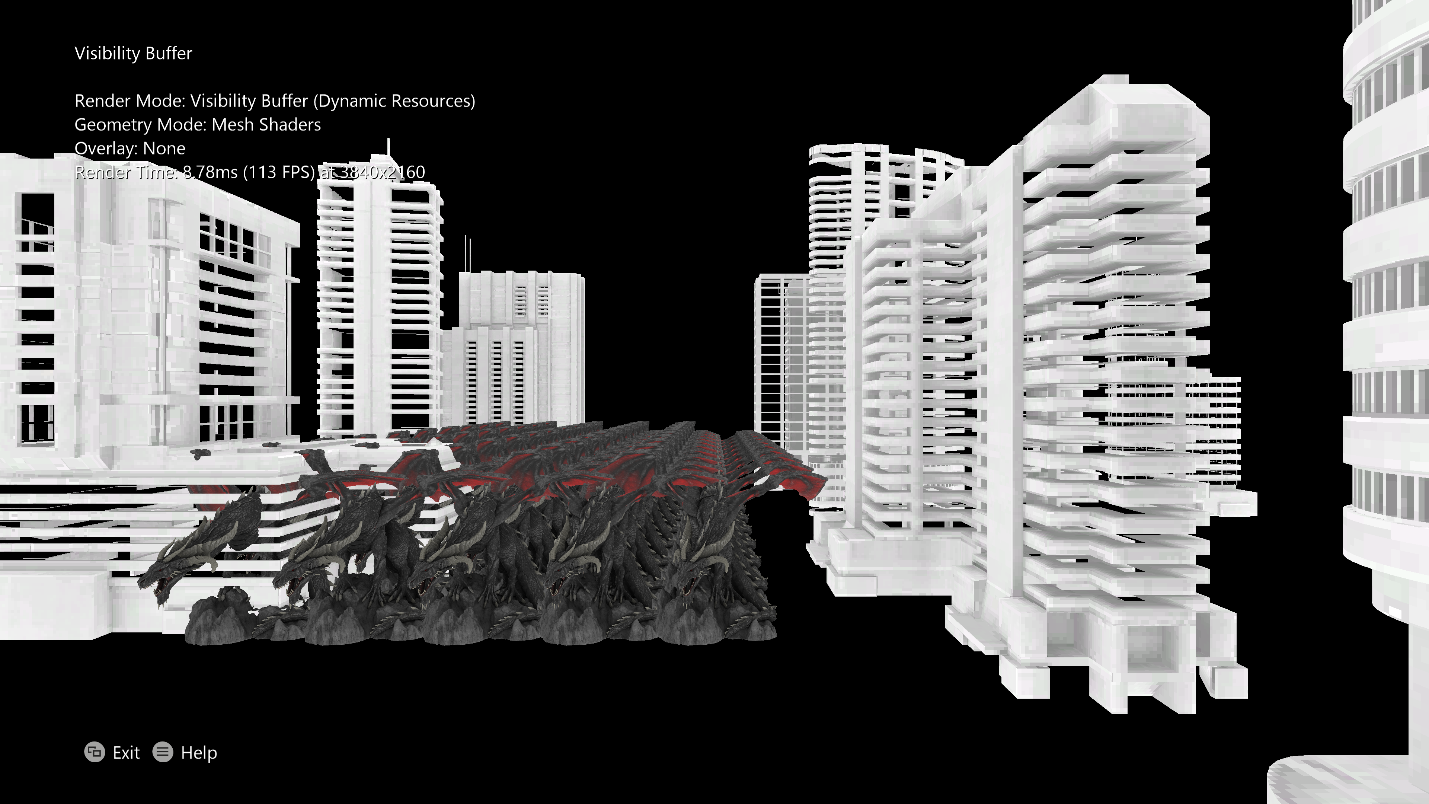
可視性バッファーのサンプル

*このサンプルは Microsoft Game Development Kit (2021 年 4 月) と互換性があります*

# 説明

このサンプルでは、メッシュ シェーダーと HLSL 6.6 動的リソースを使用した、可視性バッファー (遅延) レンダリングの手法を示します。



# サンプルのビルド

Xbox One 開発キットを使用している場合は、アクティブ ソリューション プラットフォームを Gaming.Xbox.XboxOne.x64 に設定します。

Xbox Series X|S を使用している場合は、アクティブ ソリューション プラットフォームを Gaming.Xbox.Scarlett.x64 に設定します。

PC 用のビルド (Gaming.Desktop.x64) では、HLSL SM 6.6 機能が使用されているため、[DirectX Agility SDK](https://devblogs.microsoft.com/directx/gettingstarted-dx12agility/) が必要となります。Agility SDK は、サンプル内には NuGet パッケージとして含まれています。また、Microsoft.Windows.SDK.CPP NuGet パッケージを使用して、DXC.exe コンパイラの最新の Windows SDK (22000) バージョンを取得します。開発者は、最新の DXC を直接 [Github](https://github.com/microsoft/DirectXShaderCompiler/releases) から使用することも可能です。

*詳細については、GDK のドキュメントの*「サンプルの実行」*を参照してください。*

# サンプルの使用方法

このサンプルの特色は、飛行/一人称カメラで、シーン内でどこでも配置できます。コントロールに関する全体的な説明を次に一覧で示します。

# コントロール

|  |  |
| --- | --- |
| 操作 | ゲームパッド |
| ビュー コントロール | [スタート] ボタン |
| Visibility Buffer (遅延) Rendering と Forward Rendering を切り替えます。Visibility Buffer Rendering では、動的リソースを使用します。フォワードでは使用しません。 | A |
| Visibility Buffer オーバーレイ。プリミティブ ID またはオブジェクト ID がオーバーレイとして表示されます。内部で、両方とも単一の 32 ビット UINT レンダー ターゲットにレンダリングされます。 | B |
| 頂点シェーダー/メッシュ シェーダー ジオメトリのレンダリングを切り替えます。Visibility Buffer モードにのみ適用されます。Xbox One では Vertex Shader のみがサポートされます。 | X |
| カメラの前方 / 後方 / ストラフ | 左スティック |
| カメラのヨー / ピッチ | 右スティック |
| カメラの上 / 下 | 左/右トリガー |
| 高速飛行モード (ターボ) | 左スティックをクリックして長押し |
| 終了 | ビュー ボタン |

# 実装上の注意

2 つのレンダー パスが表示されます (A キーを押して切り替えます)。

Forward パスでは、一般的にラスター化を実行し、すべてのリソース (インデックス バッファー、頂点バッファー、テクスチャ、サンプラーなど) をルート署名を介してバインドするか、または入力アセンブラーにバインドします。

Visibility Buffer パスでは、 ジオメトリ シェーディング用、および最終的なシーン シェーディングを実行するCompute Shader 用の Mesh Shader または Vertex Shader のオプションを使用して「遅延」ラスター化を実行します。メッシュ シェーダーを使用する場合、すべてのリソースは HLSL 6.6 の ResourceDescriptorHeap[] と SamplerDescriptorHeap[] を介して “バインドレス” でアクセスされます。Vertex Shader モード中は、パフォーマンス上の理由から、インデックス バッファーおよび頂点バッファーを入力アセンブラーにバインドします。

Vertex Shader モードでは、インデックス バッファーのみをバインドし、ResourceDescriptorHeap[] を介して頂点バッファーにアクセスすることも可能ですが、これにより一部のハードウェアで大幅にパフォーマンスを低下させました。

Mesh Shader は Xbox One でサポートされていないため、Vertex Shader モードのみが使用可能です。

Visibility Buffer

可視性バッファー パスは、“遅延” レンダラーの実装です。ただし、初期のラスター化パスでは、最初の 12 ビットのオブジェクト識別子 (4,096 個の実行可能なオブジェクト) および最後の 20 ビットのプリミティブ識別子 (オブジェクトあたり 1,048,576 個の三角形) で構成される単一の 32 ビット UINT レンダー ターゲット (Visibility Buffer) のみが出力されます。次に、コンピュート シェーダー パスは Visibility Buffer を使い、ResourceDescriptorHeap からオブジェクトごとの定数情報をロードしてから、ResourceDescriptorHeap からインデックス バッファー/頂点バッファー/テクスチャをロードします。コンピュート シェーダーはさらに、各三角形のヒット ポイントを再構築し、適切な補間テクスチャ座標と画面空間の導関数を決定します。これは、このデータが最初のパスの後の中間レンダー ターゲットに格納される標準的な遅延レンダラーと異なることに注意してください。最後に、SamplerDescriptorHeap を使用して SamplerState オブジェクトが読み込まれ、テクスチャがサンプリングされて画面に出力されます。

ジオメトリ シェーディング

このサンプルでは、頂点シェーダーとメッシュ シェーダーの両方を実装します (サポートされている場合)。可視性バッファーにはプリミティブ ID を含める必要があります。プリミティブ ID は、頂点シェーダーのパス後にSV\_PrimitiveID システム セマンティックから取得することも、メッシュ シェーダーからプリミティブ出力として渡すこともできます。

メッシュ シェーダーの場合、単純な起動時メッシュレット分割アルゴリズムが実行されます。このアルゴリズムでは、事前に最適化されたインデックス バッファー (-op フラグを使用した DirectXMesh の MeshConvert の出力など) が必要です。次に、既存のプリミティブ順序を維持しながら、インデックス バッファーを最大頂点 / 最大プリミティブ制限を満たすメッシュレットに分割します。次に、極めて単純なメッシュ シェーダーでは、ResourceDescriptorHeap[] を介してメッシュレット情報をロードし、標準的な頂点シェーディングを実行し、プリミティブごとのエクスポートで primitiveID を出力できます。この ID は元のインデックス バッファーと同じ順序を維持するため、コンピュート シェーダーがメッシュレット形式を理解する必要なく、同じコンピュート シェーダーに使用できます。

パフォーマンス

この手法のパフォーマンスは、多くの要因によって異なります。メッシュ シェーダーと頂点シェーダー間のジオメトリ シェーディングのコストとトレードオフは、ターゲット GPU プラットフォームとアセット サイズによって異なります。

さらに、フォワード レンダラーのピクセル シェーダーの作業は頂点シェーダーの作業と並行して実行できますが、可視性バッファー レンダラーでは、すべての頂点が処理された後、コンピューティングで “ピクセル シェーディング” を実行する必要があります。つまり、ピクセルの作業が頂点の作業を上回らない限り、オーバー ドローのコストは、より大きな頂点シェーダーの作業によって隠されることを意味します。

ただし、コンピュート シェーダーは非常に効率的であり、ピクセル シェーダーの作業が優先する場合、Visibility Buffer Renderer のパフォーマンスが向上する可能性があります。これは主に、オーバー ドローが完全に無く、メッシュ シェーダーのわずかなパフォーマンスの向上によるためですが、クワッド ヘルパー レーンや、ピクセル シェーダー ウェーブを遅くする可能性があるプリミティブ出力の風変わりな点を回避する利点もあります。

このサンプルでは、Forward Renderer と Visibility Buffer Renderer の両方が、シェーディングの最後にループ内で多くの冗長テクスチャ フェッチを実行します。これは、ピクセル シェーダーのコストを増やして、オーバー ドローのパフォーマンスの違いをより適切に反映させようとしています。これを行わないと、頂点シェーダーの作業は常に優先されます。

最後に、このサンプルに従来の Deferred Renderer は実装されていませんが、この方法にはそれ以上の利点が存在する可能性があります。重要な点は、レンダー ターゲットのメモリ要件がはるかに低い点です。可視性バッファーは、拡散 / 法線 / 反射などの複数のターゲットではなく、単一の 32 ビット ターゲットのみです。同様に、初期のパスで 1 つのターゲットのみが書き込まれ、コンピュート シェーダーから出力される UAV が 1 つだけであるため、必要なフィル レートは、はるかに少なくなります。必要なリソースは、必要に応じて記述子ヒープから直接プルされます。

# プライバシー ステートメント

サンプルをコンパイルして実行する場合、サンプルの使用状況を追跡するために、サンプル実行ファイルのファイル名が Microsoft に送信されます。このデータ コレクションからオプトアウトするには、Main.cpp の「Sample Usage Telemetry」というラベルの付いたコードのブロックを削除します。

全般的な Microsoft のプライバシー ポリシーの詳細については、「[Microsoft プライバシー ステートメント](https://privacy.microsoft.com/en-us/privacystatement/)」を参照してください。