ダイナミック解像度レンダリング

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「Dynamic Resolution Rendering Article」の日本語参考訳です。

記事とソースコードのダウンロード

ダイナミック解像度レンダリングのアップデート (英語) [PDF 0.3MB]
ダイナミック解像度レンダリング (英語) [PDF 1.1MB]
ダイナミック解像度レンダリングのサンプルページ (英語)

はじめに

3D ゲームの登場以来、画面解像度の選択は PC ゲームの特徴的な要素の 1 つとされてきましたが、もはやその必要はありません。この記事と付属のサンプルコードでは、静的な解像度を選択する代わりに、動的にレンダリングの解像度を変える方法を紹介します。

ダイナミック解像度レンダリングでは、ビューポートを使用してレンダーターゲットの一部にレンダリングを制限することで 3D シーンをレンダリングする解像度を調整し、これを出力バックバッファーにスケーリングします。グラフィカル・ユーザー・インターフェイス・コンポーネントは、通常描画にかかるコストが低いため、バックバッファーの解像度でレンダリングできます。その結果、高品質の GUI と安定した高いフレームレートを達成できます。

この記事で紹介するパフォーマンス結果とスクリーンショットは、インテル® HD グラフィックス 3000 内蔵第 2 世代インテル® Core™ i7 モバイル・プロセッサー (インテル® マイクロアーキテクチャー開発コード名 Sandy Bridge、D1 ステッピング、クアッドコア 2.40GHz CPU、4GB DDR3 1333MHz RAM) を使用して取得しました。

この記事と付属のサンプルコードは、2011 年にサンフランシスコで開催された Game Developers Conference (GDC) で紹介されたものです。当時のビデオは GDC Vault [GDC Vault 2011] にあります。また、プレゼンテーション・スライドはインテルの Web サイトから入手できます [Intel GDC 2011]。GDC でのプレゼンテーション以来、いくつかのゲーム会社がすでにこの手法をコンソールで採用しています。そのうち公開されているのは、LucasArts 社の Dmitry Andreev 氏によるアンチエイリアシングのプレゼンテーションのみで、使用されているダイナミック解像度の手法についてはわずかしか紹介されていません [Andreev 2011]。

図 1: スタティック・カメラ・ビューポイントの 1 つから見たサンプルシーン

目的

ゲームはほとんどの場合、解像度によるパフォーマンスの変動が大きく、後処理手法とシェーダーの複雑さの増加により、最近のゲームでも 1 ピクセルあたりのコストが重視される傾向が続いています。解像度を上げると、テクスチャー・サンプリングとレンダーターゲットの帯域幅も増えます。そのため、システムのパフォーマンスに応じて適切な解像度を設定することが重要です。解像度を動的に変えられることは、ゲームが安定した適切なフレームレートを維持できるように追加のパフォーマンス制御オプションを開発者に与え、全体的な体験の質を向上します。

ネイティブのスクリーン解像度でグラフィカル・ユーザー・インターフェイスをレンダリングすることは、ロールプレイング、リアルタイム戦略、そして大規模なマルチプレーヤー・ゲームにとって特に重要です。ローエンドシステムでも、プレーヤーはチームメートの統計を見ながら複雑なチャットを楽しむことができます。

最後に、PC ゲームにおいてノートパソコンが優勢になるにつれて、ゲーム開発における電力消費の重要性が高まるでしょう。マシンが主電源からバッテリーに切り替わる際にパフォーマンス設定によっては CPU や GPU 周波数が低下する可能性がありますが、ダイナミック解像度レンダリングではゲーム自身が解像度を自動調整してこれに対応することができます。ゲームによっては、電力消費をさらに抑えて長時間ゲームをプレイできるように、ユーザーが省電力プロファイルを選択できるように考えるかもしれません。サンプルコードを使用した検証では、垂直同期が有効な場合、解像度を 1/2 に下げるとプロセッサー・パッケージの電力消費が通常時の 1/7 となり、フレームレートが維持されることが分かりました。

基本原則

ダイナミック解像度レンダリングの基本原則は、ビューポイントを使用してオフスクリーン・レンダー・ターゲットの一部にレンダリングを制限し、ビューにスケーリングすることです。例えば、レンダーターゲットのサイズは (1920, 1080) ですが、ビューの原点は (0, 0) でサイズは (1280, 720) にすることができます。

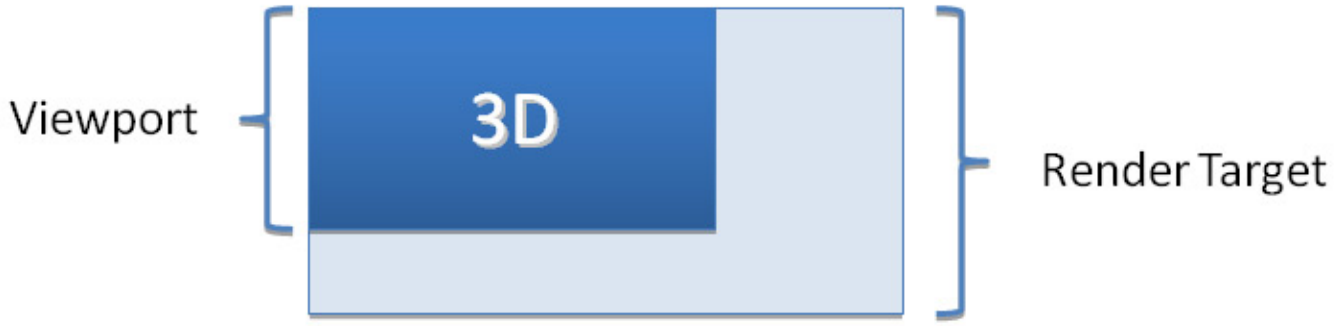

図 2: ビューポイントによるレンダリングの制限

バックバッファーよりも大きいレンダーターゲットを作成することで、サブサンプルからスーパーサンプルまでさまざまなダイナミック解像度を利用できます。必要なレンダーターゲットとテクスチャーの完全なセットがグラフィックス・メモリーに収まるようにする必要がありますが、インテル® プロセッサー・グラフィックス・ベースのシステムではシステムメモリーを使用するため、通常かなりのメモリー容量を利用できます。

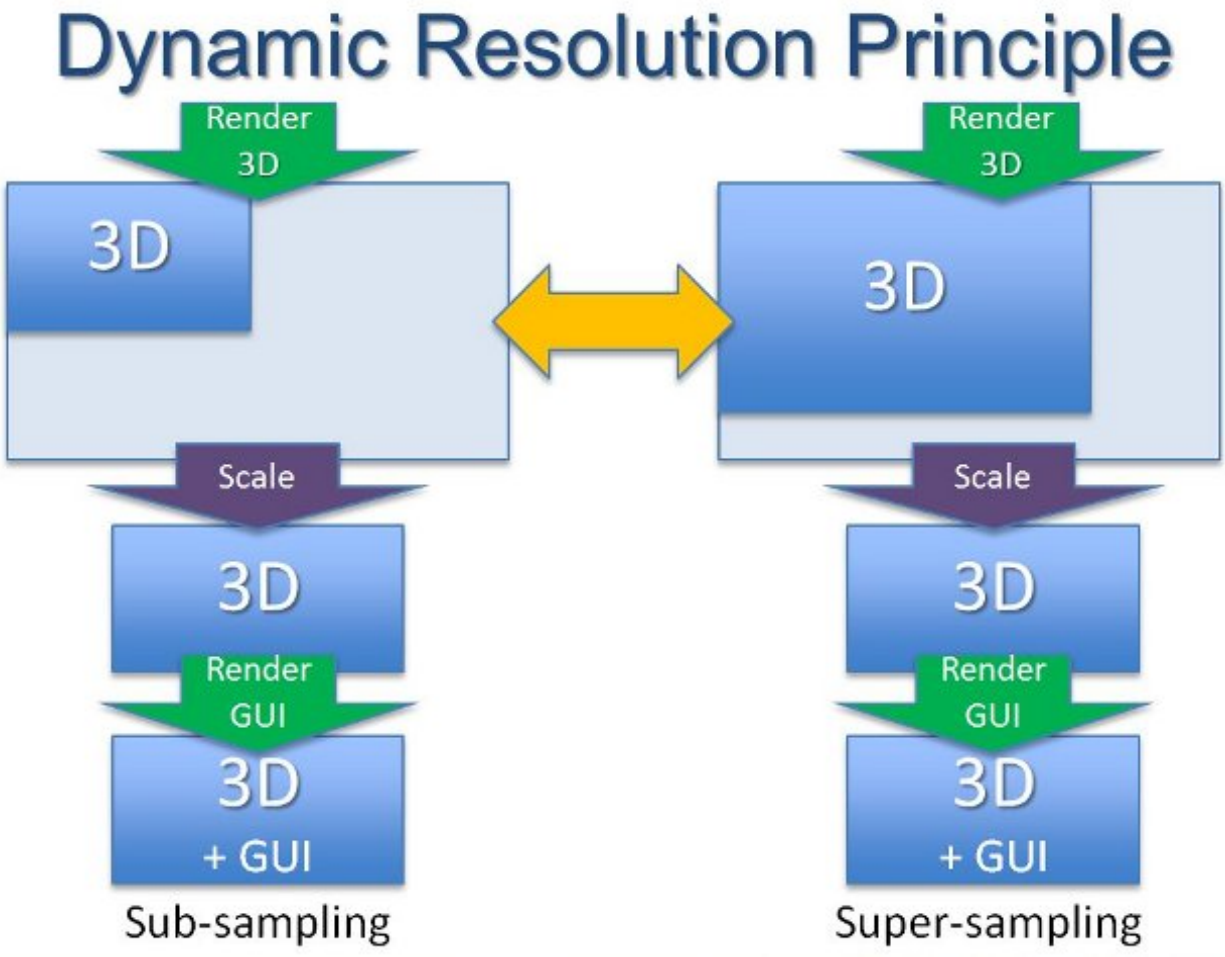

図 3: サブサンプルからスーパーサンプルまでさまざまなダイナミック解像度を利用可能

ダイナミック・ビューポイントへ通常のレンダリングを行う場合、ラスタライズ規則によって必要な変更が行われるため、変更は不要です。ただし、レンダーターゲットからレンダリングする場合、座標を適切にスケーリングし、右端と下端のクランプを処理する必要があります。

次のピクセルシェーダーのサンプルコードは、UV をクランプする方法を示します。これは主に依存読み取り (後でダイナミック・レンダー・ターゲットからのサンプリングに使用される UV 上のピクセルごとの操作) で使用されます。

```
// Clamp UVs to texture size
// PSSubSampleRTCurrRatio is the fraction of the render target in use.
float2 clampedUV = min( unclampedUV,  g_PSSubSampleRTCurrRatio.xy );
```

モーションブラー (レンダーターゲットからの依存読み取りを使用する一般的な後処理操作) では、シェーダーはテクスチャー-フェッチに依存するため、必要な追加の計算はパフォーマンスにほとんど影響しません。

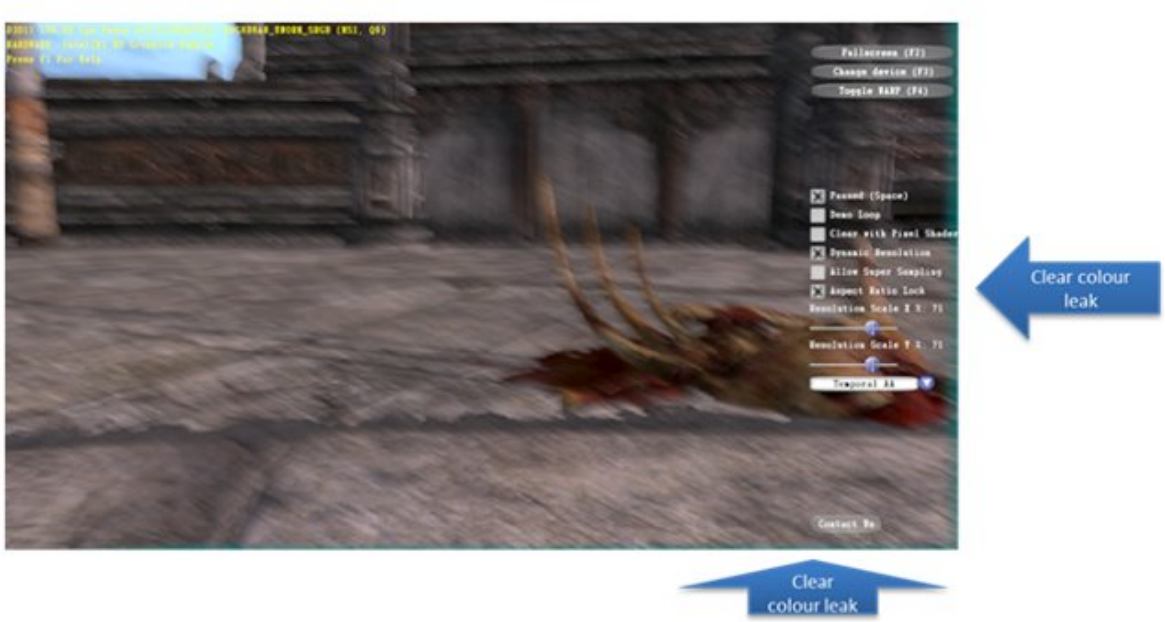

図 4: モーションブラーによる画面端の色落ち (クランプにより解決可能)

さらに、シェーダーで使用される解像度比が、単にアプリケーションに適した比率ではなく、実際のビューポイント比を表すものであることが重要です。これは、ダイナミック・ビューポイントのディメンジョンから比率を再計算することで簡単に取得できます。例えば、サンプルコードの関数 DynamicResolution::SetScale では、スケーリングが境界条件を満たすことを確認した後に次の処理が実行されます。

```
// Now convert to give integer height and width for viewport
m_CurrDynamicRTHeight     = floor( (float)m_DynamicRTHeight * m_ScaleY / m_MaxScalingY );
m_CurrDynamicRTWidth      = floor( (float)m_DynamicRTWidth *  m_ScaleX / m_MaxScalingX );

// Recreate scale values from actual viewport values
m_ScaleY = m_CurrDynamicRTHeight * m_MaxScalingY / (float)m_DynamicRTHeight;
m_ScaleX = m_CurrDynamicRTWidth *  m_MaxScalingX / (float)m_DynamicRTWidth;
```

スケーリング・フィルター

3D シーンをレンダリングした後、ビューポイント領域をバックバッファーの解像度にスケーリングする必要があります。さまざまなフィルターを使用してこの処理を実行できます。サンプルコードでは、ここで紹介するいくつかの例を実装します。

ポイントフィルター

ポインターフィルターは、高速な基本フィルターです。0.71 倍のダイナミック・ビューポイントから 1280x720 へのスケーリングには約 0.4ms かかります。

バイリニアフィルター
バイリニアフィルターは、ハードウェア・サポートによりポイントフィルターと同じくらい高速です。スムージングによりエイリアシング・アーティファクトを軽減しますが、シーンもぼかします。0.71 倍のダイナミック・ビューポイントから 1280x720 へのスケーリングには約 0.4ms かかります。

バイキュービック・フィルター
バックバッファーの 0.5 倍の解像度では、バイキュービック・フィルターはバイリニアフィルターよりもわずかに品質が優れていますが、パフォーマンスは高速なバイキュービック・フィルターを使用した場合であってもバイリニアフィルターの 1/7 です [Sigg 2005]。0.71 倍のダイナミック・ビューポイントから 1280x720 へのスケーリングには約 3.5ms かかります。

ノイズフィルター
ポイントフィルターにノイズを追加して高い周波数を追加することで、低コストでエイリアシングをわずかに解消できます。サンプルコードの実装は基本的なものであり、フィルム・グレイン・フィルターを改善することで個別のレンダリングに適した効果が得られます。0.71 倍のダイナミック・ビューポイントから 1280x720 へのスケーリングには約 0.5ms かかります。

ノイズ・オフセット・フィルター
スケーリング中にサンプリング位置に小さなランダムオフセットを追加すると、エイリアスエッジの規則性を軽減できます。このアプローチは、シャドウマップの高速フィルターでよく使用されます。0.71 倍のダイナミック・ビューポイントから 1280x720 へのスケーリングには約 0.7ms かかります。

テンポラル・アンチエイリアシング・フィルター
このスケーリング・フィルターは、X と Y を半ピクセル分オフセットした奇数フレームと偶数フレームをレンダリングするため、最初のレンダリング・パス中に追加のサポートを必要とします。ゴースト・アーティファクトを除去するようにフィルターした場合、2 倍のピクセルからサンプリングすることで、結果として得られる画像の品質が大幅に向上します。このフィルター手法については、後で詳しく説明します。0.71 倍のダイナミック・ビューポイントから 1280x720 へのスケーリングには約 1.1ms かかり、フル解像度とほぼ同じ品質が得られます。

テンポラル・アンチエイリアシングの詳細
テンポラル・アンチエイリアシングは以前からある手法ですが、連続するフレーム内のオブジェクト位置の違いから生じるゴースト問題によって使用が制限されていました。現代のレンダリング手法では、パフォーマンス・オーバーヘッドが低いことから、魅力的なオプションと考えられています。

基本アプローチは、X と Y を半ピクセル分ジッタリング (オフセット) した奇数フレームと偶数フレームをレンダリングします。サンプルコードでは、射影行列を変換することでこれを行っています。最後のスケーリングでは、現在のフレームと前のフレームを結合して、それらがジッタリングされた量の逆数分だけオフセットします。そのため、最終画像は 2 倍のピクセル数で構成され、それらは 5 点形と呼ばれるサイコロの 5 のように配置されます。

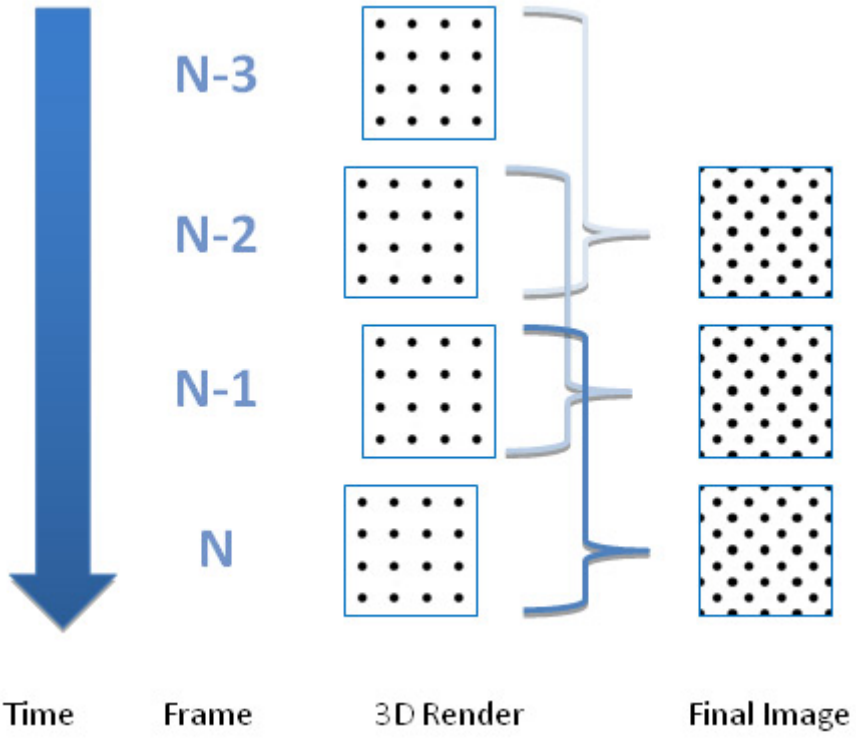

図 5: テンポラル・アンチエイリアシングの基本原則

ダイナミック解像度とともに使用することで、このアプローチではダイナミック解像度のほうがバックバッファーよりも低い場合、シーン内のピクセル数が増加しシーンの詳細が向上します。ダイナミック解像度がバックバッファーと同じかそれ以上の場合、結果はアンチエイリアシングになります。

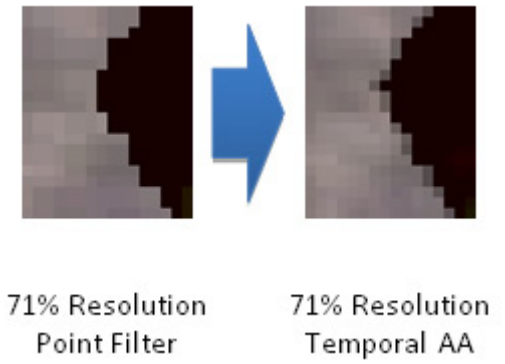

図 6: ダイナミック解像度がバックバッファーよりも低い場合のテンポラル・アンチエイリアシングの結果

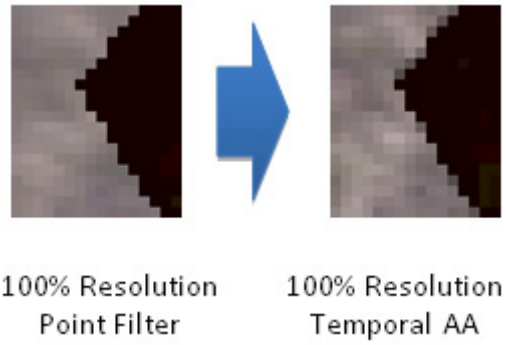

図 7: ダイナミック解像度がバックバッファーと同じかそれ以上の場合のテンポラル・アンチエイリアシングの結果

テクスチャー解像度を上げるには、テクスチャーに MIP LOD バイアスを適用する必要があります。Microsoft* Direct3D* 11 では、3D シーンのパス中に -0.5f の D3D11_SAMPLER_DESC MipLODBias を使用します。また、スケーリング中に使用されるサンプラーは、バイリニア縮小フィルターを使用する必要があります。例: D3D11_FILTER_MIN_LINEAR_MAG_MIP_POINT。

ゴーストを軽減するため、モーションブラー向けに書き出された速度バッファーを使用します。ここで重要なことは、このバッファーには画面空間の各ピクセルの速度が含まれているため、カメラの動きを把握できます。スケール係数は現在のフレームと前のフレームの速度から計算され、最終画像への関連を決定するため前のフレームの色に適用されます。これにより、両方のフレームの実空間内でサンプル位置がどれぐらい類似しているかに基づいて関連性をスケールします。

$$S = \frac{1}{1 + K \times (\mathbf{V}_n \cdot \mathbf{V}_n + \mathbf{V}_{n-1} \cdot \mathbf{V}_{n-1})}$$

$$\mathbf{C}' = \frac{(\mathbf{C}_n + S \times \mathbf{C}_{n-1})}{(1 + S)}$$

C' は最終カラー出力
$\mathbf{C}_n$ は現在のカラーバッファー、$\mathbf{C}_{n-1}$ は前のカラーバッファー
$\mathbf{V}_n$ は現在の速度バッファー、$\mathbf{V}_{n-1}$ は前の速度バッファー
定数 K のサイズは通常約 1/幅

このサンプルは、現実的に再生可能なフレームレートでゴーストが発生することなく、作成者がリアルタイム・アプリケーションにとって最適であると考える結果になるように K でチューニングされています。以下のスクリーンショットのようにコントラストの高い領域でわずかにゴーストが発生しますが、これは必要に応じて調整できます。

ゲームにおいて透明性は、速度情報が常にレンダリングされないという問題を引き起こします。この場合、速度で現在使用されているのとほぼ同じ方法で、透明性のフォワード・レンダリング中にアルファチャネルを使用して、貢献度のスケーリングに使用される値を格納できます。

ゴーストを除去する別の方法として、シーン空間の速度を使用して、前のフレームから現在のピクセルの場所をサンプリングします。これは CryENGINE* 3 で採用されている手法で、ゲーム Crysis* 2 で最初に使用されました [Crytek 2010]。LucasArts 社の Dmitry Andreev 氏は、テンポラル・アンチエイリアシングの使用を検討しましたが、彼らのエンジンはダイナミック解像度を使用していたため採用に至りませんでした [Andreev 2011]。サンプルコードが示すように、著者はこれらには互換性があると考えています。

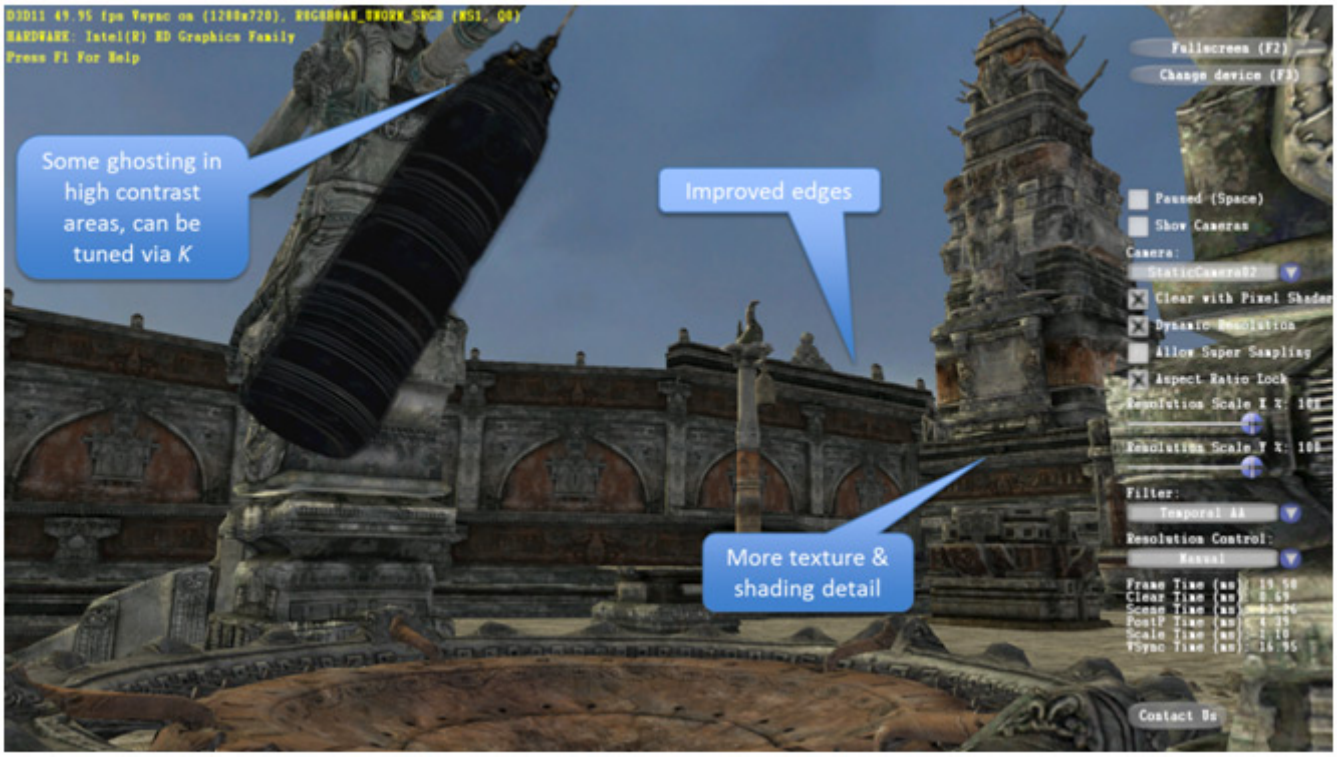

図 8: 速度スケーリングと移動オブジェクトによるテンポラル・アンチエイリアシング

モーションブラーの効果

モーションブラーはピクセルを不鮮明にし、エイリアシングを効果的に軽減するため、カメラが動いているときは低解像度を使用できます。しかし、サンプルの解像度制御アルゴリズムではこれを使用していません。次のスクリーンショットは、解像度をバックバッファーの 0.71 倍に下げて、ほぼ同じ画像で優れたパフォーマンスを達成する方法を示しています。さまざまなモーション・ブラー・サンプル・レートを組み合わせることで、一貫したパフォーマンスを維持しつつ、大きなカメラの動きでアンダーサンプリングからアーティファクトを軽減する方法です。

図 9: ダイナミック解像度がオフの場合のモーションブラー

図 10: 0.71 倍のダイナミック解像度のモーションブラー
(フレーム時間は短縮されていますが、最終結果の品質は同じぐらいです。)

スーパーサンプリング

スーパーサンプリングは単純な手法で、シーンのレンダリングに使用されるレンダーターゲットのほうがバックバッファーよりも大きくなります。この手法は、現在のリアルタイム・レンダリング・コミュニティーではほとんど注目されておらず、より優れたメモリー消費とパフォーマンスを提供するマルチサンプルのアンチエイリアシングとその他のアンチエイリアシング手法によって取って代わられました。

ダイナミック解像度を使用すると、実際の解像度を動的に調整できるため、スーパーサンプリングの追加によるパフォーマンスへの影響が大幅に軽減されます。スーパーサンプリングを有効してもパフォーマンスへの影響はわずかであり、その大部分は大きなバッファーをクリアするための追加コストによるものです。サンプルコードは、スーパーサンプリングが有効な場合、2 倍の解像度のレンダーターゲットを実装します。ただし、バックバッファーの解像度よりもやや高い解像度で優れた品質の結果が得られるため、メモリーの制約がある場合は小さいレンダーターゲットを使用できます。プロセッサー・グラフィックス・プラットフォームでは、GPU は比較的大きなシステムメモリーにアクセスでき、フルパフォーマンスではそれをすべて利用できるため、メモリーはあまり問題になりません。

ダイナミック解像度レンダリング手法を統合すれば、スーパーサンプリングは簡単に使用できます。小さなスクリーンサイズや将来のハードウェアに最高の品質を超えてゲームを実行するのに十分なパフォーマンスをもたらす可能性があるため、スーパーサンプリングの使用を検討することを推奨します。

レンダーターゲットのクリア

ダイナミック解像度レンダリングは常にレンダーターゲットのサーフェス全体を使用するわけではないため、必要な部分のみクリアするとよいでしょう。サンプルコードはピクセル・シェーダー・クリアを実装しており、テストしたインテル® HD グラフィックス 3000 ベースのシステムでは、1280x720 バックバッファーに対してダイナミック解像度が 0.71 倍の場合、ピクセル・シェーダー・クリアのほうが標準のクリアよりも優れたパフォーマンスを達成しました。多くのケースでは、レンダーターゲットが各フレームで完全に上書きされるため、クリアする必要はないかもしれません。

階層的なデプスが実装される可能性があるため、デプスバッファーは標準のクリアメソッドを使用して完全にクリアすべきです。マルチサンプル・レンダー・ターゲットは、圧縮を使用している可能性があるため、通常はクリアする必要があります。

パフォーマンス・スケーリング

サンプルコードは、詳細レベルがない大規模で非常に詳細なシーンと非常に単純なカリングのみが実行されるため、バーテックス処理の負荷が大きいにもかかわらず、解像度に応じて良好にスケーリングします。これは、選択された制御メソッドがフレームレートを任意のレベルに維持するのに大いに役立ちます。

ほとんどのゲームは、詳細レベルメカニズムを使用してバーテックスの負荷を制御しています。これがピクセル単位のオブジェクトのおおよそのサイズと関連付けられている場合、優れたパフォーマンス・スケーリングが得られます。

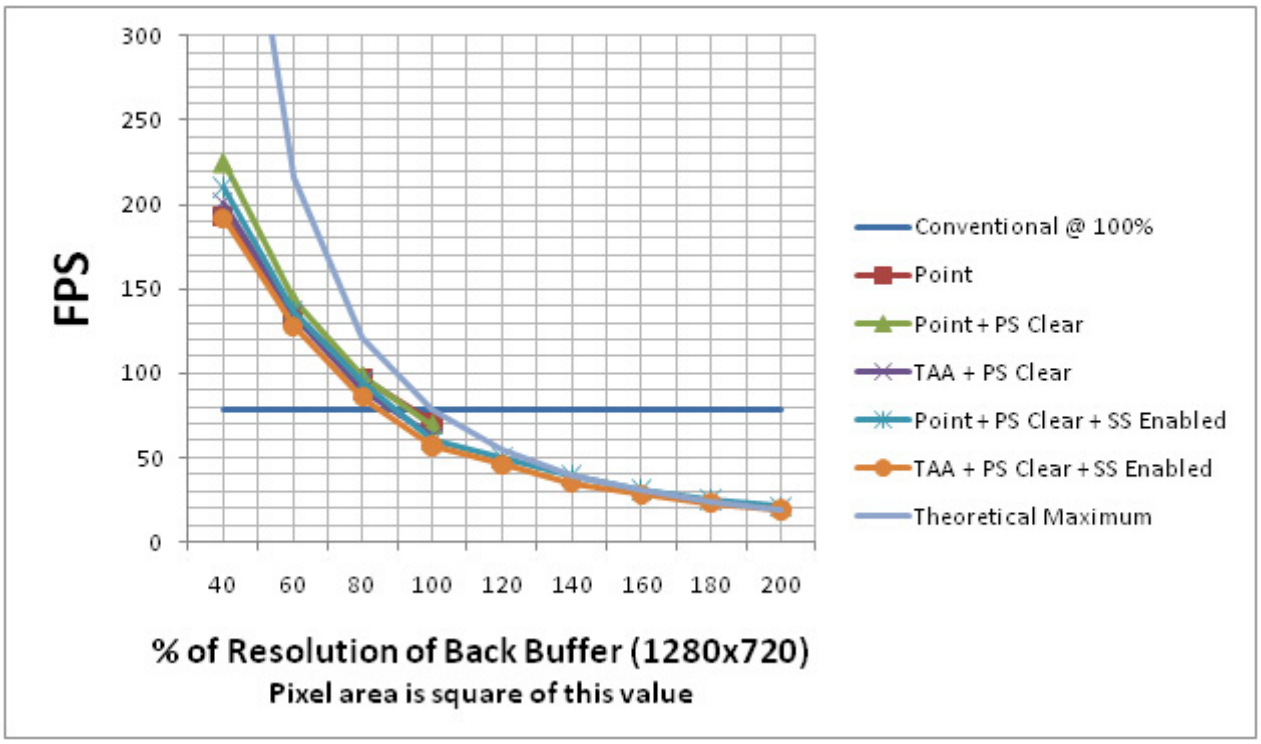

図 11: 1280x720 **でのダイナミック解像度のパフォーマンス**

解像度の制御

サンプルコードでは、手動による制御に加えて、解像度の制御メソッドを実装しています。このコードは、ファイル DynamicResolutionRendering.cpp **の関数** ControlResolution **にあります。リフレッシュ・レート (通常** 60Hz **または** 60FPS) **とリフレッシュ・レートの半分 (通常** 30FPS) **の間で最適なパフォーマンスを選択できます。**

この制御スキームは基本的なものであり、解像度スケールデルタは (選択された) 任意のフレーム時間と現在のフレーム時間の無次元差に比例して計算されます。

$$\Delta s = k \times S \times \frac{(T - t)}{T}$$

$$S' = S + \Delta s$$

S' **は新しい解像度スケール比、** *S* **は現在の解像度スケール比、** Δs **はスケールデルタ、** *k* **は変化率定数、** *T* **は任意のフレーム時間、** *t* **は現在のフレーム時間を表します。**

現在のフレーム時間は、Microsoft* DirectX* **のクエリーを使用して計算された現在を除く** GPU **内部フレーム時間と、通常のフレーム間の間隔から計算されたフレーム時間の平均を使用します。垂直同期が有効な場合、フレーム時間は同期速度により制限されますが、実際のレンダリング時間がそれよりも短いか把握するため、**GPU **内部フレーム時間が必要になります。実際のフレーム速度で平均化することで、現在と一部の** CPU **フレームの負荷を考慮に入れます。実際のフレーム時間が** GPU **内部フレーム時間を大きく上回る場合は無視されます。これらは通常、ウィンドウを最大化するなどの** CPU **側のスパイクが原因です。**

改善の可能性

次のリストは完全なものではなく、著者が現在の取り組みを拡張しようと考えるいくつかの機能にすぎません。

- ダイナミック解像度シーン・レンダリングとシャドウマップ向けの同様のメソッドを結合します。
- この手法を別の粒子系の制御メカニズムと併用することで、少量の小さな粒子のみをレンダリングする場合に品質が向上し、フィルレートが上がるとパフォーマンスが向上します。
- この手法に、テンポラル・アンチエイリアシングとともに適用可能なほかのアンチエイリアシング手法との互換性を持たせます。
- テンポラル・アンチエイリアシングで、合計ブレンドの代わりに、現在のフレームと前のフレームのピクセルの中心への距離に応じた加重和を使用します。CryENGINE* 3 [Crytek 2010] で使用されているような速度に依存するオフセット読み取りを使用するのもよいでしょう。
- ゲームによっては、メイン・キャラクターやマウスでハイライト表示された RTS ユニットなど、画像の小さい領域で高品質のアンチエイリアシング手法を実行すると効果的な場合があります。

まとめ

ダイナミック解像度レンダリングは、特にテンポラル・アンチエイリアシングと組み合わせた場合、最小限のユーザー介入で全体的な品質の向上に必要なツールを開発者に提供します。PC GPU によってパフォーマンスが大きく異なるため、開発者はゲームに適したフレームレートを達成する手段の 1 つとして、この手法を使用することを推奨します。

参考文献 (英語)

[Sigg 2005] Christian Sigg, Martin Hadwiger, "Fast Third Order Filtering", GPU Gems 2. Addison-Wesley, 2005.

[Crytek 2010] HPG 2010 "Future graphics in games", Cevat Yerli & Anton Kaplanyan. https://www.crytek.com/cryengine/presentations

[GDC Vault 2011] https://www.gdcvault.com/play/1014646/-SPONSORED-Dynamic-Resolution-Rendering

[Intel GDC 2011] https://software.intel.com/en-us/event/gdc2011

[Andreev 2011] https://www.gdcvault.com/play/1014550/Anti-aliasing-from-a-Different [PPT 4.6MB]

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。