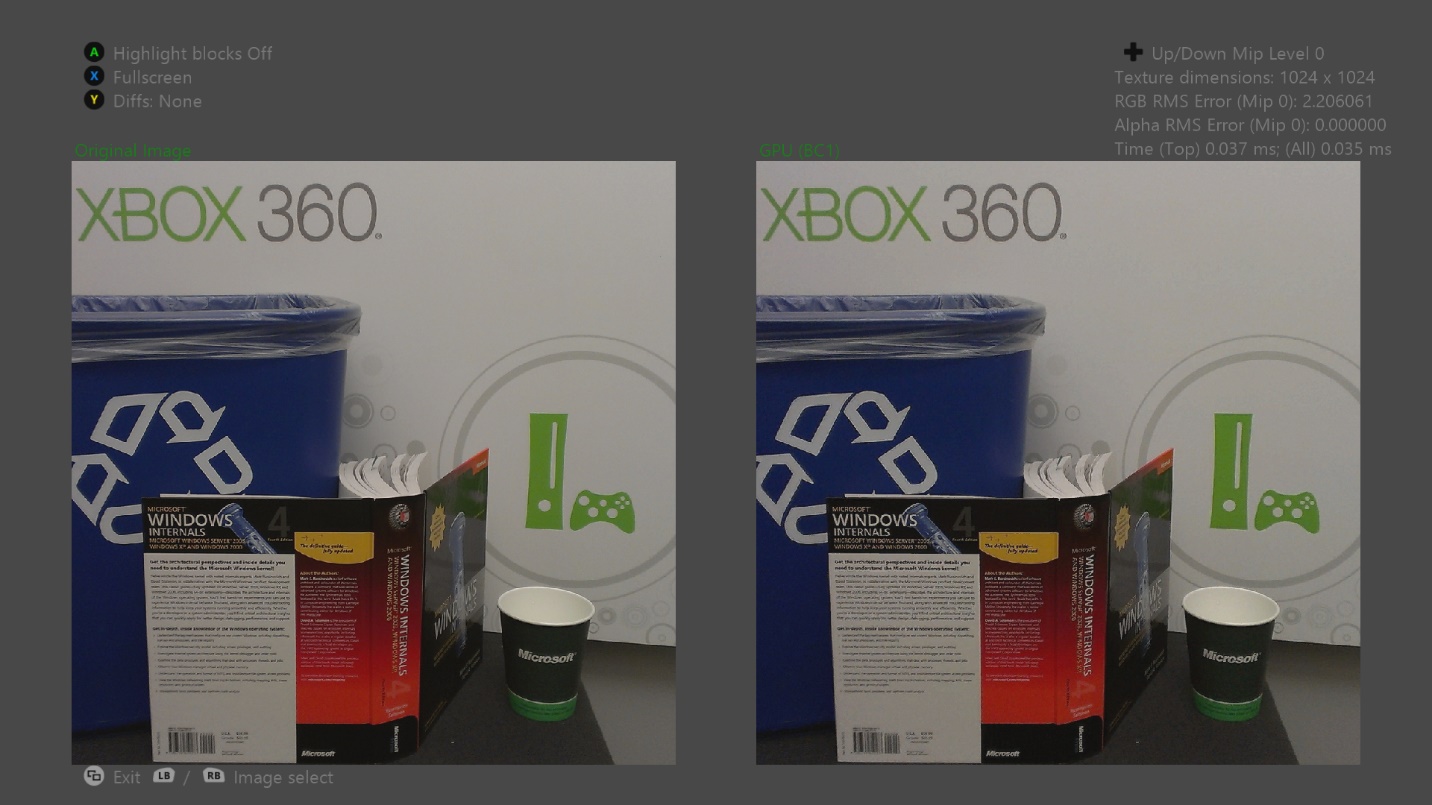


FastBlockCompress のサンプル

*このサンプルには、2017 年 3 月の XDK との互換性があります。*

# 概要

# このサンプルでは、DirectCompute を使用して、実行時にクラシックな*高速ブロック圧縮*アルゴリズムに基づいて、BC1、BC3、および BC5 の形式への高速テクスチャ圧縮を実行する方法を紹介します。また、このサンプルでは、実行時の圧縮モードとオフラインの圧縮モードを切り替えて、表示品質を比較することもできます。



# サンプルの使用

|  |  |
| --- | --- |
| 作用 | ゲームパッド |
| 前後の画像 | 左/右のバンパー |
| 前後の圧縮方法 | D-Pad 左右 |
| 前後の MIP レベル | D-Pad 上下 |
| カメラを動かす | 右スティック |
| ズームインまたはズームアウト | 左/右のトリガー |
| 全画面表示と左右に並べて表示 | X |
| ブロックを強調表示 | A |
| サイクル差分モード | Y |
| 終了 | 表示ボタン |

# バックグラウンド

Xbox One には、専用アプリ用の 5 GB の統合メモリを用意しています。これは、Xbox 360 の 512 MB の 10 倍という劇的な増加です。残念ながら、IO 帯域幅とストレージ メディアの容量については対応が遅れています。Blu-ray メディアの容量は 49 GB で、これは Xbox 360 ゲーム ディスク バージョン 3 で利用可能な 7.8 GB の 6.3 倍に過ぎません。

この事実は、ストリーミング インストールの導入と合わせて、読み込み時間を最小限にして、ゲーム資産を利用可能な記憶領域に入れるために、効率的な圧縮方法が依然として重要であることを意味します。

場合によっては、オフラインの画像圧縮形式を使用してゲーム テクスチャをエンコードすることによって、大量の記憶領域を節約できます。Xbox One にはハードウェア JPEG デコーダが内蔵されているため、JPEG は魅力的な選択肢です。ただし、JPEG ハードウェアではテクスチャがメモリ内の非圧縮 YUV 形式にデコードされます。これはレンダリングに最適とは言えません。この方法を使用する場合、実行時にテクスチャを GPU でサポートされるブロック圧縮形式のいずれかに再圧縮する必要があります。

このサンプルでは、GPU を使用してテクスチャを BC1、BC3、および BC5 の形式に効率的に圧縮します。従来、オフライン ブロック圧縮に使用されてきた標準的なアルゴリズムはリアルタイムで実行するには遅すぎました。このサンプルで使用するアルゴリズムでは、速度を重要視するために品質が大幅に犠牲になります。

現在のアルゴリズムではメモリ帯域幅がボトルネックになっているため、他の手法を利用すると、わずかにパフォーマンスが低下することで著しく品質が向上する可能性があります。

# 実装上の注意

# サンプル内の各 DirectCompute 圧縮シェーダーには、1 MIP バージョン、2 MIP バージョン、テール MIP バージョンの 3 つがあります：

# 1 MIP シェーダーでは、ソース テクスチャの単一 MIP を圧縮します。

# 2 MIP シェーダーでは、ソース テクスチャの単一 MIP を読み取り、その MIP をローカル データ ストア (LDS) メモリにダウンサンプリングします。次に、シェーダーで元のバージョンとダウンサンプリングされたバージョンの両方が圧縮され、対応する MIP レベルが出力テクスチャに書き込まれます。

# このプロセスでは、2 番目の MIP レベルのソース テクスチャからの読み込みを回避することで、メモリ帯域幅を節約します。ただし、実際のパフォーマンスの向上は、追加されたシェーダーの複雑さと、GPR と LDS の使用量の増加による占有率の減少によって大きく相殺されます。

# テール MIP シェーダーは、さまざまな MIP レベルで作業するために、さまざまなスレッドを選択することによって、1 回のディスパッチ呼び出しで 16×16～1×1 の MIP レベルのソース テクスチャを圧縮します。

# 最小の波面サイズは 64 スレッドであるため、個別のディスパッチ呼び出しで各テール MIP を圧縮する手法では、使用可能なスレッドのほとんどが無駄になります。1 つの波面とディスパッチ呼び出しだけを使用することで、テール MIP シェーダーではこの無駄な作業の大部分を回避します。

# Direct3D では BC 形式のテクスチャを UAV としてバインドすることは許可されていないため、計算シェーダーから直接ブロック圧縮テクスチャに書き込むことはできません。このサンプルでは、書き込み可能な形式の中間テクスチャをブロック圧縮テクスチャと同じメモリ ロケーションにエイリアシングすることで、この制限を回避します。中間テクスチャは 4 分の 1 のサイズで、各テクセルは圧縮テクスチャのブロックに対応しています。

# この方法でテクスチャ メモリをエイリアシングするには、2 つのテクスチャの並べて表示モードとメモリ レイアウトを正確に一致させる必要があります。さらに、Direct3D ではメモリ エイリアシングが認識されないため、GPU で同じメモリ ロケーションにエイリアシングされたさまざまなリソース上で動作する複数の描画またはディスパッチ呼び出しが同時にスケジュールされることがあります。

# 言い換えれば、中間テクスチャに書き込むシェーダーは、エイリアス ブロック圧縮テクスチャから読み取る描画呼び出しと同時にスケジュールされることがあります。これらの危険を回避するには、適切なフェンスを手動で挿入する必要があります。

オフライン圧縮アルゴリズムは、[DirectXTex](https://github.com/Microsoft/DirectXTex/) で実装されています。

# 参照

Microsoft Advanced Technology Group。高速ブロック圧縮のサンプル。Xbox 360 SDK。2010 年 2 月。

Tranchida Jason。[Texture Compression in Real-Time Using the GPU](http://www.gdcvault.com/play/1012554/Texture-compression-in-real-time)。GDC 2010。2010 年 3 月。

van Waveren, J.M.P[。Real-Time DXT Compression](https://software.intel.com/sites/default/files/23/1d/324337_324337.pdf)。Intel Software Network。2006 年 5 月

van Waveren, J.M.P と Castaño, Ignacio。[Real-Time YCoCg-DXT Compression](https://www.nvidia.com/object/real-time-ycocg-dxt-compression.html)。NVIDIA 開発者向けサイト。2007 年 9 月。

van Waveren, J.M.P と Castaño, Ignacio。[Real-Time Normal Map DXT Compression](http://developer.download.nvidia.com/whitepapers/2008/real-time-normal-map-dxt-compression.pdf)。NVIDIA 開発者向けサイト。2008 年 2 月。

# 更新履歴

初版は 2015 年にリリースされました。これは 2018 年 12 月に DirectX 12 を使用して改訂されました。

# プライバシーステートメント

サンプルをコンパイルして実行すると、サンプルの実行可能ファイルの名前がMicrosoftに送信され、サンプルの使用状況の追跡に役立ちます。このデータ収集を無効にするには、Main.cppの「Sample Usage Telemetry」というラベルの付いたコードブロックを削除します。

マイクロソフトのプライバシーポリシー全般に関する詳細については、 [Microsoftのプライバシーステートメント](https://privacy.microsoft.com/ja-jp/privacystatement/)をご参照ください。