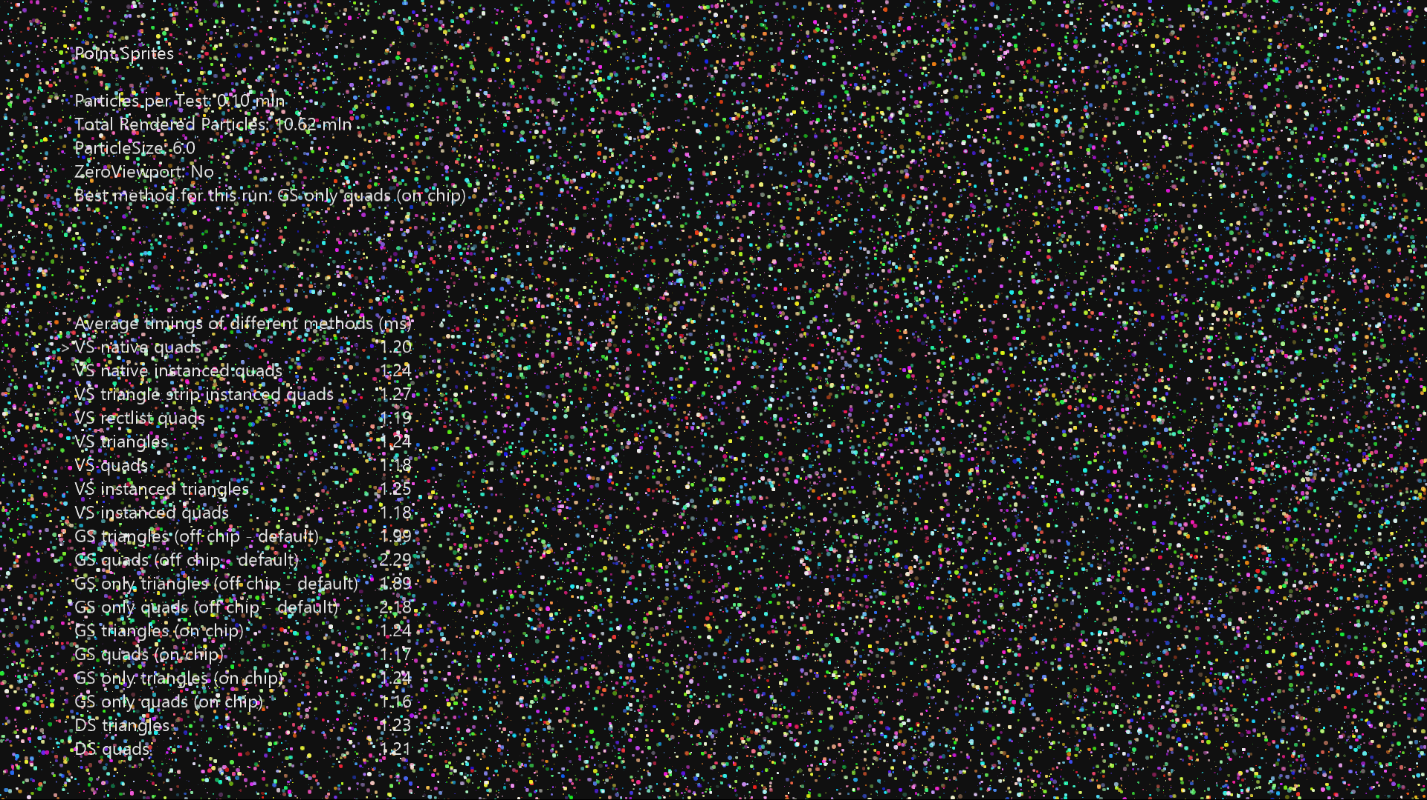


ポイント スプライト

*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、DirectX 12 でポイント スプライトをレンダリングする 10 とおりの方法を示します。



# 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。

*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

このサンプルでは、次のコントロールが使用されます。

|  |  |
| --- | --- |
| 操作 | コントローラー |
| サンプルを終了する。 | ビュー ボタン |
| スプライトのサイズを大きくする | A ボタン |
| スプライトのサイズを小さくする | B ボタン |
| ヌル ビューポートを切り替える | Y ボタン |
| 選択したテストを変更する | 方向パッド 上/下 |
| 強調表示されたテストを切り替える | X ボタン |

# 実装に関する注意事項

グラフィック パイプラインの進化に合わせて、ジオメトリ シェーダーのような非常に柔軟でカスタマイズ可能なステージの使用が優先されたため、ポイント スプライトのファーストクラスのサポートが削除されました。実際には、ポイント スプライトは、GPU のジオメトリ関連のすべてのステージ (頂点シェーダー、ジオメトリ シェーダー、およびテセレーション シェーダー) で実装できます。状況に応じたパフォーマンスの測定に基づいて方法を選択するのが理想的ですが、ポイント スプライトを展開するためにジオメトリ シェーダーを使用することは、単純なポイント スプライトをレンダリングするための最も効率的な方法ではないと考えられます。

## 三角形と四角形

レンダリング パイプラインがボトルネックになっている場所によって、三角形または四角形のパフォーマンスが異なる場合があります。フィルレート限界の状況では、四角形のパフォーマンスは三角形よりも優れています。これは、四角形では、透明ピクセルに対してピクセル シェーダーを実行する必要がないためです。レンダリングがジオメトリ ステージで限界状態になる場合は、四角形の代わりに三角形を出力する方が適切です。多くの複雑なレンダリング状況では、三角形の方が四角形よりも高速である場合があります。

このサンプルでは、三角形と四角形を対比させて各方法を示し、異なるサイズのポイント スプライトでの差異を示します。

## 方法 1 - 頂点 + ジオメトリ シェーダー

パイプラインが初期化され、頂点シェーダーが頂点を正常に読み取るようになった後、ジオメトリ シェーダーは、入力頂点ごとに四角形または三角形を出力します。また、これらのテストでは、オンチップとオフチップのジオメトリ シェーダーの割り当てが比較されます。コンパイラは、既定ではオフチップ メモリを使用します。このメモリは、頂点ステージとジオメトリ ステージの間のステージ間データを保存するために使用されます。

長所

* 既知でありシンプル。

短所

* 多くの設定が必要。
* どのケースでも最高速というわけではない。

## 方法 2 – ジオメトリ シェーダーのみ

この方法では、空の頂点シェーダーを使用します。ジオメトリ シェーダーは、頂点データを読み込むために、SV\_PrimitiveID インデックスを使用して、頂点バッファーの raw バイト ビューに対してバッファー読み込みを実行します (ポイント リストでは、ジオメトリ シェーダーの SV\_PrimitiveID は、頂点シェーダーの SV\_VertexID と同じです)。頂点が読み込まれた後、ポイント スプライト展開が方法 1 と同様に実行されます。オンチップとオフチップの GS パフォーマンスがここで同様にテストされます。

長所

* VS は処理不要であるため、入力レイアウトは不要。
* VS と GS の間で内部トラフィックが発生しないため、方法 1 よりも高速に実行される。

短所

* 手動での頂点読み込みには、頂点レイアウトごとに異なるシェーダー バリエーションが必要。コードが高度に最適化されている場合、これは問題ではありません。
* この方法は、理解するのがやや困難です。

## 方法 3 – 頂点シェーダーのみ

DX11 では、ポイントを四角形または三角形に展開するためだけにジオメトリ シェーダーを使用することは実際に必要なく、頂点シェーダーで代用できます。DX11 では、頂点シェーダー ステージで raw バイト UAV を読み取ることができます。そのため、SV\_VertexID と頂点バッファーの raw バイト ビューを使用することで、頂点を手動で読み取ることができます。

したがって、頂点を三角形または四角形に展開するために、描画呼び出しで 3 倍または 6 倍の頂点をレンダリングすること、頂点のインデックスを取得するためにシェーダーで 3 または 6 による除算を実行すること、およびスプライトのコーナーを取得するために除算の余りに基づいて頂点を展開することが必要になります。

長所

* これまでの方法のうち、ポイント スプライトを最も高速にレンダリングする方法。小規模なスプライト (ジオメトリによって制限される場合) に対しては、どのジオメトリ シェーダーの方法よりも 3 倍高速で、より大きいスプライトでも約 50% 高速です。
* システム生成値の SV\_VertexID のみが使用されるため、VS で入力レイアウトは不要。

短所

* 頂点の読み込みは手動であるため、異なる頂点レイアウトを読み込むために異なるシェーダーが必要になる場合がある。絶対最速の方法が必要な場合、これは問題にはなりません。

## 方法 4 – 頂点シェーダーのインスタンス化

インスタンス化を使用して、GPU でポイント スプライトの頂点を読み込むことができます。また、SV\_VertexID を使用して、展開のためにスプライトのコーナーを特定できます。この方法は、直前の方法よりはやや低速ですが、それ以外の方法よりは一貫して高速です。

長所

* 2 番目に高速な方法で、パフォーマンスは方法 3 とほぼ同じ。
* 除算が不要のため、シェーダーで ALU が比較的少ない。
* 頂点の読み込みは入力レイアウトを使用して実行されるため、シェーダーを変更せずに異なる頂点レイアウトを読み込むことができる。
* 非常に大きいサイズのポイント スプライトに対しては最も優れた方法と考えられる。

短所

* なし

## 方法 5 – テセレーション ステージ

テセレーション ステージを使用して、1 つの入力頂点から三角形と四角形を生成できます。この方法のパフォーマンスは、ジオメトリ シェーダー ベースの方式と同等ですが、この方法は、より多くのスプライト シェイプが可能であるため、より柔軟です。たとえば、四角形領域でのテセレーションによって、四角形の代わりに円を出力できます。ピクセル シェーダーが非常に低速で、スプライトが円の場合、これはより効率的な場合があります。

長所

* ほとんどのジオメトリ シェイプを出力できる。

短所

* ジオメトリ シェーダー アプローチと同じ。

## 結論:

上記のすべての方法は、GPU の負荷が異なる場合には、異なった方法で実行されます。そのため、状況に応じて最も有効な方法を選択してください。

# 既知の問題

このサンプルは、Gaming.Xbox.Scarlett.x64 プラットフォームでは現在ビルドできません。

# 更新履歴

2019 年 4 月 12 日 – Xbox サンプル フレームワークから DX12 に移植。

# 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。