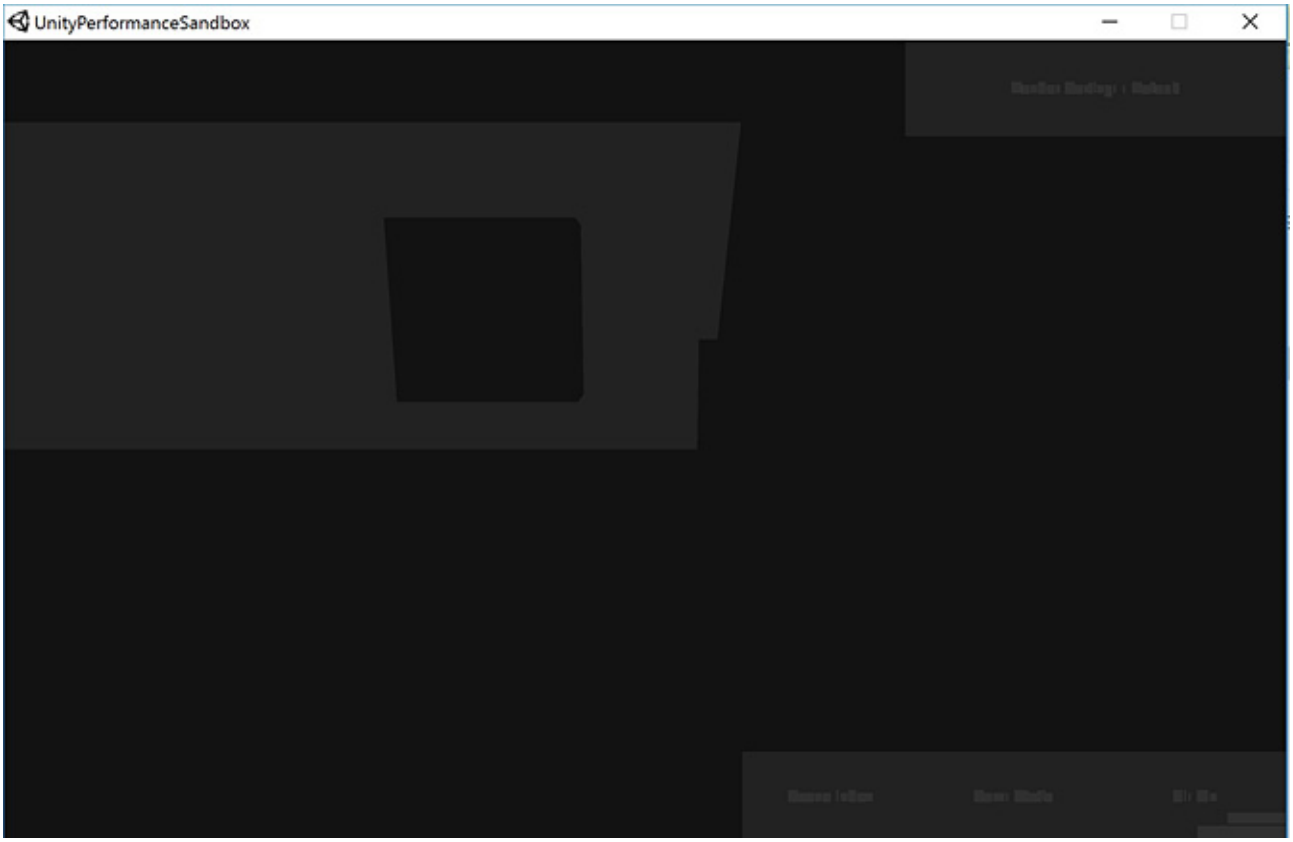

図 39. インテル® GPA System Analyzer を使用してデフォルト・レンダー・キュー順序モードで同じシーンをオーバードローした場合。灰色の領域はオーバードローを示します。上記のスクリーンショットでは、床が、上に浮かぶ緑色のクアッドの前に描画されていることが分かります。

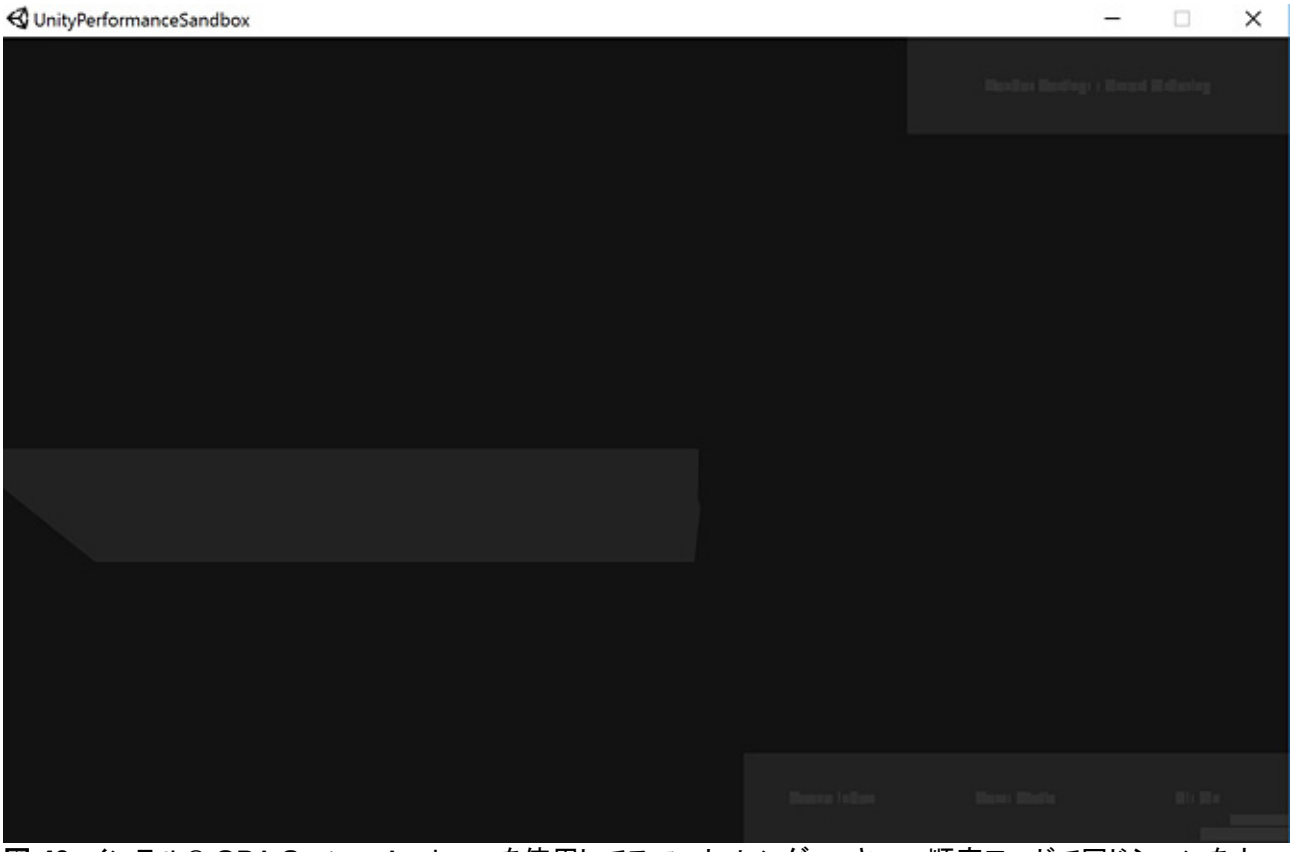

図 40. インテル® GPA System Analyzer を使用してスマート・レンダー・キュー順序モードで同じシーンをオー

バードローした場合。灰色の領域はオーバードローを示します。オーバードローされるセクションは、床クアッドの上に浮かぶ緑色のクアッドのみであることが分かります。問題の描画呼び出しの順序を注意して手動で行うことで、大量のピクセル計算を行わないようにできます。

ライトマップ

ライトマップは、シェーダーのライト値を動的に計算する代わりに、シーンのオブジェクトのシェーダーからサンプリングされるライトマップ (あらかじめ計算されたライトデータを含むテクスチャー) に、すべてのシーンのライトを最初にベイクする手法です。メモリー帯域幅/サンプラー使用率がボトルネックでない場合、この手法を使用するとパフォーマンスが大幅に向上することがあります。Unity* は、この方法でシーンにライトをベイクすることができます。

Unity* は、**ライトプローブ**を使用して、ダイナミック・オブジェクト向けにベイクされたライトデータを生成することもできます。ライトプローブは、周囲のライトとシャドウをサンプリングするシーンに配置可能なポイントです。ライトプローブが指定されているボリュームをダイナミック・オブジェクトが通過すると、それらのライトプローブにより生成されたデータがサンプリングされます。通過するオブジェクトで使用されるライト/シャドウ値は、周囲のプローブ間で補間されます。シーンの周りに配置されるプローブは、3D ボリューム形式にして、ダイナミック・オブジェクトが潜在的にカバーできる領域の周囲により密に分散させます。

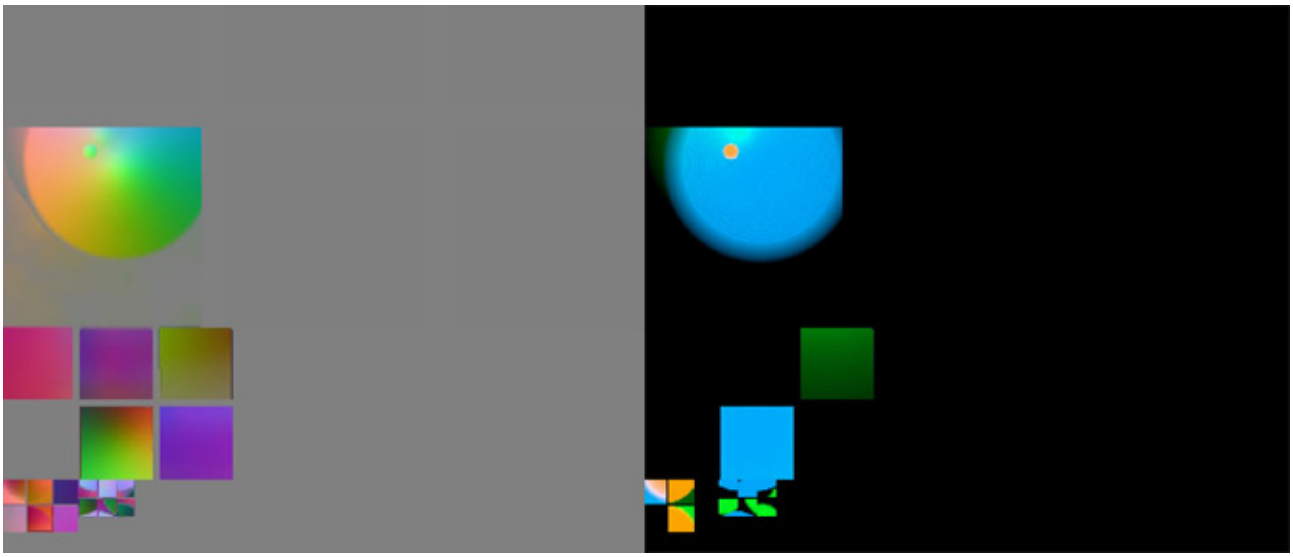

図 41. 左: Unity* で作成したディレクショナル・ライトマップ。右: Unity* で作成した強度ライトマップ。(インテル® GPA Frame Analyzer で表示。)

ライトデータをベイクするには、Inspector ですべてのスタティック・ジオメトリーをスタティックとしてマークし ([オクルージョン・カリング] セクションで説明したチェックボックス)、シーンの周りにライトプローブを配置して、ライトデータを受け取るダイナミック・オブジェクトの潜在的なパスをすべてカバーする 3D ボリュームを形成します。オブジェクトがマークして、ライトプローブを配置したら、[Lightmapping (ライトマップ)] ウィンドウ ([Window (ウィンドウ) > [Lighting (ライト)]) を開いて、[Bake Scene (シーンのベイク)] ボタンをクリックします。

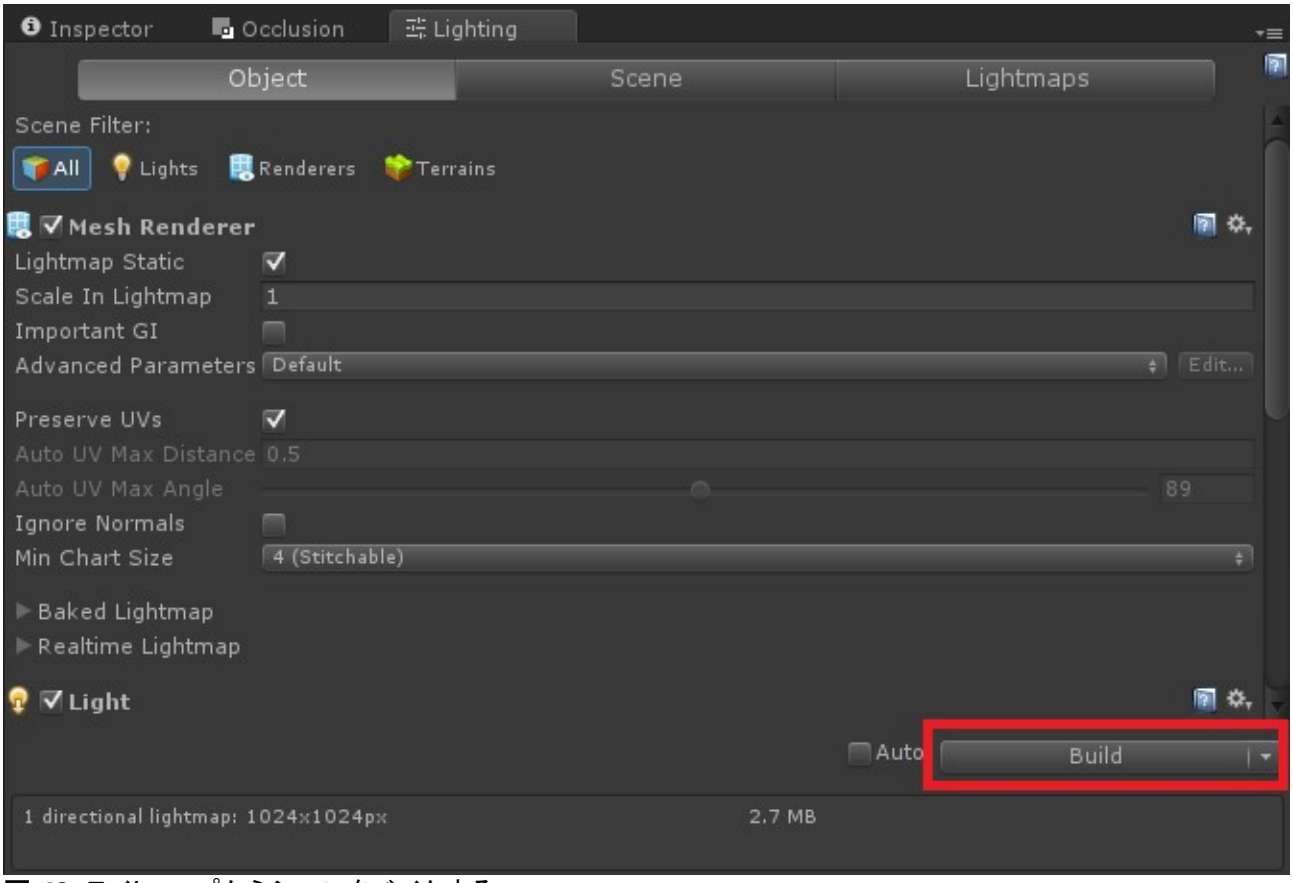

図 42. ライトマップからシーンをベイクする

ウィンドウの右下にロード状況を示す小さなバーが表示されます。ベイクが完了したら、処理は完了です。シーンから不要なダイナミック・ライトをすべて削除/無効化できます。Inspector でライトを「baked (ベイク済み)」としてマークした場合、削除/無効化の必要はありません。ベイクされたライトは自動的に適用されます。シーンのベイク済みライトのみ追跡する簡単な方法は、再ベイクが必要な場合に迅速に有効化/無効化できるように、空のゲーム・オブジェクト以下にすべてのベイク済みライトを保存することです。この方法を使用する場合、ライトマップ・ベイク・ワークフロー・モードが自動に設定されていないことを確認します。

メッシュコライダーの代わりに単純なコライダーを複雑なモデルに使用する

すべてのオブジェクトに**メッシュコライダー**を使用するのではなく、複雑なオブジェクトには**プリミティブ・コライダー**の組み合わせを使用することが重要です。プリミティブ・コライダーは単純な 3D 形状 (カプセル、球体、箱など) で、メッシュコライダーは衝突判定が可能なメッシュ形状です。可能な場合は、メッシュコライダーではなくプリミティブ・コライダーを選択します。

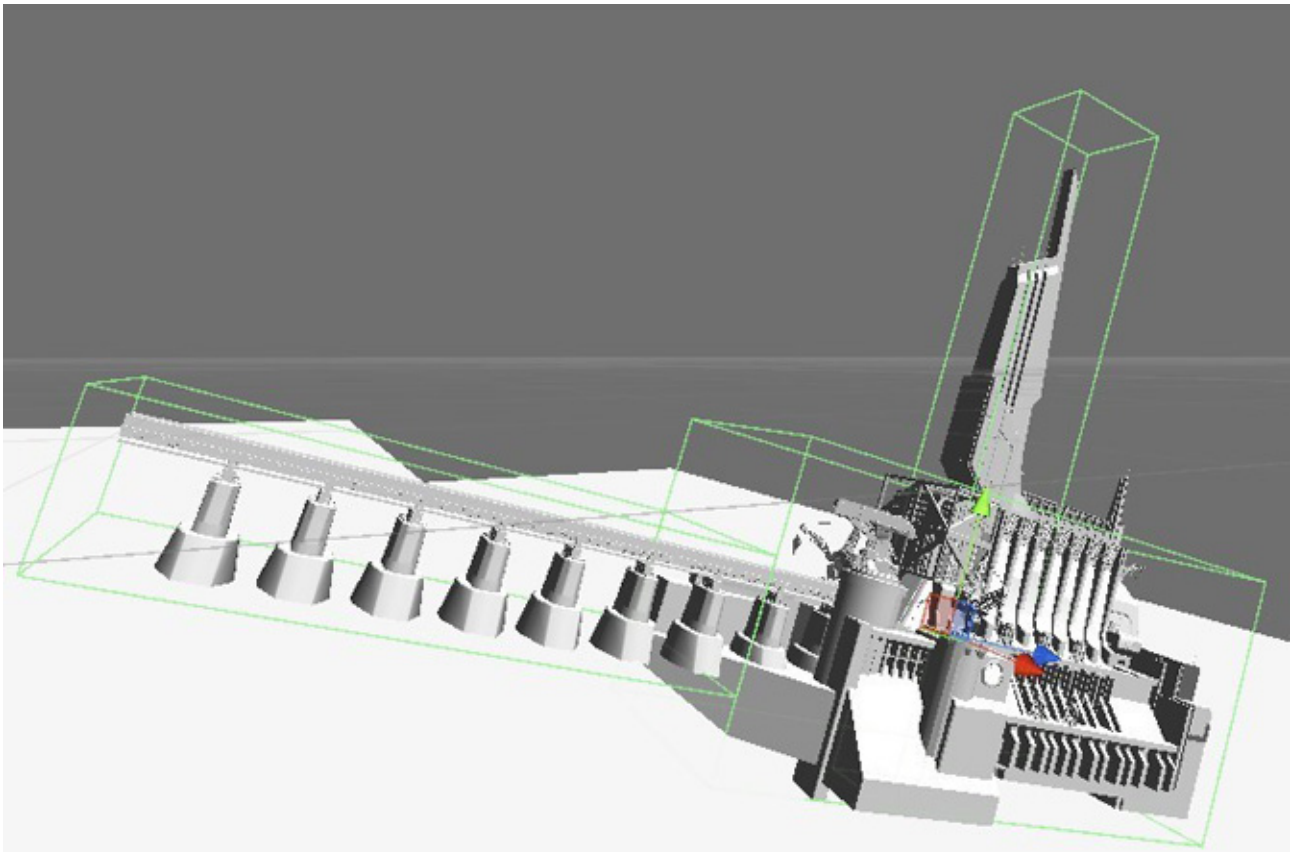

図 43. この大きな構造物には 3 つのプリミティブ・コライダーを使用している。メッシュコライダーを使用した場合よりもパフォーマンスが大幅に向上します。

チュートリアルのパート 4 に続く:
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 4 (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。