ソフトウェアによるオクルージョン・カリング

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「Software Occlusion Culling」の日本語参考訳です。

サンプルコードのダウンロード (Web サイト)

更新 2016/1/15
更新 2013/9/6
更新 2013/3/22

概要

この記事は、ソフトウェアによるオクルージョン・カリング (視界に入らないオブジェクトを描画しない) のアルゴリズムと関連するサンプルコードの詳細に触れています。サンプルコードはダウンロード可能です。この技術は、シーンのオブジェクトを occluder (オクルーダー) と occludee (オクルーディー) に分割し、ソフトウェアにより深度バッファーにラスタライズされたオクルーダーとの深さ比較に基づいてオクルーディーをカリングします。サンプルコードは、錐台カリングを利用し、インテル® ストリーミング SIMD 拡張命令セット (インテル® SSE) とマルチスレッドで最適化することにより、カリングされていないサンプルシーンの表示に比べ、8 倍の高速化を達成しています。

ソフトウェアによるオクルージョン・カリング

更新 2016/1/15

カリングパスで費やされる時間を短くするため、オクルージョン・カリングのサンプルをインテル® アドバンスト・ベクトル・エクステンション 2 (インテル® AVX2) で最適化しました。インテル® Core™ m7-6Y75 プロセッサーは、TDP 4.5W のデュアルコア・プロセッサーです。表 1 は、このプロセッサーにおけるインテル® ストリーミング SIMD 拡張命令 4.1 (インテル® SSE4.1) バージョンとインテル® AVX2 バージョンのパフォーマンスの比較です。カリング時間はシーンおよび深度バッファーの解像度により異なります。インテル® AVX2 では、(インテル® AVX2 の時間を分母とした場合) カリング時間の短縮率は最大 32% になります。

表 1. インテル® Core™ m7-6Y75 プロセッサーにおけるサンプルの 2 つのシーンのカリング時間 (ミリ秒)。
システム構成: インテル® Core™ m7-6Y75 プロセッサー、1.20GHz、TDP 4.5W、2 コア 4 スレッド、
8GB DDR3、インテル® HD グラフィックス 515、ドライバー 15.40.10.64.4300、
Microsoft* Windows* 10 Professional 64 ビット (ビルド 10586)

解像度	シーン	インテル® SSE4.1 (ミリ秒)	インテル® AVX2 (ミリ秒)	インテル® SSE4.1/ インテル® AVX2
320x192	1	2.04	1.90	1.07
	2	0.95	0.82	1.16
640x360	1	2.72	2.50	1.09
	2	1.90	1.44	1.32
1280x720	1	4.55	4.04	1.13
	2	4.29	3.62	1.19
1920x1080	1	8.38	7.80	1.07
	2	9.37	7.40	1.27

深度バッファーの解像度は、オクルージョン・カリングの CPU 資源を制御するために変更できるパラメーターです。表 1 は、解像度を 1920x1080 から 320x192 にすると時間が 7.80 ミリ秒から 1.90 ミリ秒になることを示しています。低電力システム向けに設計されたゲームでは、このようなパラメーターのチューニングを考慮する必要があります。インテル® Core™ m7-6Y75 プロセッサーは、タブレットと同レベルの消費電力で動作し、Windows* 10 のようなデスクトップ・オペレーティング・システムを実行できます。ゲームコンテンツをより小さな電力バジェットに合わせてスケーリングする場合でも、Windows* で動作するゲームをこれらのシステムで実行できる可能性があります。

オクルージョン・カリングはゲームのパフォーマンスを向上する目的でよく使用されます。カリングは GPU で行うことができますが、このサンプルでは CPU でカリングを行う方法を示しています。CPU でオブジェクトをカリングすると、GPU でカリングするよりも利点があります。例えば、ゲームによっては GPU がボトルネックになることがあります。そういったケースでは、CPU でカリングを行うと、オクルージョン・クエリーの GPU への送信コストを節約でき、GPU のボトルネックを緩和できます。この送信コストには、Direct3D* API 呼び出しおよびオクルーダーの CPU と GPU 間のメモリー転送が含まれます。

更新 2013/9/6

この更新は、Fabian Giesen により実装され、サンプルに統合された、2 つのラスタライザーの最適化からなります。これらの最適化により、パフォーマンスが (フレームレートで約 13%、合計カル時間で約 27%) 向上しました。詳細は、https://software.intel.com/en-us/blogs/2013/09/06/software-occlusion-culling-update-2 (英語) を参照してください。

更新 2013/3/22

この更新は、新機能と最適化からなります。合計カル時間は約 1/4 に、合計フレーム時間は約 1/2 になりました。詳細は、https://software.intel.com/en-us/blogs/2013/03/22/software-occlusion-culling-update (英語) を参照してください。

はじめに

どんな現実的な 3D シーンにも、少なくとも一部がオクルードされた (隠された) オブジェクトが含まれています。例えば、ビルとその周囲では多くのオブジェクトがオクルードされます。z バッファーは、オブジェクトの一部をオクルードできるレンダリング・パイプラインの最終ステージです。しかし、オクルードされたオブジェクトは (視覚効果がないにもかかわらず) レンダリングのため GPU に送られます。オクルードされたオブジェクトの計算コストを下げることができれば、シーンの品質に影響を与えることなく、アプリケーションのパフォーマンスを向上できます。

GPU が生成した深度バッファーを CPU で使用するのは、オクルードされたオブジェクトを早期に発見する 1 つのテクニックです。しかし、GPU の処理は CPU よりも数フレーム遅れているため、この方法でオクルージョン・カリングを行うと、レイテンシーやオーバーヘッド問題が発生し、ビジュアル・アーティファクトが発生する可能性があります。これらの問題を回避するには、CPU の深度バッファーをソフトウェアによりラスタライズします。

このアプローチでは、CPU を使用して深度バッファーをラスタライズします。次に、軸平行境界ボックステストを使用して、オブジェクトがオクルードされているかどうか判断します。オーバーヘッドを小さくするため、オクルードされているオブジェクトをレンダリング・パイプラインから削除します。全体または一部が可視のオブジェクトはカリングしないで、レンダリングのため GPU に送ります。ソフトウェアによるオクルージョン・カリングのサンプルに関連したサンプルコードは、この手法を実装しています。さらに、このサンプルでは、パフォーマンスを向上するため、ソフトウェアによるラスタライザーはインテル® SSE を使用してベクトル化され、マルチスレッド化されています。

キーワード

オクルーダー (occluder): シーンのほかのオブジェクトを隠すまたはオクルードするのに十分大きなオブジェクト。

オクルーディー (occludee): シーンのほかのオブジェクトにより隠されるまたはオクルードされるオブジェクト。

ソフトウェアによるオクルージョン・カリングには、深度バッファーのラスタライズと深度テストカリングの 2 つのステップがあります。次のセクションで、これらのステップに関する詳細を説明します。

図 1: ソフトウェアによるオクルージョン・カリングのサンプルのスクリーンショット

深度バッファーのラスタライズ

シーンのオクルーダーは CPU の深度バッファーにラスタライズされます。図 1 は、ソフトウェアによるオクルージョン・カリングのサンプルのスクリーンショットです。城壁と地面はシーンのオクルーダーと見なされます。人工物の特別な前処理を避けるため、細かな装飾物を加えた城壁をオクルーダーとして使用します。これは理想的には、大きな城壁のみをオクルーダーとして使用するほうが良いのですが、ソフトウェアによるオクルージョン・カリング・アルゴリズムが手間のかかるコンテンツの変更を行うことなく現実的なシーンで動作することを確認したかったためです。

このラスタライザーは、フレームバッファーをタイルに分割し、タイルとの交差に基づいてオクルーダー三角形をビニングします。三角形がタイルの境界をまたいでいる場合、ラスタライザーは三角形を複数のタイルにビニングします。タイルを処理するとき、スレッドはビニングされた三角形を処理し、境界ボックストラバースを使用して三角形を深度バッファーにラスタライズします。ラスタライザーは対象のピクセルが三角形の内部にあるかどうかチェックして、内部にある場合は、重心座標を使用してピクセル位置の深度値を変更します。新しく計算された深度が同じピクセル位置のすでに存在する深度よりもオブザーバーに近い場合、深度バッファーのラスタライズ・プロセスはピクセル位置の深度バッファーを更新します。

深度バッファーが CPU でラスタライズされたら、どのオクルーディーが可視で、どれをカリングできるか判断するため、シーンのすべてのオクルーディーの深度テストを行います。

深度テストカリング

ここでは、オブジェクト空間軸平行境界ボックス (AABB) を使用して、CPU が生成した深度バッファーに対してシーンのオクルーディーの深度テストを行います。深度テスト・アルゴリズムは、オクルーダーを含むシーンのすべてのオブジェクト (城壁および地面) をオクルーディーとして扱います。AABB を使用すると深度テストはより保守的になります。AABB がオクルードされている場合、その内部に含まれているオブジェクトもオクルードされていて、カリングすることができます。AABB が可視の場合、内部に含まれるオブジェクトも可視であると仮定されます。しかし、境界ボックスが保守的だと、この仮定は必ずしも真とは限りません。また、誤検出されることもあります。

理想的には、カメラが AAAB の内部にある場合、境界ボックスをクリップし、カメラの正面のオブジェクト部分を処理すべきです。しかし、サンプルではクリップを実装していません。このため、オクルーディーの境界ボックスがニア・クリップ・プレーンでクリップされる場合、オクルーディーを可視として扱い、レンダリングのため GPU へ送っています。

オブジェクトがニア・クリップ・プレーンでクリップされるかどうかの判断には、境界ボックスを構成する頂点のホモジニアス座標 ***w*** を使用しました。サンプルでは、ニア・クリップ・プレーンを 1.0 に設定しました。カメラの前のオブジェクトは ***w> 1.0*** になります。境界ボックスを構成する頂点のいずれかが ***w < 1.0*** の場合、そのオブジェクトはニア・クリップ・プレーンによりクリップされます。

しかし、ラスタライザーの複数の場所で ***w*** による除算が行われるのを避けるため、ホモジニアス座標 ***w*** の代わりに ***1/w*** を使用しました。

$0.0 < w < 1.0$ の場合 $1/w > 1.0$
しかし $w < 0.0$ の場合は $1/w < 0.0$

そのため、境界ボックスの任意の頂点が ***1/w > 1.0*** または ***1/w < 0.0*** の場合、オクルーディーの境界ボックスはクリップされ、オクルーディーを可視として扱います。

錐台カリングが有効な場合、AABB 深度テスト・アルゴリズムは視錐台の完全に内側のオクルーディー、または視錐台によりクリップされるオクルーディーを処理します。つまり、カメラの後ろのオクルーディーは錐台カリングされ、レンダリングされません。サンプルは、錐台カリングが無効な場合でも深度バッファーカリングを有効にすることができます。この機能は実用的なものではなく、錐台カリング単独で得られるパフォーマンスの向上を測定するために提供したものです。この設定には既知の問題があることに注意してください。カメラの後ろのオブジェクトをすべて扱うため、錐台カリングが無効で深度テストカリングが有効な場合、描画呼び出しの数は増加します。

オブジェクトが可視で、その境界ボックスの最初のピクセルが深度テストをパスすると、深度テスト・アルゴリズムはオブジェクトを可視としてマークし、レンダリングを行うため GPU へ送ります。しかし、オブジェクトと境界ボックスが完全にオクルードされている場合、深度テスト・アルゴリズムは境界ボックスのピクセルをすべてテストします。つまり、深度テスト・アルゴリズムは、オブジェクトがレンダリングのため GPU に送られるとき最小の時間を費やし、オブジェクトがカリングされるとき最大のワークを実行します。

最適化

ソフトウェアによるオクルージョン・カリングのサンプルには、複数の最適化が実装されています。

- **ビニング:** 深度バッファーのラスタライズで説明したように、オクルーダー三角形を深度バッファーにラスタライズする前に、ビニングの事前パスが三角形で実行されます。フレームバッファーはタイルに分割され、オクルーダー三角形はタイルとの交差に基づいてビニングされます。各タイルは個々に動作し、三角形を深度バッファーにラスタライズするとき、タイルの内側のピクセルのみ考慮されます。ここでは、2 つ以上のスレッドが同時に同じピクセルを処理することを防ぐためにビニングを使用しました。ビニングによりスレッドは同じ小さなタイルと繰り返し対話するため、キャッシュ・コヒーレンシーの維持にも役立ちます。
- **錐台カリング:** 錐台カリングが有効な場合、視錐台内部のオブジェクトのみ処理されます。
- **インテル® SSE を使用したベクトル化:** ソフトウェアによるオクルージョン・カリングのサンプルは、インテル® SSE 組込み関数を使用して最適化されています。可能であれば、コードは 4 つの三角形または 4 つのピクセルを一度に処理します。
- **タスク化:** ソフトウェアによるオクルージョン・カリングのサンプルのマルチスレッド化には、インテル® スレッディング・ビルディング・ブロック (インテル® TBB) を使用しました。インテル® TBB は、開発者がタスクのセットを CPU のすべての利用可能なコアに分散して優れたロードバランスを達成できるライブラリーです。サンプルは、深度バッファーのラスタライズおよび深度テストのアルゴリズムをマルチタスクにするために作成されたタスキングシステム (Minadakis, 2011) を使用しています。タスク・マネージャーは、インテル® TBB により実行されるタスク作成と同期プロセスを抽象化および単純化するために使用されます。
- **オクルーダー・サイズしきい値:** オクルーダー・サイズしきい値を使用して、深度バッファーにラスタライズされるオクルーダーの数を制限します。画面領域サイズがしきい値よりも大きなオクルーダーは、シーンのほかのオブジェクトをオクルードできる十分な大きさがあり、深度バッファーにラスタライズされます。深度バッファー・ラスタライザーはシーンのほかのオクルーダーをラスタライズしないため、深度バッファーのラスタライズ時間が短縮されます。
- **オクルーディー・サイズしきい値:** オクルーディー・サイズしきい値を使用して、描画のため GPU に送られるオクルーディーの数を制限します。画面領域サイズがしきい値よりも小さいオクルーディーは非常に小さいため (数ピクセル)、可視の場合でも描画しません。

ソフトウェアによるオクルージョン・カリングのサンプルの実行

図 1 は、ソフトウェアによるオクルージョン・カリングのサンプルのスクリーンショットです。サンプルには、サンプルの動作を調査および変更できるさまざまなユーザー・インターフェイス要素が含まれています。

Rasterizer technique ドロップダウン・リストを使用して、アルゴリズムのバージョン (スカラーまたはインテル® SSE) を選択します。

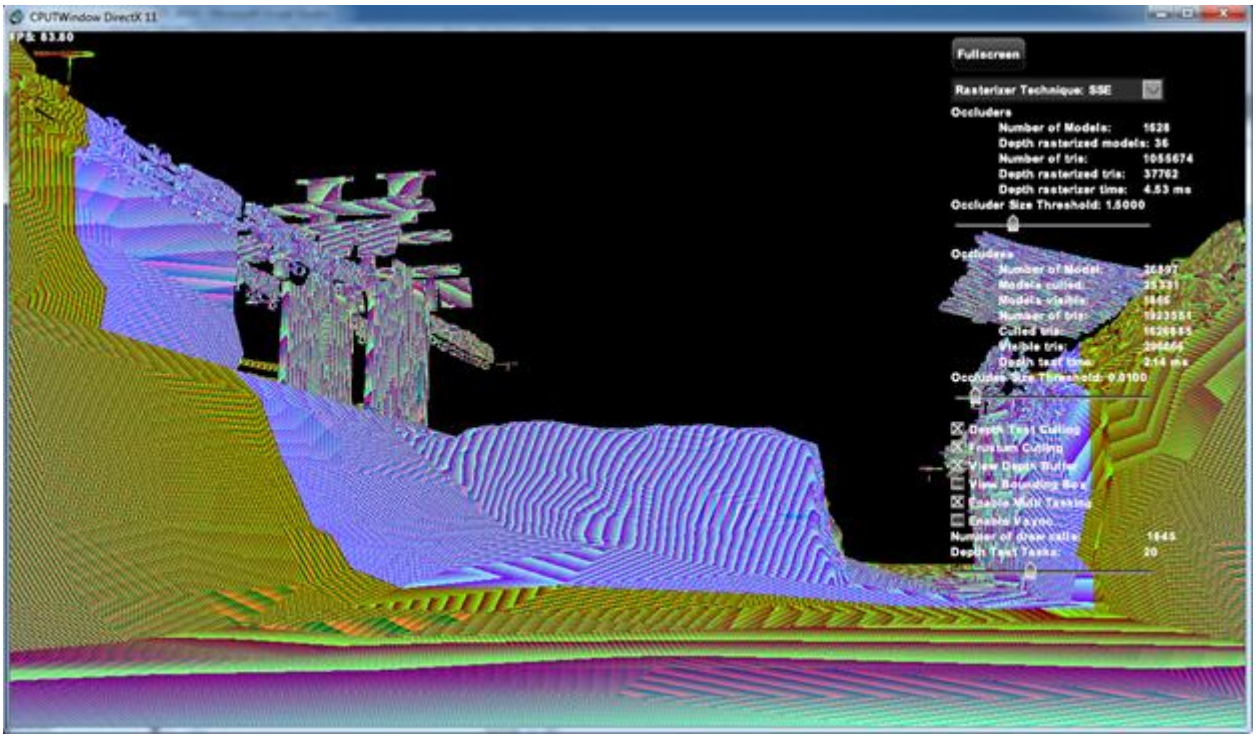

図 2: CPU ラスタライズされた深度バッファーのスクリーンショット

GUI の **Occluders** セクションに、シーンのオクルーダーに関する情報が表示されます。

- **Number of Models:** シーンのオクルーダー・モデルの数
- **Depth rasterized models:** CPU で深度バッファーにラスタライズされたオクルーダー・モデルの数。
- **Number of tris:** オクルーダー・モデルの三角形の数。
- **Depth rasterized tris:** 深度バッファーに CPU ラスタライズされたオクルーダー三角形の数。
- **Depth rasterizer time:** オクルーダー・モデルを深度バッファーに CPU ラスタライズするためにかかった時間。
- **Occluder Size Threshold:** このスライダーを変更して、深度バッファーにラスタライズするオクルーダー・モデルを決定します。オクルーダーの画面領域サイズがしきい値よりも大きい場合、そのオクルーダーはシーンのほかのオクルーディーを隠すことができる十分な大きさがあり、深度バッファーにラスタライズされます。

GUI の **Occludees** セクションには、シーンのオクルーディーに関する情報が表示されます。

- **Number of Models:** シーンのオクルーディー・モデルの数。
- **Models culled:** オクルーダーにより隠されカリングされたオクルーディー・モデルの数。
- **Models visible:** シーンの可視のオクルーディー・モデルの数。
- **Number of tris:** オクルーディー・モデルの三角形の数。
- **Culled tris:** シーンでオクルードされカリングされたオクルーディー三角形の数。
- **Visible tris:** シーンの可視のオクルーディー三角形の数。
- **Depth test time:** CPU が生成した深度バッファーに対してオクルーディー・モデルの AABB の深度テストを行うためにかかった時間。
- **Occludee Size Threshold:** このスライダーを変更して、描画のため GPU に送られるオクルーディー・モデルを決定します。オクルーディーの画面領域サイズが非常に小さい場合 (数ピクセル)、可視の場合でも、そのオクルーディーは描画しません。

Depth Test Culling チェックボックスは、サンプルのソフトウェアによるオクルージョン・カリングを有効 / 無効にします。

Frustum Culling チェックボックスは、サンプルの錐台カリングを有効 / 無効にします。錐台カリングが有効な場合、視錐台内部のオクルーダーおよびオクルーディーのみサンプルで処理されます。

View Depth Buffer チェックボックスは、CPU ラスタライズされた深度バッファーの表示を有効 / 無効にします。有効な場合、深度バッファーの FP32 深度値は DXGI_FORMAT_R8G8B8A8_UNORM カラーとして扱われ、画面に表示されます。図 2 は、CPU ラスタライズされた深度バッファーです。

View Bounding Box チェックボックスは、オクルーダーおよびオクルーディーで AABB の描画を有効 / 無効にします。

Multi Tasking チェックボックスは、サンプルのマルチスレッドを有効 / 無効にします。

Vsync チェックボックスは、サンプルのフレームレートの制限を有効 / 無効にします。有効な場合、サンプルのフレームレートは 60FPS に制限されます。

Number of draw calls は、シーンの描画を完了するために発行された描画呼び出しの数を示します。

Depth Test Tasks スライダーを使用して、シーンのオクルーディーの深度テストを行うためにタスク・マネージャーにより作成されるタスク数を変更します。

パフォーマンス

ソフトウェアによるオクルージョン・カリングのサンプルのパフォーマンスは、第 3 世代インテル® Core™ プロセッサー (Ivy Bridge)、2.30GHz、4 コア / 8 スレッド、インテル® HD Graphics 3000 のシステムで測定されました。ラスタライザーの手法をインテル® SSE、オクルーダー・サイズのしきい値を 1.5、オクルーディー・サイズのしきい値を 0.01、深度テストタスクの数を 20 にそれぞれ設定しました。錐台カリングおよびマルチタスクを有効にして、vsync を無効にしました。

城のシーンのオクルーダー・モデルは 1628、オクルーダー三角形は 105 万でした。オクルーディー・モデルは 26897 (オクルーダーはオクルーディーとして扱われました)、オクルーディー三角形は 190 万でした。

オクルーダーを CPU の深度バッファーにラスタライズするためにかかった時間は 3.77 ミリ秒、オクルーディーの深度テストにかかった時間は 2.23 ミリ秒でした。

	インテル® SSE		
	マルチスレッド	マルチスレッド + 錐台カリング	マルチスレッド + 錐台カリング + 深度テストカリング
フレームレート (fps)	3.75	10.42	31.54
フレーム時間 (ミリ秒)	266.67	95.96	31.71
描画呼び出しの数	22877	7683	1845

表 1: インテル® SSE バージョンのパフォーマンス。フレームレートの単位はフレーム毎秒 (fps)、フレーム時間の単位はミリ秒。

表 1 は、インテル® SSE バージョンのパフォーマンスを示しています。マルチスレッドと錐台カリングが有効な場合、マルチスレッドのみ有効な場合の約 3 倍にパフォーマンスが向上しました。同様に、マルチスレッド、錐台カリング、深度テストカリングが有効な場合、マルチスレッドのみ有効な場合の約 8 倍、マルチスレッドと錐台カリングが有効な場合の約 3 倍にそれぞれパフォーマンスが向上しました。

今後の予定

次のように、ソフトウェアによるオクルージョン・カリングのサンプルのさらなる最適化を予定しています。

- インテル® AVX2 を使用してラスタライザー・コードを最適化する。ラスタライザーは固定小数点演算を実装していてインテル® AVX2 は整数演算をサポートしているため、インテル® AVX2 を使用してラスタライザーを最適化できます。
- レンダリングのため GPU に送るオブジェクトを決定するため、ソフトウェアによるオクルージョン・カリング・アルゴリズムの実行を CPU が完了するのを GPU が待つ必要がないように、パイプライン処理を実装する。
- 深度バッファー・ラスタライザーのビニングなしバージョンを実装して、そのパフォーマンスをビニングありバージョンと比較する。
- より小さな深度バッファーでテストしてパフォーマンスへの影響を確認する。CPU 深度バッファーのサイズは現在 1280x720 ピクセルに設定されています。これはフレームバッファーと同じ解像度です。深度バッファーのサイズは、*SCREENW* および *SCREENH* の値を変更して、サンプルを起動する前に変更できます。
- タイル数を変更する。現在の深度バッファーはビニングのために 32 のタイルに分割されています。このタイル数を変更して、パフォーマンスへの影響をテストします。

リリースノート

サンプルのリリースノートは次のとおりです。

- デバッガーをアタッチしてリリースモードでサンプルを実行すると (<F5>)、ロード時間は長くなります。デバッガーをアタッチしないでサンプルを実行すると (<Ctrl> <F5>)、ロード時間は大幅に短くなります。
- パフォーマンスと対話性を向上するため、サンプルはいくつかのモデルを意図的にデバッグモードでロードしています。

参考文献

Anderson, J. (2009). Parallel Graphics in Frostbite-Current and Future. *Siggraph*, (pp. http://s09.idav.ucdavis.edu/talks/04-JAndersson-ParallelFrostbite-Siggraph09.pdf).

Intel Threading Building Blocks. (n.d.). http://threadingbuildingblocks.org/

Kas. (2011, Feb). Software Occlusion Culling. http://fatkas.blogspot.com/2011/02/software-occlusion-culling.html.

Marschner, P. S. (2009/07/21). *Fundamentals of Computer Graphics [Hardcover]* (3rd ed.).

Minadakis, Y. (2011, March). タスク処理によるゲーム・エンジン・システムのスケーリング