Graphics Hardware (2007)
Timo Aila and Mark Segal (Editors)

Practical logarithmic rasterization for low-error shadow maps

D. Brandon Lloyd[1] Naga K. Govindaraju[2] Steven E. Molnar[3] Dinesh Manocha[1]

[1] University of North Carolina at Chapel Hill
[2] Microsoft Corporation
[3] NVIDIA Corporation

Abstract

Logarithmic shadow maps can deliver the same quality as competing shadow map algorithms with substantially less storage and bandwidth. We show how current GPU architectures can be modified incrementally to support rendering of logarithmic shadow maps at current GPU fill rates. Specifically, we modify the rasterizer to support rendering to a nonuniform grid with the same watertight rasterization properties as current rasterizers. We also describe a depth compression scheme to handle the nonlinear primitives produced by logarithmic rasterization. Our proposed architecture enhancements align with current trends of decreasing cost for on-chip computation relative to off-chip bandwidth and storage. For a modest increase in computation, logarithmic rasterization can greatly reduce shadow map bandwidth and storage costs.

Categories and Subject Descriptors (according to ACM CCS): I.3.3 [Computer Graphics]: Hardware Architecture-Graphics processors, I.3.7 [Computer Graphics]: Color, shading, shadowing, and texture

1. Introduction

Shadows are an important element in interactive 3D applications. Unfortunately, shadow rendering can require considerable computation resources. In some cases shadow rendering can take up to one half of the time to render a frame [Sho03]. The importance of shadows has led vendors to introduce hardware features and optimizations specifically targeted to reduce the cost of shadow rendering [NVI04]. In this paper we present a hardware enhancement to support high-quality shadow map rendering.

Shadows in today's interactive applications are typically rendered using stencil shadow volumes [Cro77] or shadow maps [Wil78]. These algorithms capitalize on the enormous fill rates of current GPUs. For example, the GeForce 8800 generates 192 samples per clock for z-only rendering [NVI06]. At the GPU's clock rate of 575 MHz this amounts to an astounding fill rate of 110.4 billion pixels per second. These fill rates are obtained by using rasterization in conjunction with depth buffer compression. Rasterization makes heavy use of parallelism and exploits a high degree of coherence both in computation and memory access. Compression provides more efficient use of available memory bandwidth. Current GPUs also have high-speed memory interfaces. Nevertheless, memory bandwidth can still be a limiting factor for the performance of shadow volumes and shadow maps.

Shadow maps are an attractive algorithm because they are a flexible, image-based approach and can easily handle complex, dynamic scenes. However, shadow maps must use high resolution to avoid aliasing artifacts. High resolution shadow maps limit performance because of bandwidth bottlenecks and increased contention for limited GPU memory.

In this paper, we build on recent work that highlights the importance of a logarithmic shadow map parameterization for reducing aliasing [WSP04, LTYM06, ZSXL06]. The recently proposed logarithmic perspective shadow map (LogPSM) algorithm [LGQ*07] combines a logarithmic parameterization with a perspective projection. Potentially, LogPSMs could have the same good performance as competing algorithms, while requiring significantly lower bandwidth and storage. The main drawback is that the logarithmic rasterization required to support LogPSMs is not available on current GPUs and simulating it is too slow for interactive applications.

Main results: We propose the following incremental enhancements to graphics hardware to support logarithmic rasterization at fill rates comparable to the linear rasterization on current GPUs:

- **Rasterization on a nonuniform grid:** We extend exist-

GH 2007, San Diego, California, August 04 - 05, 2007
© 2007 ACM 978-1-59593-625-7/07/0008 $ 5.00

ing rasterizers that sample on a uniform regular grid to rasterize on a regular grid with nonuniform spacing in one direction.

- **Generalized polygon offset:** Polygon offset is used with shadow maps to avoid self-shadowing artifacts. The polygon offset on current GPUs is constant over a primitive. For logarithmic rasterization the offset varies linearly in one direction.
- **A new depth compression scheme:** Depth compression is important for reducing memory bandwidth. Existing depth compression techniques exploit the fact that primitives are planar. These techniques fail for logarithmic rasterization, which causes planar primitives to become curved. We present a depth compression scheme that is better suited for logarithmic rasterization, which, in practice, also works well for linear rasterization.

The primary advantage of our proposed enhancements is that they capitalize on existing hardware designs. High performance GPUs are the product of years of careful tuning and optimization. Our low-cost enhancements enable significant improvement over existing shadow map algorithms without compromising previous capabilities. Logarithmic rasterization requires a modest amount of additional computational power and produces significant bandwidth and storage savings. Therefore these enhancements align well with current hardware trends of decreasing cost for on-chip computation and a relatively high cost for off-chip bandwidth.

The rest of the paper is organized as follows. Section 2 describes related work. Section 3 provides an overview of the LogPSM algorithm. Section 4 describes the hardware enhancements required to support logarithmic rasterization. Section 5 presents results and analysis, and concludes with ideas for future work.

2. Related work

The underlying cause of shadow map aliasing is a mismatch between the sample locations used for rendering the image from the eye and the samples used for rendering the shadow map from the light [JMB04, AL04]. Ray tracing avoids this problem completely because eye and light samples coincide exactly. Despite considerable recent progress, interactive ray tracing at high resolution for deformable scenes remains a challenge. Hardware accelerators for ray tracing have been proposed [WSS05], but no mass-market solution is currently available. The irregular z-buffer architecture [JLBM05] can also render a shadow map with samples corresponding to those taken by the eye. However, irregular sampling requires a spatial data structure to identify nearest samples. The spatial data structure increases storage costs, increases memory bandwidth when accessing or updating samples, and decreases the locality of memory accesses. Implementing the algorithm with performance comparable to other shadow map algorithms is challenging.

Several techniques that can be used on today's GPUs reduce aliasing by creating a better match between the light and eye sampling rates. These techniques redistribute shadow map samples either by partitioning the shadow map into a collection of smaller, independent shadow maps adapted to the local sampling rate [TQJN99, FFBG01, Arv04, ZSXL06, LSO07, GW07], or by applying a global warping function [SD02, WSP04, MT04]. The partitioning techniques tend to be slow because of the large number of render passes they require. Warping techniques are fast and fairly simple, but because the warping is performed with projective transformations that poorly approximate the eye sample distribution, they still require high resolution to control the resulting errors.

Shadow volumes [Cro77] can robustly render error-free hard shadows on today's GPUs, but they consume an enormous amount of fill rate and do not scale well with geometric complexity.

3. Logarithmic perspective shadow maps

In this section we provide an overview of the logarithmic perspective shadow map (LogPSM) algorithm and highlight its advantages over existing techniques. We refer the interested reader to [LGQ*07] for a detailed derivation, description, and analysis of the algorithm.

3.1. Algorithm

LogPSMs use a small number of warped shadow maps to produce low-error shadow maps for both point and directional light sources. The LogPSM algorithm proceeds as follows:

1. **Face partitioning and perspective warping.** This step is essentially equivalent to Kozlov's perspective warped cube map [Koz04]. In the camera's post-perspective space, the view frustum becomes a cube. Kozlov fits a shadow map to each face of this cube that faces away from the light. In world space this corresponds to a perspective warping of the shadow maps for the sides of the view frustum. The shadow map is oriented so that its y axis is aligned with the direction along the length of the side face (as in Figure 1).
2. **Balanced resolution allocation.** The available shadow map resolution is allocated across the partitions using an aliasing error metric so that maximum errors of the partitions are equalized and the overall maximum error is minimized.
3. **Shadow map rendering.** Each shadow map is rendered separately. The perspective portion of the LogPSM is applied using a standard 4×4 matrix. The logarithmic part is applied only to the y coordinate of the side faces using the following equation:

$$y' = F(y) = c_0 \log(c_1 y + 1) \tag{1}$$

$$c_0 = \frac{-1}{\log(f/n)}, \quad c_1 = \frac{1-(f/n)}{(f/n)} \tag{2}$$

   where the y coordinate lies in the range $[0,1]$, and n and f are the near and far plane distances of the view frustum.
4. **Image rendering.** During image rendering a fragment program is used to sample from the appropriate shadow map for each fragment.

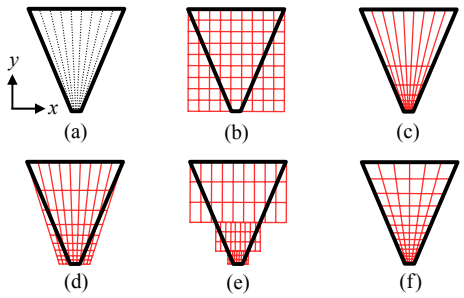

Figure 1: ***Comparison of various parameterizations.*** *(a) In this view from an overhead directional light we see that the spacing between eye samples in the view frustum increases linearly in y. (b) A standard shadow map produces a poor match for the eye sample distribution near the viewer. (c) Perspective projections can obtain a good fit in x at the expense of poor fit in y [SD02], or (d) a better fit in y at the expense of the fit in x [WSP04]. (e) Cascaded shadow maps are a discrete approximation of (f) the continuous logarithmic perspective parameterization.*

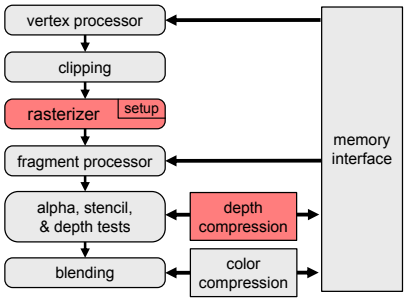

Figure 2: ***Graphics pipeline.*** *We propose modifications to the rasterizer and depth compression units. Note that both compression units perform decompression as well.*

The main differences between the standard shadow maps and the LogPSM algorithm are the parameterization and the use of more than one shadow map.

3.2. Advantages of LogPSMs over other algorithms

Of the many shadow map algorithms, those that achieve high performance on today's hardware are based on perspective warping [SD02, WSP04, MT04, Koz04] and/or are variants of cascaded shadow maps [TQJN99, ZSXL06, Eng07]. Perspective warping algorithms have two main drawbacks that are addressed by LogPSMs. The first drawback is that a perspective parameterization still leads to high error. Analysis by Lloyd et al. [LTYM06] showed that the error for a perspective parameterization is $O(f/n)$. This is an improvement over the $O((f/n)^2)$ error of a standard shadow map, but not nearly as good as the $O(\log(f/n))$ of a logarithmic parameterization. The second drawback is that the single shadow map algorithms [SD02, WSP04, MT04] revert back to standard shadow maps when the light direction approaches the optical axis of the view frustum. This leads to a huge variation in error as the light moves. The face partitioning of Kozlov's algorithm [Koz04] solves this problem but can occasionally cause shearing artifacts on shadow edges. LogPSMs correct for these by splitting side faces that have a large angle between their edges (as seen from the light) and by rotating the perspective projection applied to each half (see [LGQ*07] for more details).

The cascaded shadow map (CSM) approaches partition the view frustum along its optical axis. These partitions are a discrete approximation of a logarithmic parameterization. To approach the same quality as LogPSMs in scenes with a high depth ratio, CSMs require a large number of partitions. For omnidirectional point lights, multiple sets of CSMs are required, leading to even more partitions. Rendering a large number of partitions degrades performance. LogPSMs, on the other hand, require an average of about 4 partitions for directional lights and 6 for point lights inside the view frustum. For the same number of shadow maps as LogPSMs, CSMs have significantly higher error. (See Figure 7 for comparison of LogPSMs with other algorithms).

4. Hardware enhancements for logarithmic rasterization

The perspective part of the LogPSM parameterization can be handled using the standard graphics pipeline. The logarithmic part, however, requires logarithmic rasterization which causes planar primitives to become curved. In this section we describe the incremental hardware enhancements that allow LogPSMs to be rendered with the same fill rates as competing shadow map algorithms, but with the benefits of reduced storage and bandwidth. We modify just two fixed-function hardware components – the rasterizer and the depth compression. After briefly reviewing rasterization based on edge equations, we describe our modifications.

4.1. Rasterizing with edge equations

The rasterizer performs two main functions. First, it determines which pixels are covered by a primitive. Second, it linearly interpolates attributes from the vertices to the covered pixels. Coverage determination on modern, high-performance graphics hardware is typically performed using implicit edge equations of the general form:

$$E(x,y) = Ax + By + C. \tag{3}$$

The sign of E at a point (x,y) indicates on which side of the edge the point lies. Points inside a convex primitive lie on the positive side of all its edges. One of the main advantages of rasterizing with edge equations is that the evaluation of E over many pixels is easy to parallelize. Current GPUs evaluate edge equations for a tile of samples (e.g. a 4×4 block) in parallel. To make the most efficient use of hardware, GPUs perform a hierarchical traversal of the image. A coarse stage identifies tiles that are at least partially covered by a primitive. The tiles are traversed in a manner that is favorable for paged memories and maximizes cache coherence [Pin88, MM00, MWM01]. A fine stage then computes coverage for individual samples within a tile. Attributes for the pixels, such as color, depth, and texture coordinates, can be linearly interpolated from the vertices using an equation of the same form as Equation 3. Olano and Greer [OG97] show

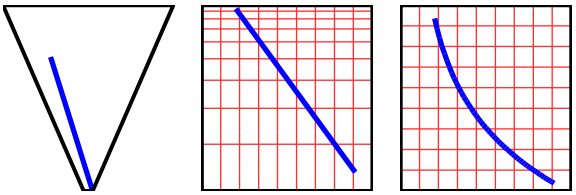

Figure 3: ***Transformation pipeline for LogPSMs.*** *(Left) A world space view of a line (blue) and a side face of the view frustum (black). (Middle) After the perspective transformation the side face has been become a square. (Right) The logarithmic transformation causes the line to become curved. The uniform sample grid (red) in the log warped space corresponds to a nonuniform rectilinear grid in the linear post-perspective space.*

how to compute the coefficients for these equations. The coefficients are computed in the setup phase of the rasterizer.

The edge equations are typically computed in fixed point. The conversion of vertex positions from floating point to fixed point can be thought of as "snapping" the vertices to discrete coordinates on a uniform grid. With discrete coordinates the edge equations can be evaluated at grid locations *exactly* by using a sufficient number of bits to avoid truncation. This provides a water-tight rasterization without the double-hit or missing pixels that can result from numerical robustness problems.

4.2. Logarithmic rasterization

Logarithmic rasterization can be thought of as rasterizing curved primitives on a uniform grid, or as rasterizing linear primitives on a grid that is nonuniform in only one dimension (see Figure 3). We adopt the latter view to facilitate the explanation of how linear rasterization can be extended to support logarithmic rasterization. Our approach is to snap the nonuniform grid samples to positions on an underlying uniform grid that is fine enough to distinctly represent the samples. The edge equations can then be evaluated exactly using fixed-point arithmetic to provide water-tight rasterization.

We will now describe our modified edge equations, how to compute coverage with them, the amount of precision required to evaluate them, and a generalized polygon offset for handling self shadowing artifacts.

4.2.1. Modified edge and interpolation equations

The edge equations for logarithmic rasterization can be computed by transforming y' in the warped space back to y in the linear space using the inverse of Eq. 1 and composing the result with Eq. 3:

$$E'(x,y') = E(x,G(y')) = Ax + BG(y') + C \tag{4}$$

$$G(y') = F^{-1}(y') = \frac{e^{(y'/c_0)} - 1}{c_1} = \frac{(f/n)(1-(f/n)^{-y'})}{(f/n)-1}. \tag{5}$$

The interpolation equations are handled similarly. Note that the coefficients of these equations are based on the vertex positions in linear space resulting from the perspective part

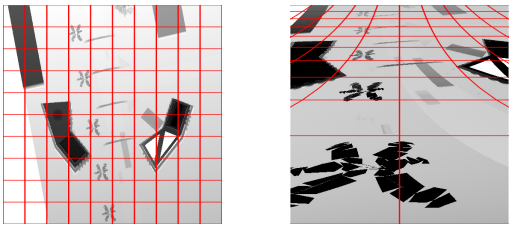

Figure 4: ***Standard and logarithmic shadow maps.*** *A uniform grid is superimposed on a standard shadow map (left) to show the warping from the logarithmic parameterization (right). Note that straight lines become curved*

of the LogPSM parameterization. Rasterization algorithms that initialize tile traversal using vertex positions can apply Equation 1 to the y coordinate of the vertices during triangle setup.

Several algorithms used during rasterization rely on the fact that quantities computed at tile corners provide conservative bounds for the those quantities over the whole tile. The values of the edge equations at the tile corners are used to steer tile traversal. Conservative depth bounds obtained from tile corners are used for culling optimizations such as the hierarchical z-buffer or z-min and z-max culling. The warped edge and interpolation equations remain monotonic so values at tile corners still provide conservative bounds. This means that algorithms that rely on this property can be used for logarithmic rasterization without modification.

4.2.2. Coverage determination

Coverage for a tile sample could be computed brute-force by an array of edge equation evaluators for each sample in the tile. We conserve die area, however, by exploiting the linearity of E' in the x dimension to compute the edge equations incrementally:

$$E'(x_0 + \Delta x, y') = E'(x_0, y') + A\Delta x. \tag{6}$$

We first perform a full evaluation of the edge equations in parallel for the samples in the first column of the tile. We then compute the values for the remaining columns in parallel by simply adding a constant to the first column. These calculations can be pipelined, so that a sustained rate of one 4×4 tile of samples can be computed per clock. A few cycles of downstream buffering allow the coverage bits for each tile to be aggregated together and presented broadside to downstream units. The extra latency incurred by this approach is small compared to the depth of current GPU pipelines. Because the logarithmic rasterization mode is expected to be used only for shadow maps, issues regarding sample placement for multisample antialiasing are not a concern.

Several other possibilities exist for evaluating the edge equations. One possibility is to also compute the $G(y')$ term incrementally along the first column. After a full evaluation of $G(y')$ for the first row, $y' = y'_0$, the values on subsequent rows k could be computed in parallel with a multiply-add

operation:

$$G(y'_0 + k\Delta y') = G(y'_0)P_k + Q_k \tag{7}$$

$$P_k = (f/n)^{-k\Delta y'}, \quad Q_k = (f/n)\frac{(1-(f/n)^{-k\Delta y'})}{(f/n)-1}. \tag{8}$$

The constants P_k and Q_k depend only on the parameterization. Computing $G(y')$ for the first column incrementally is somewhat more involved because it requires an extra step and creates extra latency. Another possibility is to precompute the values of $G(y')$ at each scanline and store them in a table. These values depend only on the parameterization and could be used for all primitives. The table could possibly be constructed using floating point hardware already available for vertex/fragment processors. The evaluation of the edge equations for the first column in a tile would then involve only a table look-up and a dot product. Because tiles are traversed in a predefined order, table entries used by a tile could be prefetched. The table would have to be quite large for a high resolution buffer. The size could be reduced somewhat by precomputing $G(y')$ only for the first row of a tile and computing the other rows incrementally.

4.2.3. Precision requirements

To evaluate the edge equations exactly we require fixed-point $G(y')$ values. We must use at least enough fixed-point precision that any two subsequent values of $G(y')$ and $G(y'+\Delta y')$ can be represented distinctly. In other words, the least significant bit should represent a quantity no larger than $\Delta G_{\min} = \min_{y'} |G(y') - G(y'+\Delta y')|$. Given that the range of G is $[0,1]$, the minimum number of bits required is:

$$b_{\min} = \lceil \log_2(1/\Delta G_{\min}) \rceil. \tag{9}$$

Because $\Delta G_{\min}$ depends on the far to near plane depth ratio (f/n), using a fixed number of bits will place constraints on the range of depth ratios that can be rendered accurately. Suppose we desire to rasterize a $2^{b_x} \times 2^{b_y}$ buffer. The distance between uniform samples in $y' \in [0,1]$ is $\Delta y' = 1/2^{b_y} = 2^{-b_y}$. From Figure 3b we can see that the resulting nonuniform spacing in $y = G(y')$ is minimal at $y' = 1$. Thus $\Delta G_{\min} = G(1) - G(1-2^{-b_y})$. Plugging this value into Equation 9 yields an expression that can be bounded fairly tightly for $(f/n) \in [1, 10^5]$ as:

$$b_{\min} < b_y + 0.8\log_2(f/n). \tag{10}$$

The vertices used to setup the edge equations are limited to the 24 bit precision of a 32-bit floating point mantissa. Therefore we choose to represent fixed-point y coordinates and $G(y')$ values with 24 bits. Plugging $b_{\min} = 24$ in Equation 10 we get an expression for the largest (f/n) that can be handled accurately:

$$(f/n)_{\max} = 2^{\frac{(24-b_y)}{0.8}}. \tag{11}$$

For a $4K \times 4K$ buffer $b_y = 12$ and $(f/n)_{\max} \approx 3.3 \times 10^4$. This depth ratio is quite large. For view frusta with even higher depth ratios the side faces may be split to reduce the depth ratios of the individual pieces.

Standard rasterization typically renders to a sub-pixel grid for increased accuracy. Adding sub-pixel resolution in y effectively increases b_y, which would impose further limitations on $(f/n)_{\max}$. However, this may not be necessary because our approach already provides sub-pixel resolution in y. ΔG increases linearly from $\Delta G_{\min}$ at $y = 1$ to $(f/n)\Delta G_{\min}$ at $y = 0$. When the depth ratio is equal to $(f/n)_{\max}$, the regions near the viewer enjoy an ample amount of sub-pixel resolution. While less sub-pixel resolution is available for the distant regions, this may not be a problem because the distant regions tend to be less important. For low depth ratios, sub-pixel resolution is available for both the near and distant regions.

4.2.4. Attribute interpolation

Attribute interpolation uses planar equations of the same form as E'. The interpolation equations can be evaluated in the same way as the edge equations. The only difference is that they need not be evaluated exactly, so floating point or lower precision fixed-point representations may be used. We modify the depth interpolation slightly to handle polygon offset as described in the following section.

4.2.5. Generalized polygon offset

Polygon offset is often used to generate a depth bias used to prevent false self-shadowing. The standard offset is $sm_z + u$, where s is a scale factor, m_z is the depth slope of the polygon, and u is a constant. The OpenGL Specification [SA06] defines an approximation of the depth slope m_z as:

$$m_z = \max\left(\left|\frac{\partial z}{\partial x}\right|, \left|\frac{\partial z}{\partial y}\right|\right). \tag{12}$$

The polygon offset can be computed in setup and folded into the depth interpolation equation. The depth slope is computed in linear space so we compute the value of the warped space $|\partial z \partial y'|$ in linear space using the chain rule:

$$\left|\frac{\partial z}{\partial y'}\right| = \left|\frac{\partial z}{\partial y}\right|\left|\frac{\partial y}{\partial y'}\right| = \left|\frac{\partial z}{\partial y}\right|\left|\frac{c_1 y + 1}{c_0 c_1}\right|. \tag{13}$$

The $|(c_1 y + 1)/(c_0 c_1)|$ term decreases with y. Unlike linear rasterization, m_z may switch from $|\partial z/\partial y'|$ to $|\partial z/\partial x|$ over a polygon. One way to handle this is to use two depth interpolation equations, each combined with the polygon offset for $|\partial z/\partial x|$ and $|\partial z/\partial y'|$, respectively. The max operator could then be performed per pixel. Another possibility is to split the polygon along the scan line where the switch occurs. This can be done by adding one more edge equation. If edge equations are used to implement the viewport or scissor operations, one of these may be used. Each polygon half is rasterized with the appropriate depth interpolation equation. The simplest approach, however, is to compute m_{z0}, the value of m_z at the lower y-bound of the polygon y_0, and m_{z1}, the value of m_z at the upper bound y_1, and use a single equation $m_z(y)$ that varies linearly between them:

$$m_z(y) = \frac{m_{z0}(y1 - y) + m_{z1}(y - y0)}{(y_1 - y_0)} \tag{14}$$

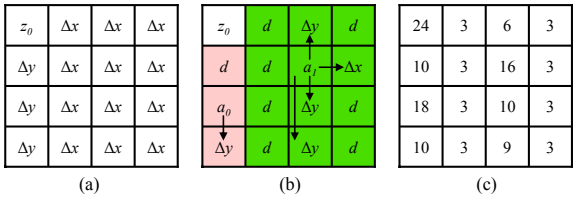

Figure 5: ***Depth buffer compression algorithm for logarithmic rasterization.*** *(a) First-order differentials are computed in y and x by subtracting the depth value of the neighbor immediately above or to the left, respectively. (b) This is followed by a variation of anchor encoding. For the pink region, the anchor a_0 and its Δy offset form a line used as a predictor, and d is a correction term. In the green region a_1, Δx, and Δy values form a predicting plane. A different Δy is used for encoding each row. (c) Bit allocation for each quantity.*

The offset computed using this equation can be folded into a single depth interpolation equation. When no switch occurs on the polygon, Eq. 14 gives the correct result. When there is a switch, it provides a conservative approximation. The maximum relative difference in the approximation is largest when $y_1 - y_0 = 1$ with a value of about 1, but for shorter extents the difference is far less. Note that for paths through our test scenes a switch actually occurs for less than 1% of the polygons. For those polygons with a switch the mean of the maximum relative difference from using Eq. 14 is less than .001.

4.3. Depth buffer compression

Depth buffer compression is an important optimization for reducing the memory bandwidth requirements of rasterization. We refer the reader to Hasselgren and Möller [HA06] for an excellent survey and detailed explanations of existing depth compression schemes. Many of the current algorithms exploit the planarity of a tile's depth values to achieve fast, lossless compression. Because logarithmic rasterization causes planar primitives to become curved, many of the existing algorithms fail to achieve good compression ratios.

We compose two existing first-order schemes to produce a second order compression scheme that is better able to capture the curvature in the y direction. The algorithm is summarized in Figure 5. First, we store the base value in the upper-left corner of the tile at full 24-bit precision. We compute first order differentials for the remaining values [DMFW02]. For the green 4×3 block on the right, the differentials should be fairly small since the edge equations are linear in x. We then use a variation of an anchor encoding scheme [VM05]. Delta offsets are computed with respect to anchor values. Together the delta offsets and anchor values form a line or plane that is used to predict values for the rest of the locations. Correction terms for the predicted values are also computed. If the values of the anchors, delta offsets, and correction terms all fit within their allotted bit budget the tile can be compressed. Otherwise, it is stored uncompressed. Decompression simply reverses the process. Our scheme uses a 128-bit allocation to achieve lossless compression with a 3 : 1 compression ratio. The results are shown in the next section.

4.4. Summary

To summarize, logarithmic rasterization requires the following modifications:

- A modest increase in bit width in the rasterizer to support the 24-bit fixed-point y coordinates.
- Evaluators to compute the exponentials in the edge and interpolation equations.
- Application of the logarithmic parameterization to y coordinates in setup.
- Computation of generalized polygon offset in setup.
- A new depth compression unit.

The rest of the graphics pipeline remains the same.

5. Results

In this section we describe our logarithmic rasterization simulator and compare the results of LogPSMs with several competing approaches. We also demonstrate the effectiveness of our depth compression and discuss the overall feasibility of our approach.

We have implemented a simulator that uses a fragment program to perform logarithmic rasterization. The simulator performs the logarithmic transformation on triangle vertices and renders the axis-aligned bounding quad that encloses them. The fragment program then evaluates the triangle edge equations and discards fragments that fall outside the triangle. It computes depth for the remaining fragments using the depth interpolation equation (which includes the generalized polygon offset). This simulator runs at about 2–3 frames per second on a GeForce 6800 GT on our test models – far below what a native hardware implementation could do, but fast enough to be interactive.

We have also implemented several competing shadow map algorithms on the same GPU. Figure 6 shows a comparison of LogPSMs with each of these other algorithms on a town scene. As the figure shows, for the same shadow map resolution LogPSMs deliver lower, more uniform error than competing algorithms. (See [LGQ*07] for a more extensive comparison with other algorithms).

5.1. Depth compression

We evaluated our depth compression scheme in a way similar to that of Hasselgren and Möller [HA06]. We captured data for camera paths through three models of varying complexity using 4 different resolutions. We also used some of the same compression algorithms [VM05, DMFW02, OPS*05, HA06]. We could not test the depth offset algorithm because this requires the z-min and z-max values, which are not supplied by our simulator. For each frame we split the finished depth buffer into 4×4 tiles (excluding tiles untouched during scene rasterization) and ran them through the compression algorithms. A more detailed simulation would capture the bandwidth of partially completed tiles. The results shown in Figure 6 are intended to indicate the relative performance of the algorithms.

On depth buffers rendered with linear rasterization, our compression scheme is on par with some of the other algorithms. For logarithmic rasterization the other algorithms

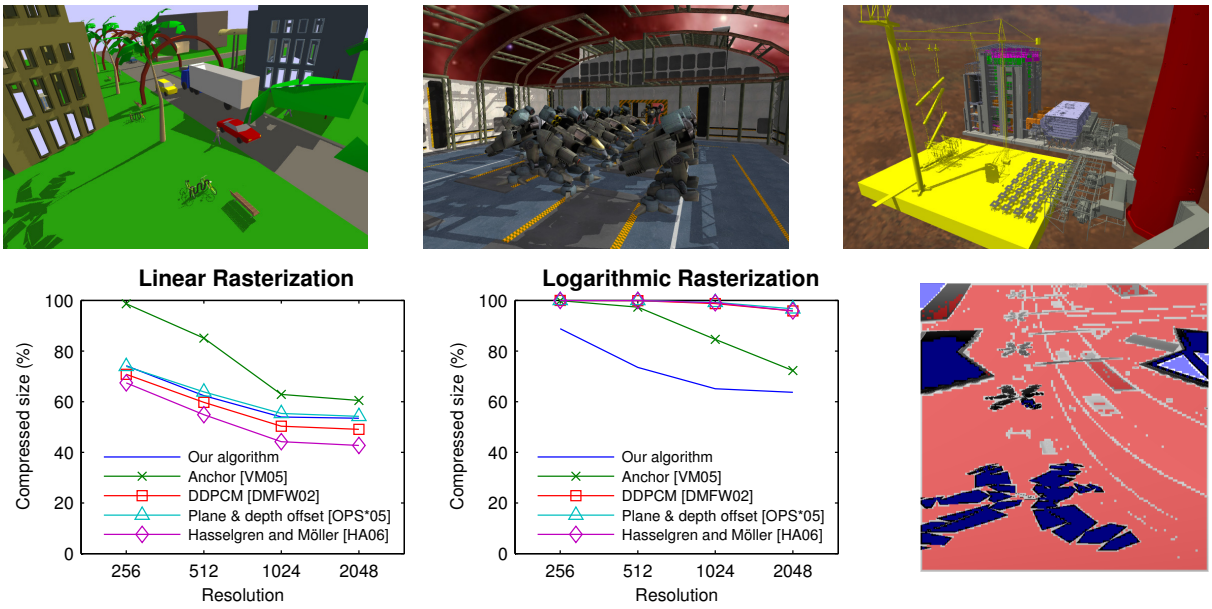

Figure 6: ***Depth compression*** *(Top row) Benchmark scenes: Town (58K triangles), Robots (95K triangles), and Power Plant (1.7M triangles) The full Power Plant has 13M triangles, but we left out internal pipes. (Bottom row) The two graphs show the average depth compression for several algorithms for shadow maps rendered with both linear and logarithmic rasterization. Our depth compression algorithm provides reasonable compression for both. On the right is the logarithmic shadow map from Figure 4 color coded with red for compressed tiles and blue for tiles where the sample depths are all 1 because they are untouched or all 0 due to depth clamping.*

fare poorly. Our compression scheme is still able to achieve fairly good compression ratios, especially at higher resolutions, although the compression ratios are somewhat lower than those for linear rasterization.

5.2. Limitations

One of the limitations of our approach to logarithmic rasterization is the limit on the depth ratio. Both LogPSMs and current techniques such as cascaded shadow maps handle high depth ratios by partitioning the view frustum and using more shadow maps. Cascaded shadow maps must split to control the error. LogPSMs split due to precision limitations. Fortunately, LogPSMs require few splits. For example, a single LogPSM can accurately render shadows on receivers from 1 m to 33 km away. A single split squares this range to over 1000 km. Another limitation is that like all other global warping algorithms, LogPSMs do not take the orientation of surfaces in the scene into account and may not perform as well when the surfaces in the scene are nearly parallel to the light direction.

5.3. Feasibility

We believe that the enhancements we propose to support logarithmic rasterization are feasible because they are incremental and leverage existing hardware designs. The modifications are isolated to only two fixed-function elements of the GPU pipeline. The changes to the rasterizer are incremental. A new depth compression unit using our algorithm would have complexity comparable to other algorithms. If desired, the depth compression unit could sit alongside the existing one used for linear rasterization. Because our algorithm also provides comparable compression ratios for linear rasterization, our depth compression unit could feasibly replace the existing unit, thereby enabling a single unit to handle both linear and logarithmic rasterization.

Our enhancements require only a modest increase in on-chip calculations but deliver significant reductions in bandwidth and storage requirements. Using increased computation to save bandwidth aligns well with current hardware trends because for the same cost, computational power continues to increase rapidly while bandwidth lags considerably behind. Therefore, our proposed hardware enhancements provide a good balance between bandwidth reduction and implementation cost.

Conclusion

We have shown how logarithmic rasterization can be implemented on GPUs by extending existing hardware to rasterize to a nonuniform grid. We have also demonstrated a depth compression scheme that produces reasonable results for logarithmic rasterization. Using a simulator, we have also shown that logarithmic rasterization can produce shadow maps with lower error than competing algorithms.

For future work we would like to investigate further improvements to our depth compression scheme. We would also like to explore the topic of user programmable rasterization. LogPSMs are one example of the benefits a programmable rasterizer could provide. Rasterization is an obvious next step for increasing user programmability of the graphics pipeline.

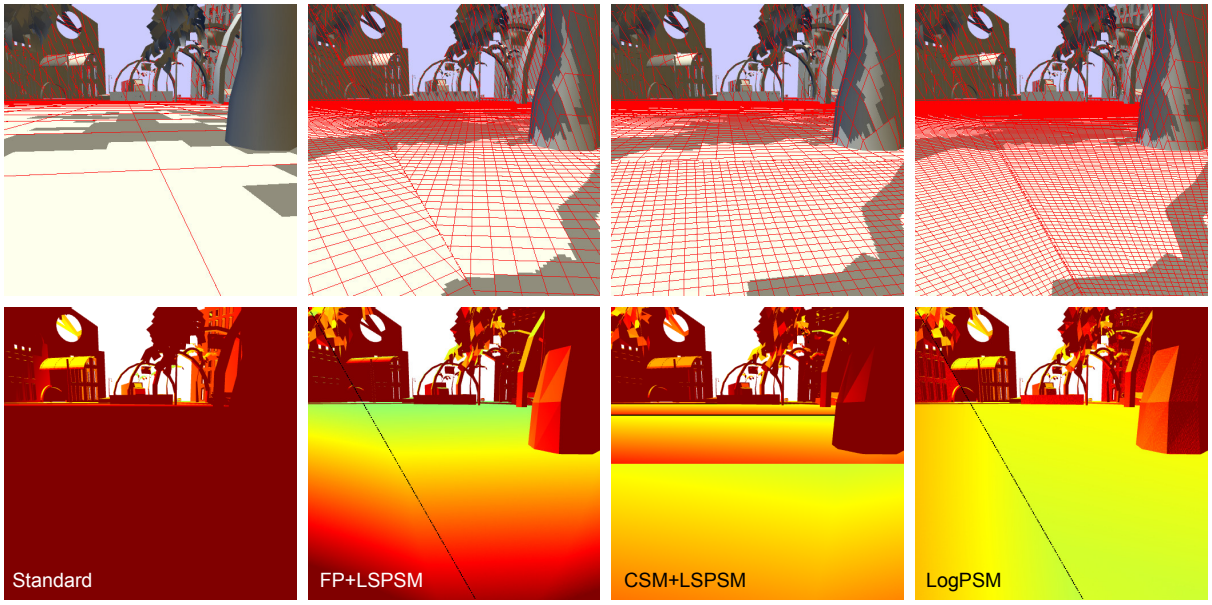

Figure 7: ***Benefits of logarithmic rasterization.*** *The viewer is positioned below a tree in a town scene. Both the image and* total *shadow map resolution are* 512×512. $(f/n) = 1000$. *(Top row) Grid lines for every 5 texels projected onto the scene. (Bottom row) Color encoding of aliasing error m: green* ($m = 1$)*, yellow* ($m = 3$)*, red* ($m > 10$)*. m measures the maximum extent of the projection of shadow map texels in the image. Black pixels occur at partition boundaries. Standard shadow map have extremely high error.* $FP + LSPSM$ *combines face partitioning with perspective warping using the light-space perspective shadow maps (LSPSM) [WSP04] algorithm. The cascade shadow map (CSM) uses perspective warping (LSPSM) on the individual shadow maps for additional error reductions. LogPSMs use logarithmic rasterization to deliver low error both close to the viewer and far away. 5 shadow maps are used for* $CSM + LSPSM$ *while only 3 are used for* $FP + LSPSM$ *and LogPSM.*

Acknowledgements

We would like to thank Cory Quammen for help with the video, Ben Cloward for the robot model, Aaron Lefohn and Taylor Holliday for the use of the town model, and the anonymous reviewers for their helpful comments. Special thanks go to Jon Hasselgren and Thomas Akenine-Möller for the use of their depth compression code. This work was supported in part by an NSF Graduate Fellowship, an NVIDIA Fellowship, ARO Contracts DAAD19-02-1-0390 and W911NF-04-1-0088, NSF awards 0400134, 0429583 and 0404088, DARPA/RDECOM Contract N61339-04-C-0043 and the Disruptive Technology Office.

References

[AL04] AILA T., LAINE S.: Alias-free shadow maps. In *Proceedings of Eurographics Symposium on Rendering 2004* (2004), Eurographics Association, pp. 161–166. 2

[Arv04] ARVO J.: Tiled shadow maps. In *Computer Graphics International* (2004), pp. 240–247. 2

[Cro77] CROW F. C.: Shadow algorithms for computer graphics. *ACM Computer Graphics 11*, 3 (1977), 242–248. 1, 2

[DMFW02] DEROO J., MOREIN S., FAVELA B., WRIGHT M.: Method and apparatus for compressing parameter values for pixels in a display frame. *US Patent 6,476,811* (2002). 6

[Eng07] ENGEL W.: Cascaded shadow maps. In *ShaderX*^{5}, Engel W., (Ed.). Charles River Media, 2007, pp. 197–206. 3

[FFBG01] FERNANDO R., FERNANDEZ S., BALA K., GREENBERG D.: Adaptive shadow maps. In *Proceedings of ACM SIGGRAPH 2001* (2001), pp. 387–390. 2

[GW07] GIEGL M., WIMMER M.: Queried virtual shadow maps. In *Proceedings of ACM SIGGRAPH 2007 Symposium on Interactive 3D Graphics and Games* (2007), ACM Press, pp. 65–72. 2

[HA06] HASSELGREN J., AKENINE-MÖLLER T.: Efficient depth buffer compression. In *Proceedings of Graphics Hardware 2006* (2006), Eurographics Association, pp. 103–110. 6

[JLBM05] JOHNSON G. S., LEE J., BURNS C. A., MARK W. R.: The irregular z-buffer: Hardware acceleration for irregular data structures. *ACM Trans. Graph. 24*, 4 (2005), 1462–1482. 2

[JMB04] JOHNSON G., MARK W., BURNS C.: The irregular z-buffer and its application to shadow mapping. In *The University of Texas at Austin, Department of Computer Sciences. Technical Report TR-04-09* (2004). 2

[Koz04] KOZLOV S.: Perspective shadow maps: Care and feeding. In *GPU Gems*, Fernando R., (Ed.). Addison-Wesley, 2004, pp. 214–244. 2, 3

[LGQ*07] LLOYD D. B., GOVINDARAJU N. K., QUAMMEN C., MOLNAR S., MANOCHA D.: *Logarithmic perspective shadow maps*. Tech. Rep. TR07-005, University of North Carolina at Chapel Hill, 2007. (available at http://gamma.cs.unc.edu/logpsm). 1, 2, 3, 6

[LSO07] LEFOHN A. E., SENGUPTA S., OWENS J.: Resolution-matched shadow maps. *ACM Trans. Graph.* (2007). To appear. 2

[LTYM06] LLOYD B., TUFT D., YOON S., MANOCHA D.: Warping and partitioning for low error shadow maps. In *Proceedings of the Eurographics Symposium on Rendering 2006* (2006), Eurographics Association, pp. 215–226. 1, 3

[MM00] MCCORMACK J., MCNAMARA R.: Tiled polygon traversal using half-plane edge functions. In *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS workshop on Graphics hardware* (2000), pp. 15–21. 3

[MT04] MARTIN T., TAN T.-S.: Anti-aliasing and continuity with trapezoidal shadow maps. In *Proceedings of the Eurographics Symposium on Rendering* (2004), Eurographics Association, pp. 153–160. 2, 3

[MWM01] MCCOOL M., WALES C., MOULE K.: Incremental and hierarchical hilbert order edge equation using polygon rasteization. *Eurographics Workshop on Graphics Hardware* (2001), 65–72. 3

[NVI04] NVIDIA CORP.: UltraShadowII: Accelerated shadow calculations. *Technical Brief* (2004). 1

[NVI06] NVIDIA CORP.: NVIDIA GeForce 8800 GPU architecture overview. *Technical Brief* (2006). 1

[OG97] OLANO M., GREER T.: Triangle scan conversion using 2d homogeneous coordinates. *Eurographics Workshop on Graphics Hardware* (1997), 89–95. 3

[OPS*05] ORNSTEIN D., PELED G., SPERBER Z., COHEN E., MALKA G.: Z-compression mechanism. *US Patent 6,580,427* (2005). 6

[Pin88] PINEDA J.: A parallel algorithm for polygon rasterization. In *Computer Graphics (SIGGRAPH '88 Proceedings)* (Aug. 1988), Dill J., (Ed.), vol. 22, pp. 17–20. 3

[SA06] SEGAL M., AKELEY K.: The OpenGL graphics system: A specification. *http://www.opengl.org/* (2006). 5

[SD02] STAMMINGER M., DRETTAKIS G.: Perspective shadow maps. In *Proceedings of ACM SIGGRAPH 2002* (2002), pp. 557–562. 2, 3

[Sho03] SHOEMAKER B.: John Carmack speaks on doom 3. *GameSpot, http://www.gamespot.com/pc/action/doom3/news.html?sid=6027989* (2003). 1

[TQJN99] TADAMURA K., QIN X., JIAO G., NAKAMAE E.: Rendering optimal solar shadows using plural sunlight depth buffers. In *Computer Graphics International 1999* (1999), p. 166. 2, 3

[VM05] VAN DYKE J., MARGESON J.: Method and apparatus for managing and accessing depth data in a computer graphics system. *US Patent 6,961,057* (2005). 6

[Wil78] WILLIAMS L.: Casting curved shadows on curved surfaces. In *Computer Graphics (SIGGRAPH '78 Proceedings)* (1978), vol. 12, pp. 270–274. 1

[WSP04] WIMMER M., SCHERZER D., PURGATHOFER W.: Light space perspective shadow maps. In *Proceedings of the Eurographics Symposium on Rendering* (2004), Eurographics Association, pp. 143–152. 1, 2, 3, 8

[WSS05] WOOP S., SCHMITTLER J., SLUSALLEK P.: RPU: a programmable ray processing unit for realtime ray tracing. *ACM Trans. Graph. 24*, 3 (2005), 434–444. 2

[ZSXL06] ZHANG F., SUN H., XU L., LUN L. K.: Parallel-split shadow maps for large-scale virtual environments. In *Proceedings of ACM International Conference on Virtual Reality Continuum and Its Applications 2006* (2006), ACM SIGGRAPH, pp. 311–318. 1, 2, 3