ピクセル同期を利用した順不同半透明描画 (更新)

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「Order-Independent Transparency Approximation with Pixel Synchronization (Update 2014)」の日本語参考訳です。

サンプルコードのダウンロード

DirectX SDK (June 2010) への依存性を排除し、Windows* 8 SDK および Visual Studio* 2012/2013 で動作するようにサンプルコードを更新しました。*

任意の数の半透明レイヤーを正しい順序で合成するのは困難です。そのため、半透明描画はリアルタイム・レンダリングにおける基本的な課題です。インテル® Iris™ グラフィックスのピクセル同期拡張を利用する順不同半透明描画 (OIT) サンプルは、第 4 世代インテル® Core™ プロセッサーでこの拡張を利用してリアルタイム・ソリューションを示します。Codemasters* の GRID* 2 および GRID Autosport* はこのアルゴリズムを使用して、図 1 に示すように、木々の葉やレーストラック脇の半透明オブジェクトのレンダリングを向上しました。

図 1. 木々の葉や金網に OIT を適用した Codemasters* GRID* 2 の美しい野外風景

このサンプルは、Marco Salvi 氏、Jefferson Montgomery 氏、および Aaron Lefohn 氏による記事 adaptive-transparency (英語) で紹介されているサンプルコードを基に構築した新しいアルゴリズムを使用します。オリジナルの記事は、アダプティブ半透明描画により、A バッファーの合成から取得したグラウンドトゥルース結果の近似を 5 倍～ 40 倍高速に求める方法を詳しく説明しています。すべての色と深度データをピクセルごとのリストに格納し、それらをソートおよび合成する (図 2) 代わりに、アルファブレンドの方程式を見直して、再帰とソートを回避し、"視感度関数" (VF) を生成するようにしました (図 3)。

図 2. A バッファールーチン

図 3. 視感度関数

ピクセルごとのリストを 1 回スキャンし、ピクセルデータをデータノードの配列に追加します。視感度データの格納に使用するノード数は、視感度関数のステップ数に対応します。特定の画面上の位置に対して、スキャンしたピクセル数がノードのサイズを超えると、アルゴリズムはデータセットのサイズを固定に保ちながら、視感度関数をわずかに変更するため、既存のどのピクセル・フラグメントをマージできるかを計算します。最終ステージでは、視感度関数 vis() を評価し、次の式を使用してフラグメントを合成します:

final_color = $\sum c_i \alpha_i vis(z_i)$ 。

新しいアルゴリズムは、主に 2 つの点でこのアプローチを変更しています。1 つ目は、インテル® Iris™ グラフィックスのピクセル同期拡張を利用していることです。ピクセル同期は、特定のピクセルの読み取り/変更/書き込みの順序付けを行います。2 つのピクセルを画面上の同じ X、Y 位置にレンダリングする場合、関連データへのアクセスで競合状態が発生する可能性があります。ピクセルシェーダー拡張は、ピクセルシェーダーにおけるバリアの役割を果たし、1 つのシェーダーのみ続行できるようにします。どちらのシェーダーが実行されるかはフロントエンドに送られた順番によって決まり、1 つ目のシェーダーが実行を完了すると、もう一方のシェーダーが実行を再開します。図 4 は、この概念図です。

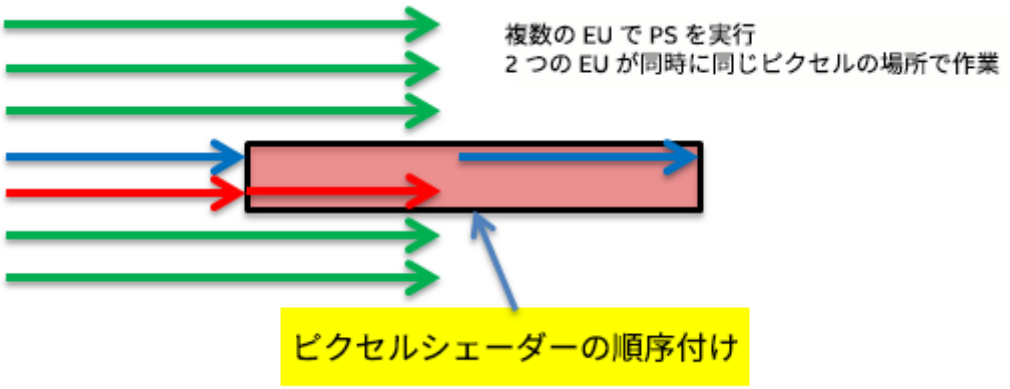

図 4. ピクセルシェーダーの順序付け

これにより、ピクセルを解像フェーズではなくレンダリング時にマージすることができます。挿入フェーズでマージすることで、ピクセルごとのリストを格納する必要がなくなり、アルゴリズムは固定メモリーサイズになります。また、通常 A バッファーやアダプティブ半透明描画アルゴリズムで、リンクリスト・ストレージのオーバーフローにより情報が紛失した場合に見られるアーティファクトも排除します。さらに、占有する帯域幅が減ることでパフォーマンスが向上します。

2 つ目のアルゴリズムの変更は、入力ピクセルを固定のノードセットにマージしていることです。アダプティブ・ルーチンを使用して視感度関数を作成する代わりに、ピクセルをソートし、最も離れたピクセルをマージすることで近似を求めるようにしました。これは、木々の葉のレンダリングのように、同系色のピクセルを

マージする場合に適してします。ユーザーの要件に応じて、異なる挿入ルーチンを簡単に使用できるでしょう。

サンプルは、複雑な形状のレンダリングの課題を示す単純なシーンで構成されています (図 5)。このシーンでは、マテリアルを正しくレンダリングする上で、半透明描画が重要な役割を果たします。

図 5. インテルの OIT サンプル

ユーザーは、次の半透明描画手法を選択することができます。

1. アルファ・ブレンディング (図 6)。最初に立体形状をレンダリングし、内側から外側へソートされた半透明オブジェクトをレンダリングします。
2. アルファ・ブレンディングとアルファ・トゥ・カバレッジ (図 7)。MSAA が必要です。木々の葉のように単純な半透明オブジェクトに対して、深度バッファリングを使用できます。
3. DX11 で実装されているオリジナルのアダプティブ半透明描画ルーチン。
4. インテル® Iris™ グラフィックスのピクセル同期拡張を利用した OIT アルゴリズム (図 8)。

最後のオプションを実行するには、インテル® Iris™ グラフィックスのピクセル同期拡張をサポートするハードウェアが必要です。以下は、各オプションの視覚的な違いを示したものです。

図 6. アルファ・ブレンディング

図 7. アルファ・トゥ・カバレッジ

図 8. ピクセルシェーダーの順序付け

サンプルでは、オリジナルのアルファ・ブレンディング・ソリューションのアーティファクトを排除していません。実際のゲームでは、モデルをさらに分割し、カメラとの相対位置でソートすることで一部のアーティファクトを排除できるでしょう。ここでは単純に、レンダリングのためグラフィックス API へ送る前に半透明の形状をソートしなくても、OIT によって解決されるアーティファクトの種類を示すことを目的としています。半透明の形状が深度バッファーを更新した場合に発生するハローパターンの種類を示すため、アルファ・ブレンディング処理済みの木々の葉を深度バッファーへ読み書きできるようにするチェックボックスがあります。このデバッグ用のオプションは、木々の葉のシェーダーでアルファテストをパスしたピクセルの量を確認できるように用意されています。

サンプルは、ピクセル同期 OIT アルゴリズムの実行時に、次のステップを実行します。最初に、すべての立体形状をシーンへレンダリングします。次に、半透明描画が必要なマテリアルをレンダリングします。この第 2 ステージで、すべての半透明オブジェクトは、**ClearMaskUAV** と AOIT サーフェスを更新します。AOIT サーフェスには、ピクセルごとの色と深度データが格納された複数のノードが含まれます。最後に、全画面の解像パスが、**ClearMaskUAV** が設定されたバックバッファーに半透明のピクセルをマージします。

デバッグ時には、深度バッファーと ClearMaskUAV の値を確認できます。今回の記事およびサンプルコードの更新により、図 9 のように、イメージを拡大し、ピクセルごとの変更を確認できるようになりました。

図 9.イメージを拡大した場合

また、([Enable Pixel Sync] チェックボックスをオンにすることで) 実際のピクセル同期を無効にしたまま、ピクセル同期 OIT ルーチンを実行できるようになりました これにより、同期プリミティブが利用できない場合に、競合状態によって生成されるアーティファクトをピクセルごとのレベルで確認できます。インテルの OIT アルゴリズムは、データ圧縮の程度 (2、4、または 8 ノード) に応じて、異なる品質レベルを提供します。ノード数が多いほうが、より正確に視感度関数の近似を求めることができますが、より多くのメモリーと帯域幅が必要になります。GRID* 2 は、パフォーマンスと画質を考慮して 2 ノードバージョンを使用することで、わずかな画質の違いで大幅なパフォーマンスの向上を達成しました。

アルゴリズムのメインコードは、**AOIT.hlsl** にあるピクセルシェーダー関数 **WriteNewPixelToAoit** と **AOIT_resolve.hlsl** にある **AOITSPResolvePS** 関数です。一般に、最も時間がかかるルーチンは、挿入フェーズの **WriteNewPixelToAoit** です。ピクセルシェーダーでこのルーチンを呼び出す場合、**earlydepthstencil** テストを使用して隠しピクセルを排除することで、パフォーマンスを大幅に向上できます。この時点での深度バッファーの正確さと完全性により、より美しい半透明の形状を描画することができます。これが、GRID* 2 の最適化につながりました。GRID* 2 では、木々の葉がほぼ 100% 不透明でない場合、不要な描画を減らすため、木々も深度バッファーにレンダリングされました。

OIT-Update-2014.zip (83.71MB)

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