

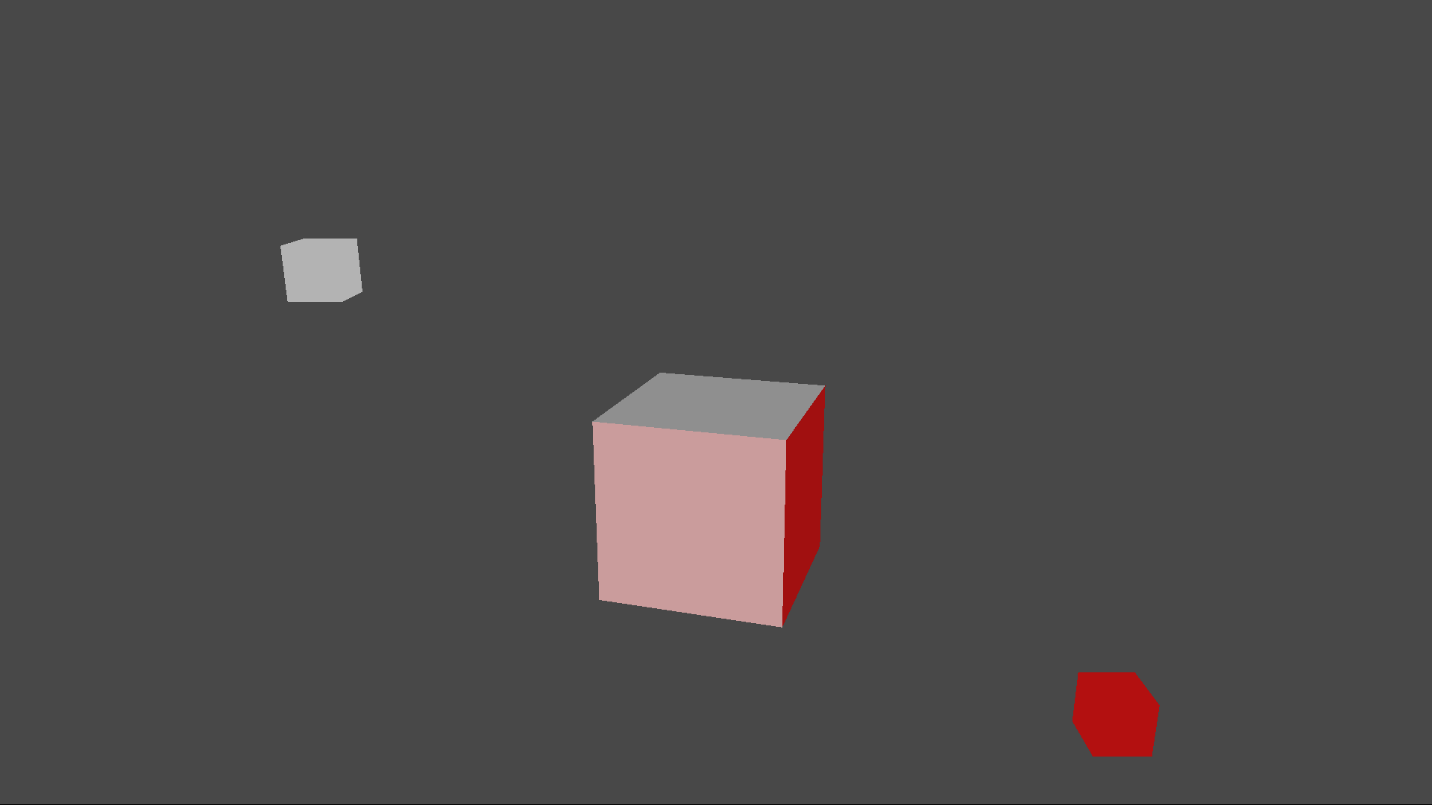
単純な照明のサンプル

*このサンプルは、Microsoft ゲーム開発キットのプレビュー (2019 年 11 月) に対応しています。*

# 説明

このサンプルでは、静的な Direct3D 12 の頂点、インデックス、および定数バッファーを作成して、静的および動的なランバート照明を使用して照らされるインデックス付きのジオメトリを描画する方法を示します。

このサンプルでは、2 つの明かり (白色と赤色) で照らされた大きい立方体をレンダリングします。明かりも立方体として表されます。白色の明かりは固定されており、赤色の明かりは中央の立方体の周りを周回します。中央の立方体も回転します。この動きによって、さまざまな角度から、この色付きの明かりの効果を確認できます。



# サンプルのビルド

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

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

*詳細については、GDK 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

|  |  |
| --- | --- |
| 操作 | コントローラー |
| 終了する | ビュー ボタン |

# 実装に関する注意事項

## シェーダー

## このサンプルでは、3 つのシェーダー (1 つの頂点シェーダー ("TriangleVS") と 2 つのピクセル シェーダー ("LambertPS"、"SolidColorPS")) を使用してシーンをレンダリングします。 コンパイルされたシェーダー blob が CreateDeviceDependentResources に読み込まれ、シェーダーの組み合わせごとにパイプライン状態オブジェクトを作成するときに参照されます。 すべてのシェーダーは同じ HLSL インクルード ファイル ("SimpleLighting.hlsli") で定義されています。3 つのスタブ シェーダーにこのファイルが含まれます。3 つのシェーダー blob を作成するために、異なるエントリ ポイントで各スタブ シェーダーがコンパイルされます。

## パイプライン状態オブジェクト (PSO)

この単純な照明のサンプルには、2 つの固有のシェーダーの組み合わせがあります。1 つは TriangleVS と LambertPS の組み合わせであり、もう 1 つは TriangleVS と SolidColorPS の組み合わせです。DirectX 12 では、固有のシェーダーの組み合わせごとにパイプライン状態オブジェクト (PSO) を作成する必要があります。名前から分かるように、PSO は、特定のシェーダー セットを使用した複数の描画呼び出しに必要なすべてのパイプライン状態をカプセル化します。PSO によって、ルート署名、パイプラインの各ステージのシェーダー、ラスタライザー状態、深度ステンシル状態、ブレンド状態などの状態設定が結合されます (詳細については、MSDN のドキュメントを参照してください)。

## ルート署名

ルート署名によって、グラフィックス パイプラインにバインドされるリソースの種類と、リソースのレイアウト方法が定義されます。ルート署名は API 関数署名に似ていますが、ルート署名では、パラメーターの種類、パラメーターの順序、およびレイアウトが示される一方で、実際のパラメーター インスタンスは定義されません。ルート パラメーターは、ルート署名の要素に対応する実際のデータ インスタンスです。このサンプルの頂点シェーダーには、シェーダー定数用の 1 つの構造しか必要ないため、ルート署名は非常に単純です。ルート署名には、ConstantBufferView 型の単一のルート パラメーターが含まれます。

## ジオメトリ

## シーンのジオメトリは、立方体の 6 つの正方形を表す 24 個の頂点のデータを含む静的頂点配列とインデックス配列で構成されます。2 つの配列は Sample::CreateDeviceDependentResources の内部で宣言されます。ここで、これらの 2 つの配列は、バッファー用の ID3D12Resources を作成するために ID3D12Device::CreateCommittedResource によってただちに使用されます。簡略化のために、このサンプルでは D3D12\_HEAP\_TYPE\_UPLOAD を使用します。これにより、リソースの作成と、データによる各リソースの初期化を 1 つのステップで行うことができるためです。ただし、\_UPLOAD ヒープは、ジオメトリ データにとって次善の場所です。より効率的な実装では、ジオメトリ データ用に D3D12\_HEAP\_TYPE\_DEFAULT を使用します。\_DEFAULT ヒープを初期化するために、\_UPLOAD ヒープを使用する必要があります。このため、2 つのヒープを使用することになり、実装が複雑になります。

ジオメトリ用のバッファーが作成された後、このサンプルでは、頂点用の D3D12\_VERTEX\_BUFFER\_VIEW とインデックス用の D3D12\_INDEX\_BUFFER\_VIEW を作成できます。これらのビューは、ID3D12GraphicsCommandList::IASetVertextBuffer と ID3D12GraphicsCommandList::IASetVertextBuffer の呼び出しを介して入力アセンブラーを設定するときに Sample::Render で使用されます。

## シェーダー定数の管理

この非常に単純なシーンでは、すべてのシェーダー定数が 1 つの定数バッファーにまとめられます。これには、以下が含まれます。

* ワールド、ビュー、および投影マトリックス
* 明かりの方向と色
* 単色

通常、シーンがより複雑な場合、定数が更新される頻度に応じて、定数を複数のバッファーに分割します。

大きい立方体と赤色の明かりはアニメーション化されるため、シェーダー定数を描画呼び出し間でフレームごとに複数回更新する必要があります。ルート署名に ConstantBufferView 型の 1 つのルート パラメーターが含まれることに注意してください。このパラメーターで、各描画呼び出しで使用するシェーダー定数のコピーを参照する必要があります。CPU と GPU は並列で動作するため、GPU が定数バッファーの使用を終了するまで、CPU はその定数バッファーを更新しないようにする必要があります。定数バッファーが 1 つしかない場合、GPU が描画を終了するまで CPU をブロックする必要がありますが、複数の描画呼び出しに対して定数を更新する必要があるため、これを実現するのは簡単ではありません。したがって、このサンプルでは複数の定数バッファーを使用しています。これにより、GPU が描画を行っている最中に CPU は GPU に定数を送り続けることができます。

このサンプルは単純で、フレームごとに固定数の描画呼び出しがあります。これはコンパイル時に認識されます。このサンプルでは、各描画呼び出しに、スワップ チェーンのバック バッファーの数を乗算した数のバッファーが作成されます。この数によって、CPU が書き込む空きバッファーが常に存在します。すべての定数バッファーが、Sample::CreateDeviceDependentResources で作成される 1 つの連続アップロード バッファーに保存されます。このアップロード バッファーは、CPU アドレス空間と GPU アドレス空間の両方のベース メモリ アドレスを取得するために、ただちにマッピングされます。

Sample::Render メソッドでは、定数アップロード ヒープの CPU ベース アドレスからインデックスが付けられた場所に定数が書き込まれます。ID3D12GraphicsCommandList::SetGraphicsRootConstantBufferView の呼び出しを通じて、同じインデックスが GPU アドレスと結合され、バッファーがパイプラインにバインドされます。インデックスでは、描画カウントとともにバック バッファーの一連の流れを考慮する必要があります。詳細については、Sample::Render の実装を参照してください。

## CPU/GPU の同期

GPU がコマンド リストを処理するよりも早く CPU がそれらのコマンド リストを GPU に発行できる場合、CPU は GPU が追いつくまで待たなければなりません。このサンプルでは、ラウンドロビン方式で定数バッファ メモリを使用しています。したがって、一定期間の後にバッファー スロットが再利用されます。通常、共有リソースを再利用する前に、リソースが使用中でないことを確認するために何らかの同期方法を使用することが重要です。 このサンプルでは ID3D12Fence を使用して CPU と GPU を同期しています。CPU は、コマンドをコマンド キューに挿入して、フレーム インデックス値を含むフェンス オブジェクトを "シグナル" として送ります。GPU でシグナル コマンドが処理されると、CPU は、ただちに指定のフレーム インデックス値を認識できるようになります。これにより、CPU は、現在のフレーム インデックスと、GPU がシグナルで通知した最後のフレーム インデックスを比較でき、CPU と比べて GPU がどの程度遅れているか特定できます。GPU のフレーム数と CPU のフレーム数の差がバック バッファー数を超えた場合、CPU は待機する必要があります。

# プライバシーに関する声明

サンプルをコンパイルして実行すると、サンプルの使用状況を追跡するため、サンプルの実行可能ファイルのファイル名が Microsoft に送信されます。このデータ収集を無効にするには、「Sample Usage Telemetry」とラベル付けされた Main.cpp 内のコードのブロックを削除します。

Microsoft のプライバシーに関する声明の詳細については、「[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。