Depth Buffer Compression for Stochastic Motion Blur Rasterization

Magnus Andersson\textsuperscript{1, 2} \hspace{1cm} Jon Hasselgren\textsuperscript{1} \hspace{1cm} Tomas Akenine-Möller\textsuperscript{1, 2}

\textsuperscript{1} Intel Corporation \hspace{1cm} \textsuperscript{2} Lund University
Motivation

- Depth buffer memory transactions require a significant amount of BW
- Reduced with caching...
- ...and with compression
- **Adding stochastically sampled motion blur to the mix**
  - Doesn’t work well with existing algorithms
Overview

• Architectural/Compression Frameworks
• Previous Work
• Our Algorithm
• Results
• Conclusions
Architectural Framework

Depth Unit

Tile table cache

Tile cache

Rasterizer & Depth Test

Memory bus

RAM
Architectural Framework

Depth Unit

Tile table cache

Tile cache

RAM

Tile table
- Clear
- Mode
- \(z_{min}\)
- \(z_{max}\)
- ...

Memory bus

Rasterizer & Depth Test

Tile
Architectural Framework

[Morein 2000, Hasselgren and Akenine-Möller 2006]
Compression Framework

Existing compression schemes can be described with the three following steps:

1. **Clustering**
   - Group samples with similar characteristics

2. **Predictor function generation**
   - Find suitable predictors for each cluster that minimizes the error

3. **Residual encoding**
   - Capture the remaining error
Previous Work

*Depth Offset* (DO) compression:

- Uses $z_{\text{min}}$ and $z_{\text{max}}$ of the tile
  - *We assume that these are freely available in the tile table*

Described by Hasselgren & Akenine-Möller [2006]
Previous Work

• Most other compression schemes assumes that $z = z_c / w_c$ is linear over a triangle in screen space;

$$z(x, y) = a + bx + cy$$

• Perfectly valid for static scenes
Previous Work

**Anchor encoding / DDPCM (Differential Differential Pulse Code Modulation)**

- Create a predictor plane from three neighboring pixels
- Store residuals in few bits
- DDPCM can handle two planes originating from different corners
  – *Clustering*

Described by Hasselgren & Akenine-Möller [2006]
Previous Work

Improvements on Anchor encoding / DDPCM:

[Hasselgren and Akenine-Möller 2006]
- Smarter bit distribution
- Better clustering

[Ström et al. 2008]
- Predicts from a larger number of pixels
- Handles floating point buffers
- Variable rate residuals with Rice coding

[Lloyd et al. 2007]
- Targets logarithmic shadow maps
Previous Work

Plane encoding

• Communicates with the rasterizer
  – Input: coverage mask and plane equation
  – Can store many planes in one tile
  – Store compressed in cache

\[
\begin{align*}
0: & a_0x + b_0y + c_0 \\
1: & a_1x + b_1y + c_1 \\
2: & a_2x + b_2y + c_2
\end{align*}
\]

Described by Hasselgren & Akenine-Möller [2006]
Motion Blur Challenges

• Assumptions made by previous work:
  – \( z \) is linear over a triangle in screen space
  – Samples are arranged in a grid

(Note: Neither of these assumptions is made by DO)
Motion Blur Challenges

• Assumptions made by previous work:
  – $z$ is linear over a triangle in screen space
  – Samples are arranged in a grid

Introducing *motion blur*
Motion Blur Challenges

• Assumptions made by previous work:
  — \( z \) is linear over a triangle in screen space
  — Samples are arranged in a grid

Introducing \textit{motion blur}
Our Algorithm

Algorithm steps:

1. Clustering
2. Predictor function generation
3. Residual encoding
Our Algorithm

1. **Clustering**
   - Different depth layers often have different characteristics

Ex. 1
- Camera
- Static background
- Moving object

Ex. 2
- Moving camera
- Cleared background
- Moving/rotating object
Our Algorithm

1. **Clustering**

Clustering is very useful around moving silhouettes

*Normal shaded*  
* = Two layers
Our Algorithm

1. Clustering

Assume that there is at least some separation in depth between layers
Our Algorithm

2. **Predictor function generation**

For each layer we use one of 3 different predictors:

- **Static patch**: \( \text{Patch}(x, y) \)
- **Moving plane**: \( \text{Plane}(x, y, t) \)
- **Moving patch**: \( \text{Patch}(x, y, t) \)

**Goal**: Minimizing error => fewer residual bits

But *which* error do we wish to minimize?
Our Algorithm

2. Predictor function generation

Minimize the *maximum* error of any sample

- Use *minimax* (related to the convex hull)
  - Very expensive

[Houle and Toussaint 1988]
Our Algorithm

2. **Predictor function generation**

We use an approximation of *minimax*

- Simplify the problem by reducing the number of points to a few representatives
- A similar approach is used as a first step in all of our compression modes
Our Algorithm

2. **Predictor function generation**

1. Split the samples into two sub-tiles. Then for each sub-tile:
   A. Find samples with **minimum** and **maximum** $z$ values
   B. Use the mid-points as representative points

2. Use the representative points to solve for the predictor

More details in paper...
Our Algorithm

2. **Predictor function generation**

**Static patch:** \[ z = a + bx + cy + dxy \]

- Not time dependent
- Select 2x2 sub-tiles in xy
Our Algorithm

2. **Predictor function generation**

Moving plane: \( z = a + bx + cy + dt \)

- Time dependent plane
- Select 2x2x2 sub-tiles in xyt
  - Select 4 points that are not coplanar
Our Algorithm

2. **Predictor function generation**

Moving patch:

\[ z = (1 - t) (a_0 + b_0 x + c_0 y + d_0 xy) + t (a_1 + b_1 x + c_1 y + d_1 xy) \]

- Interpolate two patches
- Select 2x2x2 sub-tiles in xyt
  - Create one patch in each 2x2x1-slice
  - Extrapolate to \( t = 0 \) and \( t = 1 \)
  - Predict by interpolating between the two
Our Algorithm

3. Residual encoding

• Calculate the offset coefficient, $a$, so that all errors are positive
• Each sample is given the same number of residual bits
  – I.e. that of the largest remaining error
• We “steal” one bit combination to signal clear instead
  – Use the maximum representable error given residual bit count
Our Algorithm

Selecting the best combination

• Try all predictor combinations and select the one with the lowest total bit count
• We also try to compress with DO
  – Will present results from our algorithm alone, and in combination with DO
Implementation

Tiles are extended in the t-dimension as well

• \( w \times h \times n \)
Implementation

Depth Unit

Tile table cache

Cache size
4kB

Tile cache

Cache size
64kB

Compressor / Decompressor

RAM

Rasterizer & Depth Test

Precision
32b

Fixed

MSAA rates
4 spp
16 spp

Tile sizes (w x h x t)
8x8x4
4x4x4

Bus width
512b

30
Results

Airship & Cannon

Original images courtesy of Unigine

Images are rendered in 1920x1200
Results
Spiders & Stone giant

Original images courtesy of BitSquid

Images are rendered in 1920x1200
Results
Spheres

Original image
Results
Increasing motion

16 MSAA

4x4x4

Depth Offset
Our
Combined

8x8x4

Increasing motion
Compression Ratio

![Comparison of Compression Ratios](comparison_images.png)

- **Uncompressed**
- **Best compression**

**PE**, **DO**, and **Our + DO** compression results are shown for various datasets.
Conclusions

First steps into motion blur depth compression

• Good compression rates are possible on stochastically sampled motion blur buffers

• DO is quite good at handling noisy tiles!
  – Good complement to our algorithm

• Linearly approximating $t$ works quite well
Thank you!
Questions?
Tile header layout

Bit combination

- \( N \) lines = 0 \( \rightarrow \) Uncompressed
- Mode 0 = 00
  - Mode 1 = 00 \( \rightarrow \) Cleared
  - Mode 1 = 01 \( \rightarrow \) Compressed with DO
- Mode 0 = 01, 10, 11
  - Mode 1 = 00 \( \rightarrow \) Layer 1 predictor mode
  - Mode 1 = 01, 10, 11 \( \rightarrow \) Layer 2 predictor mode
Compressed tile layout

**DO**

- Mode 0 & 1: 256 bits
- Mode 2: 512 bits

**Our**

1 Layer

- Predictor coefficients
- Per-sample residuals
- 128 / 256 bits
- k * n bits

2 Layers

- Predictor 0 coefficients
- Predictor 1 coefficients
- Per-sample predictor index
- Per-sample residuals
- 256 / 384 / 512 bits
- n bits
- k * n bits

n: Number of samples per tile
k: Residual bits
Acknowledgements

Thanks to Tobias Persson from BitSquid for letting us use the StoneGiant demo, and to Denis Shergin from Unigine for letting us use images from Heaven 2.0. Tomas Akenine-Möller is a Royal Swedish Academy of Sciences Research Fellow supported by a grant from the Knut and Alice Wallenberg Foundation. In addition, we acknowledge support from the Swedish Foundation for strategic research.
References


