Hierarchical Stochastic Motion Blur Rasterization

Jacob Munkberg[1] Petrik Clarberg[1] Jon Hasselgren[1] Robert Toth[1] Masamichi Sugihara[1]
Tomas Akenine-Möller[1,2]

[1]Intel Corporation [2]Lund University

Abstract

We present a hierarchical traversal algorithm for stochastic rasterization of motion blur, which efficiently reduces the number of inside tests needed to resolve spatio-temporal visibility. Our method is based on novel tile against moving primitive tests that also provide temporal bounds for the overlap. The algorithm works entirely in homogeneous coordinates, supports MSAA, facilitates efficient hierarchical spatio-temporal occlusion culling, and handles typical game workloads with widely varying triangle sizes. Furthermore, we use high-quality sampling patterns based on digital nets, and present a novel reordering that allows efficient procedural generation with good anti-aliasing properties. Finally, we evaluate a set of hierarchical motion blur rasterization algorithms in terms of both depth buffer bandwidth, shading efficiency, and arithmetic complexity.

CR Categories: I.3.1 [Computer Graphics]: Hardware architecture—Graphics processors I.3.3 [Computer Graphics]: Picture/Image Generation—Antialiasing I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Hidden line/surface removal

Keywords: stochastic rasterization, motion blur, hierarchical traversal, occlusion culling

1 Introduction

At the heart of every rendering engine there is some form of *visibility* computations. A more advanced algorithm allows effects such as motion blur and depth of field to be rendered by a more elaborate camera model. Depth of field helps to direct the viewer's attention, and motion blur reduces temporal aliasing, so that lower frame rates can be used. Both these effects are also highly desired in the field of real-time graphics.

While an incredible amount of research and engineering effort has been spent on perfecting and fine-tuning the algorithms and the corresponding hardware units for rasterizing static triangles [Fuchs et al. 1989; Pineda 1988; Olano and Greer 1997; McCormack and McNamara 2000; McCool et al. 2001], the same is far from true for rasterization of motion-blurred geometry. However, there has been increased research activity in this field [Cook et al. 1987; Akenine-Möller et al. 2007; Fatahalian et al. 2009; McGuire et al. 2010; Brunhaver et al. 2010], but much remains to be done before the relative efficiency of rasterizing triangles with blur effects is close to that of static triangle rasterization. To be able to add correct motion blur to current and future games, one of our goals is to support efficient rendering of motion blur with mixed sizes of the triangles, i.e., both large triangles and smaller triangles, generated by, e.g., tessellation.

To that end, we present what we believe is the first *hierarchical* rasterization algorithm for motion-blurred triangles. We strive for an algorithm that extends current real-time GPU pipelines, while retaining many of its important features, such as per-tile occlusion culling, mixed triangle sizes, shading after visibility, and multisampling anti-aliasing (MSAA).

Our contributions are:

- ◇ A hierarchical algorithm for motion blurred triangle rasterization, including a low-cost tile vs moving triangle overlap test that returns a conservative time interval of overlap.
- ◇ Modification of an existing hardware-friendly sampling pattern for use in motion blur rasterization with high-quality anti-aliasing. We present an efficient algorithm for computing the samples within a time interval on the fly.
- ◇ Detailed performance evaluation of several different motion blur rasterization algorithms in terms of arithmetic intensity, memory bandwidth usage, and shading efficiency.

We hope that our new algorithms will advance the field of motion blur rasterization so that in the near future, fixed-function rasterization units will have support for such effects.

2 Related Work

Efficient rendering of motion blur has been a long-standing problem in computer graphics. Existing solutions often rely on approximate post-processing based methods or stochastic ray tracing [Cook et al. 1984]. We will not go into detail on these methods, and instead focus on rasterization-based methods for correct motion blur that can be integrated into future hardware GPU pipelines.

A brute-force technique is to draw the scene at N different times and average the result using accumulation buffering [Korein and Badler 1983; Haeberli and Akeley 1990]. The resulting strobing artifacts can be replaced by noise by using stochastic rasterization [Cook et al. 1987]. Here, a bounding box around the blurred triangle is traversed, and all samples are tested against the primitive displaced according to the samples' times. This becomes inefficient when the bounding box is large compared to the primitive. The screen space area of the traversed region can be reduced using either an oriented bounding box (OBB) in 2D homogeneous space [Akenine-Möller et al. 2007], or the convex hull in screen space [McGuire et al. 2010]. Existing (two-dimensional) hierarchical rasterization methods can be leveraged to efficiently traverse these bounds, but all temporal samples still have to be tested. This becomes expensive with large motion. In contrast, our algorithm derives temporal bounds per tile to cull samples.

Fatahalian et al. [2009] improve the situation for stochastic micropolygon rasterization by partitioning the time domain into multiple intervals (initially proposed by Pixar), or by using interleaved sampling [Keller and Heidrich 2001] with a fixed number of sample times. Both methods rasterize the primitive independently for each time/interval, which generates samples in an incoherent order, i.e., sparse in screen space. For a REYES pipeline with shading at the

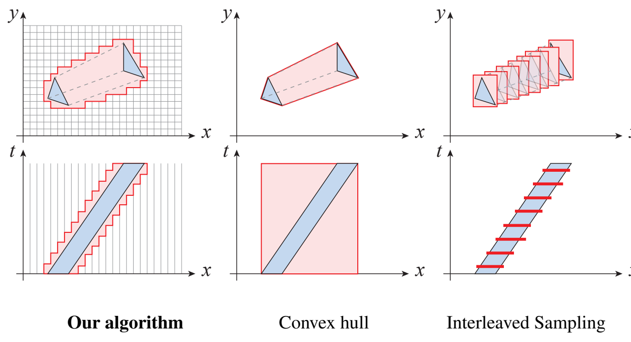

Figure 1: *The three-dimensional sampling space, (x, y, t), traversed with different methods for stochastic motion blur rasterization. Sample-in-triangle inside tests are performed for all samples within the red regions. Our algorithm, based on novel hierarchical tile tests with temporal overlap computations, significantly reduces the amount of inside tests compared to using the convex hull in screen space [McGuire et al. 2010]. With interleaved sampling [Keller and Heidrich 2001] the samples are restricted to a fixed number of pre-defined times.*

vertex level, this is not a problem, but applied to a graphics pipeline with shading at the fragment level, it makes reusing shading over multiple samples (MSAA) difficult. Per-tile occlusion culling also becomes substantially more expensive. Furthermore, each triangle has to be setup multiple times. In contrast, our algorithm uses a coherent screen space traversal order, which facilitates MSAA and efficient occlusion culling. Figure 1 shows the spatio-temporal coverage of each algorithm.

Although a hard problem, analytical determination of visibility has been explored. Most recently, Gribel et al. [2010] presented a method for analytical motion blur rasterization where the samples' temporal overlaps with a moving primitive are analytically determined and stored in linked lists per pixel. Our work is similar in that we analytically determine conservative time bounds, but we do this for entire tiles of pixels and the generated samples are stored in a traditional multi-sampled render target. The use of a tiled traversal with temporal bounds allows us to quickly reject samples.

Hierarchical occlusion culling is critical for achieving good performance in modern GPUs by early determining if a tile is entirely occluded ($z_{\max}$-culling) [Morein 2000] or entirely visible ($z_{\min}$-culling) [Akenine-Möller and Ström 2003]. However, motion blur makes culling using a traditional hierarchical z-buffer [Greene et al. 1993] less efficient. By storing multiple temporal depth values (*tz-slice*) [Akenine-Möller et al. 2007], or a full temporal pyramid of depth values (*tz-pyramid*) [Boulos et al. 2010] per node, efficiency can be improved. Our rasterization order, i.e., one tile at the time, together with conservative temporal bounds, makes the use of these occlusion culling techniques efficient and straightforward.

3 Overview

Our hierarchical motion blur traversal algorithm works entirely in two-dimensional homogeneous space (2DH) to robustly handle moving triangles crossing the $z = 0$ plane. The first step is conservative backface culling [Munkberg and Akenine-Möller 2011] and *temporal* view frustum culling. Each triangle vertex moves along a line in 2DH, and by finding the intersection of these three lines with each frustum plane, we obtain the time interval, $[t_s, t_e]$, when the moving triangle is inside the view frustum.

The traversal algorithm for a triangle can be summarized as follows, where a *tile* is a rectangular block of pixels:

```
1  BBOX = Compute moving triangle bounding box
2  for each tile in BBOX [hierarchical traversal]
3      TIME = Compute time interval of overlap
4      Occlusion culling of tile in TIME
5      for each sample in tile in TIME
6          Test sample against primitive
```

To compute a screen space bounding box around the moving triangle, we bound the screen space projections of the six vertices (the triangle vertices at t_s and t_e) if the moving triangle is entirely in front of the $z = 0$ plane, and revert to the conservative bounding approach presented by McGuire et al. [2010] otherwise.

In Section 4, we introduce the tile vs moving triangle tests (line 3), which form the necessary basis for our hierarchical traversal algorithm. The output for a certain tile is either *trivial reject*, or a conservative time interval where overlap possibly occurs. The computation of per-tile time bounds greatly reduces the number of temporal samples that are tested for fast moving primitives, as large subsets of the spatio-temporal samples within a tile can be discarded. It also makes hierarchical occlusion culling simple and efficient. For each tile, we only test the primitive against occlusion information in the relevant time interval (line 4).

As with all stochastic methods, the statistical distribution of the sample points has a large impact on the result. Stochastic rasterization has the additional constraints that the samples must be consistent from primitive to primitive (otherwise cracks may appear), and extremely fast to generate as the sampling takes place in the inner loop of the rasterizer. We have chosen to base our samples on binary (t, m, s)-nets [Niederreiter 1992] for their extensive stratification properties. Section 5 introduces a remapping of a known pattern to provide a temporally ordered sequence that is extremely inexpensive to compute in hardware. Last, we discuss temporal filtering for high-quality shading of the generated samples in Section 6, followed by implementation details and a thorough evaluation in Sections 7 and 8, respectively.

4 Tile Tests with Temporal Bounds

It is well-known that efficient rasterization of static geometry can be obtained by hierarchical testing of a tile of pixels against a triangle [McCormack and McNamara 2000]. This is done by overlap testing the bounding box of the triangle against the tile, and also testing each triangle edge against the tile [Akenine-Möller and Aila 2005]. We will extend this to moving geometry, where the bounding box becomes a moving box, and the triangle edges sweep through space.

More specifically, we derive tight bounds for the overlap between a screen space tile and a triangle with linear per-vertex motion in three dimensions. Each vertex, $\mathbf{p}_i$, moves from the position $\mathbf{q}_i$ at $t = 0$, to $\mathbf{r}_i$ at $t = 1$, that is: $\mathbf{p}_i(t) = (1-t)\mathbf{q}_i + t\mathbf{r}_i$. All computations are performed in 2D homogeneous coordinates, with a vertex defined as $\mathbf{p} = (p_x, p_y, p_w)$. The main idea is to find a conservative time interval, $\hat{t}_{tot} = [\underline{t}_{tot}, \overline{t}_{tot}]$, in which the moving triangle overlaps the tile. Per-sample tests are then done *only* for samples whose times belong to the time interval. In the following, we first describe how a tile is tested against a moving box, and then how a tile is tested against a moving triangle edge.

4.1 Frustum Plane–Moving AABB Overlap

We create a moving AABB in 2DH by bounding the triangle at $t = 0$ and $t = 1$ and interpolating between the two AABBs. This is an approximation to the true swept bounding box, but it is guaranteed

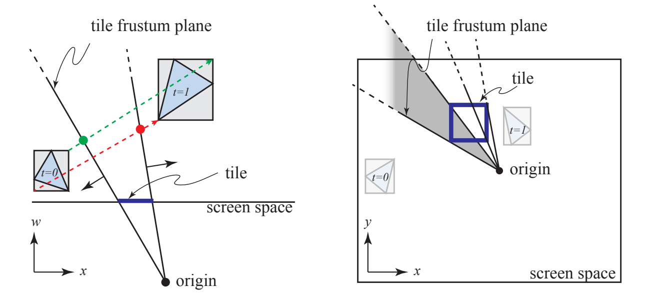

Figure 2: *A moving triangle is enclosed by an AABB in 2DH with linear per-vertex motion. The left figure shows the xw plane, with indicators when the moving AABB enters (green dot) and exits (red dot) the tile frustum. The right illustration shows the screen space view of this example.*

to be conservative at all times. Based on a tile on screen, we then setup four frustum planes that are aligned to the sides of the tile. Each frustum plane, π_i, passes through the origin and is defined by its plane equation $\mathbf{n}_i \cdot \mathbf{p} = 0$, where $\mathbf{n}_i$ is the plane's normal. A point $\mathbf{p}$ is outside the plane if $\mathbf{n}_i \cdot \mathbf{p} > 0$. If a point is inside all planes, then it is inside the frustum. For static geometry, it is sufficient to test the corner of an enclosing AABB that is farthest in the negative direction (n-vertex) relative to π_i [Greene 1994], in order to determine if the box is entirely in the positive half-space. The sign bits of the plane's normal, $\mathbf{n}_i$, directly decides which corner is the n-vertex. We note that the same holds for linearly moving bounding boxes, as the orientations of the frustum planes remain constant. Figure 2 shows an example of a moving triangle, whose moving AABB intersects with two tile frustum planes.

The point of intersection in time between the moving n-vertex and a plane π_i is given by:

$$\mathbf{n}_i \cdot ((1-t)\mathbf{q}_n + t\mathbf{r_n}) = 0 \iff t = \frac{\mathbf{n}_i \cdot \mathbf{q}_n}{\mathbf{n}_i \cdot (\mathbf{q}_n - \mathbf{r}_n)}, \quad (1)$$

where $(1-t)\mathbf{q}_n + t\mathbf{r_n}$ is the moving n-vertex for π_i. Let $d_0 = \mathbf{n}_i \cdot \mathbf{q}_n$ and $d_1 = \mathbf{n}_i \cdot \mathbf{r}_n$. The temporal overlap, $\hat{t}_i$, between the AABB and the plane π_i is given by:

$$\hat{t}_i = \begin{cases} \emptyset & \text{if } d_0, d_1 > 0 & \text{both outside} \\ [\max(0,t), 1] & \text{else if } d_0 > 0 & \mathbf{q}_n \text{ outside} \\ [0, \min(1,t)] & \text{else if } d_1 > 0 & \mathbf{r}_n \text{ outside} \\ [0,1] & \text{otherwise} & \text{both inside,} \end{cases} \quad (2)$$

where t is computed using Equation 1. The temporal overlap between all the tile planes and the moving AABB is given by $\hat{t}_{box} = \bigcap_i \hat{t}_i$, where we can test for fine-grained trivial rejection after each iteration of the loop over the four frustum planes, π_i.

4.2 Moving Triangle Edge Tests

For triangles with linear vertex motion in three dimensions, each triangle edge sweeps out a bilinear patch. The corresponding *time-dependent* edge functions are quadratic in t. To determine if a screen space tile overlaps the swept triangle, we evaluate the triangle's three edge equations for the four corners of the tile and check if any corner is inside all three edges. In that case, we again determine a time interval in which the triangle conservatively overlaps the tile to reduce the number of per-sample inside tests.

The edge equation for a triangle with linear per-vertex motion can be written as follows [Akenine-Möller et al. 2007]:

$$e(x,y,t) = \mathbf{n}(t) \cdot \mathbf{s} = (\mathbf{f}t^2 + \mathbf{g}t + \mathbf{h}) \cdot \mathbf{s}, \quad (3)$$

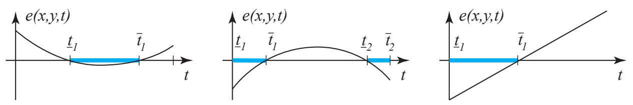

Figure 3: *Edge equations as functions of t for a specific (x, y) location. We are interested in finding the time intervals where $e < 0$ (highlighted in turquoise).*

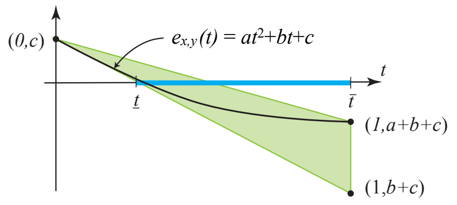

Figure 4: *The lower bound of a quadratic polynomial from Equation 4 is bounded by a linear approximation around $t = 0$.*

where $\mathbf{s} = (x, y, 1)$ is a sample position in screen space. For a given $\mathbf{s}$, we have a maximum of two roots to $e(x, y, t) = 0$, and up to two time intervals per edge, $\hat{t}_i = [\underline{t}, \bar{t}]$, where the sample is inside ($e < 0$). Some examples are given in Figure 3.

Handling near-linear edge motion in an efficient and robust way is extremely important, because often a large portion of the triangles in a scene will have close to linear motion. In these cases, directly finding the roots of $e = 0$ involves a division with a very small quadratic coefficient, which may lead to numerical instability. Therefore, we have devised a robust test that performs well when the edge equations are near-linear, and that is increasingly conservative when the quadratic term grows. We bound the quadratic edge function's projection within a screen space tile using lines with constant slopes. This *linearization* of the overlap test greatly reduces the computations needed. The edge function (Equation 3) is linearized according to:

$$e(x,y,t) = \mathbf{n}(t) \cdot \mathbf{s} \geq \mathbf{o} \cdot \mathbf{s} + \gamma t, \quad \forall \mathbf{s} \in S, \quad (4)$$

where S is a region in screen space, e.g., the bounding box of the swept triangle. With $\mathbf{n}(t) = \mathbf{f}t^2 + \mathbf{g}t + \mathbf{h}$, we rewrite the edge function for a certain screen space position, $\mathbf{s}$, as [Gribel et al. 2010]:

$$\mathbf{n}(t) \cdot \mathbf{s} = \mathbf{f} \cdot \mathbf{s}\, t^2 + \mathbf{g} \cdot \mathbf{s}\, t + \mathbf{h} \cdot \mathbf{s} = at^2 + bt + c, \quad (5)$$

where $a = \mathbf{f} \cdot \mathbf{s}$, $b = \mathbf{g} \cdot \mathbf{s}$ and $c = \mathbf{h} \cdot \mathbf{s}$. It can be shown that this curve is included in the triangle given by the points $(0, c)$, $(1, b+c)$ and $(1, a+b+c)$ as seen in Figure 4. We search for a lower linear bound of the curve's slope, which is given by:

$$\min(b, a+b) = \min(\mathbf{g} \cdot \mathbf{s}, (\mathbf{f} + \mathbf{g}) \cdot \mathbf{s}). \quad (6)$$

A conservative minimal slope, γ, for all $\mathbf{s} \in S$, is given by:

$$\gamma = \min_{\mathbf{s} \in S}(\mathbf{g} \cdot \mathbf{s}, (\mathbf{f} + \mathbf{g}) \cdot \mathbf{s}). \quad (7)$$

If we set $\mathbf{o} = \mathbf{n}(0) = \mathbf{h}$, we have obtained a linearized version of the edge equation according to Equation 4. This linear representation is conservative even if the edge function has a large quadratic term. Note that γ can be computed in the triangle setup using the moving triangle's screen space AABB as S. For more accurate bounding, γ can be recomputed on a coarse tile level, using the tile extents as S.

Given the linearization, the tile vs moving edge test is considerably simplified. By looking at the signs of the xy-components of $\mathbf{o}$, we

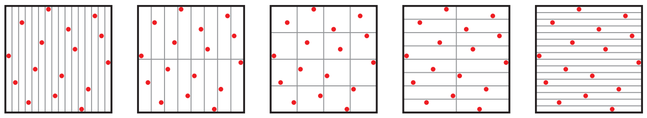

Figure 5: *Example of a $(0, 4, 2)$-net in base 2. The five figures illustrate all elementary intervals with area $b^{t-m} = 2^{-4}$ over the unit square, where each one has exactly $b^t = 2^0 = 1$ samples. We present a method for procedural construction of three-dimensional (t, m, s)-nets with properties targeted at motion blur rasterization.*

only need to test one tile corner, $\mathbf{s}$. A conservative time for the intersection of the triangle edge and the tile is given by:

$$\mathbf{o} \cdot \mathbf{s} + \gamma t = 0 \qquad \Longleftrightarrow \qquad t = -\frac{\mathbf{o} \cdot \mathbf{s}}{\gamma}. \tag{8}$$

Note that $-\frac{\mathbf{o}}{\gamma}$ can be precomputed, so the time of overlap for a tile only costs 2 `MADD` per edge. Depending on the sign of γ, the tile's temporal overlap, $\hat{t}_k$, with edge e_k is defined as:

$$\hat{t}_k = \begin{cases} [\max(0,t), 1] & \text{if } \gamma < 0, \\ [0, \min(1,t)] & \text{otherwise,} \end{cases} \tag{9}$$

where t is computed according to Equation 8. Once all three triangle edges have been tested, the temporal overlap between the tile and the swept triangle is given by $\hat{t}_{edges} = \bigcap_k \hat{t}_k$, where we can test for fine-grained trivial rejection after each iteration of the loop over edges (e_k). The final interval is the intersection of the intervals from both the moving box test (Section 4.1) and the moving edge test, i.e., $\hat{t}_{tot} = \hat{t}_{box} \bigcap \hat{t}_{edges}$.

Given $\hat{t}_{tot}$, we first perform spatio-temporal occlusion culling, and for the surviving tiles and time intervals, we proceed with individual sample-in-triangle inside tests. The following section describes the computation of our sampling positions, (x, y, t).

5 Sampling

In a motion blur rasterizer, each pixel is associated with a number of fixed (x, y, t) samples. The samples must be the same from triangle to triangle to get correct visibility, but vary randomly from pixel to pixel to reduce temporal and spatial aliasing. Stochastic sampling introduces noise, and it is well-known that sample points with good statistical properties, e.g., large minimum distance, provide a good balance between noise and aliasing. For us, it is also desirable that the samples project to a good distribution in (x, y) for high-quality anti-aliasing of static primitives.

Our application imposes a number of further constraints. First, sampling needs to be fast and use minimal storage, as it is performed at the core of the rasterizer. Second, each pixel should have the same number of samples to simplify hardware design. Additionally, since our tile tests compute the temporal overlap, $\hat{t}_{tot}$, it is important to be able to quickly find the relevant samples for a tile, i.e., the samples should be ordered in t. These requirements severely restrict our options. For example, Poisson disk points are not guaranteed to project to a good distribution in two dimensions, and it may be hard to guarantee a fixed number of samples.

For these reasons, we have chosen to work with sampling distributions that are realizations of digital (t, m, s)-nets [Niederreiter 1992]. Although often used for quasi-Monte Carlo integration in offline rendering [Kollig and Keller 2002], we believe samples based on digital nets are ideal also for motion blur rasterization due to their extensive stratification properties and ease of construction. Next, we will give a brief introduction (see Niederreiter's work [1992] for more details), and introduce a novel variation of a known method for generating three-dimensional samples with good properties. Our samples are ordered in t and have a good spatial distribution when projected to screen space.

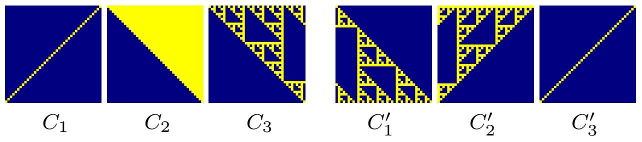

Figure 6: *The right three images show examples of our generator matrices for $m = 40$. The first two components are given by shifted and reflected Sierpiński triangles. These two matrices generate the same points as C_1 and C_2, but permuted into an order that better suits our purposes (ordered in t).*

Definition A set of b^m s-dimensional points $\mathbf{x}_j = (x_j^{(1)}, \ldots, x_j^{(s)})$ is a (t, m, s)-net in base b if every *elementary interval* of volume b^{t-m} contains exactly b^t points, where $b \geq 2$ and $0 \leq t \leq m$ are integers. The elementary intervals are discrete subintervals of space:

$$E = \prod_{i=1}^{s} \left[\frac{a_i}{b^{l_i}}, \frac{a_i + 1}{b^{l_i}} \right) \subseteq [0, 1)^s, \tag{10}$$

where $0 \leq l_i$ and $0 \leq a_i < b^{l_i}$ are integers. The volume constraint gives $\sum_{i=1}^{s} l_i = m - t$. An example in two dimensions is shown in Figure 5. In our case, $s = 3$ and we work exclusively with binary numbers ($b = 2$) for efficiency reasons. Lower t value (referred to as "quality") gives better stratification, i.e., fewer points per stratum. Hence, we are interested in $(0, m, 3)$-nets in base 2, which have 2^m points and exactly one point per elementary interval. This property ensures that samples near in time are spatially far apart, and vice versa, which is important to minimize noise.

A *digital* (t, m, s)-net can be defined using a set of *generator matrices* $C_1, \ldots, C_s$ over a finite field $\mathbb{F}_q$, where q is prime [Niederreiter 1992]. $\mathbb{F}_q$ consists of elements numbered $\{0, \ldots, q-1\}$, and all arithmetic operations are performed modulo q. Since we work in base 2, $q = 2$ and the C_i matrices are binary $m \times m$ matrices. The i^{th} component of the j^{th} point is given by:

$$x_j^{(i)} = \left(2^{-1}, \ldots, 2^{-m}\right) \left[C_i \begin{pmatrix} d_0(j) \\ \vdots \\ d_{m-1}(j) \end{pmatrix} \right] \in [0, 1), \tag{11}$$

where $d_k(j)$ are the bits of j, $j \in \{0, \ldots, 2^m - 1\}$, with d_0 being the least significant bit. The matrix-vector product $C_i \left(d_0(j) \cdots d_{m-1}(j)\right)^T$ is performed in $\mathbb{F}_2$.

Our Method Three-dimensional digital nets with good 2D projections are not very well explored. Grünschloß and Keller [2009] propose one method based on reordering of the Sobol' sequence, where the first two dimensions are the Larcher-Pillichshammer (LP) points [Kollig and Keller 2002], which have a good spatial distribution. The first component is sequentially ordered, i.e., $x_j^{(1)} = \frac{j}{2^m}$. Unfortunately, we have observed that the projection onto the other two dimensions is not as well-distributed, exhibiting a structure of diagonal lines. See Figure 7 (left). This leads to inferior spatial anti-aliasing, as our algorithm uses the ordered dimension as time.

To address this, we propose a *permuted* construction that is ordered in t, while still projecting to the LP-points in the remaining two dimensions, as shown in Figure 7 (right). Our samples are given by

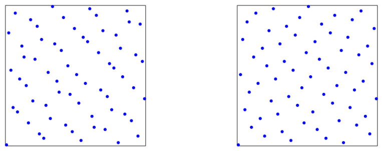

Figure 7: *The original samples [Grünschloß and Keller 2009] are ordered in the first component. The projection onto the other two dimensions are shown on the left for $m=6$. After our permutation, the non-time dimensions project to the Larcher-Pillichshammer points (right), which give better spatial anti-aliasing.*

the generator matrices (see Appendix A for details):

$$C'_1 = \left(\binom{m+1-l}{m+1-k} \bmod 2 \right)_{k,l=1}^{m}, \qquad (12)$$

$$C'_2 = \left(\binom{m-l}{k-1} \bmod 2 \right)_{k,l=1}^{m} \quad \text{and} \quad C'_3 = \begin{pmatrix} 0 & & 1 \\ & \iddots & \\ 1 & & 0 \end{pmatrix}.$$

These binary matrices are visualized in Figure 6. The figure compares our matrices to the original matrices of Grünschloß and Keller, denoted C_1, C_2, C_3. Note that our modified matrices compute the *same* set of points as before, but the points are generated in a different order. The order is important, as our input is a sequential index in time, and the remaining two dimensions are used as the spatial sample position. We want the projection to screen space to be as good as possible for high-quality spatial anti-aliasing. This is especially important for static and slowly moving primitives, where the user gets plenty of time to study the quality. Note that reordering the points by permuting the matrices is not the same as just assigning the dimensions differently.

Although the matrices look deterring, they make an efficient procedural computation of samples possible. Addition equals XOR in $\mathbb{F}_2$, so entire columns of the matrix-vector product in Equation 11 can be added using single XOR operations. Additionally, we omit the leftmost vector multiplication and view the result as the digits of a fixed-point representation. The following C-function computes the xy-coordinates of the point with sequential index j, i.e., sample time $t = j/2^m \in [0,1)$, for any $m < 32$. All coordinates are integers in $\{0,\dots,2^m-1\}$.

```
void GetXY(uint j, const uint m, uint& x, uint& y) {
   x = y = 0;
   uint c1 = 0x3, c2 = 0x1 << (m-1);
   for (j <<= 32-m; j != 0; j <<= 1) {
      if (j & 1u<<31) { // Add matrix columns (XOR)
         x ^= c1 >> 1;
         y ^= c2;
      }
      c1 ^= c1 << 1;     // Update matrix columns
      c2 ^= c2 >> 1;
   }
}
```

The algorithm examines j one bit at the time, starting at its high bit $m-1$, and adds up columns of C'_1 and C'_2. The matrices are computed on the fly, using only bit shifts and XOR operations. In hardware, the above algorithm can be implemented using a very small number of gates. During rasterization, we generate samples for $t \in \hat{t}_{tot}$ at the finest hierarchical level in the traversal. For example, with 4×4 pixel tiles at 16 samples/pixel, we have 256 samples, so $m = 8$ and the samples' xy-coordinates are interpreted as 2.6 bits fixed-point numbers (i.e., the top two bit gives the pixel position, and the lower six bits the sub-pixel placement). Throughout the paper, we also quantize time to 64 discrete values (see Section 7), although this is not a necessity.

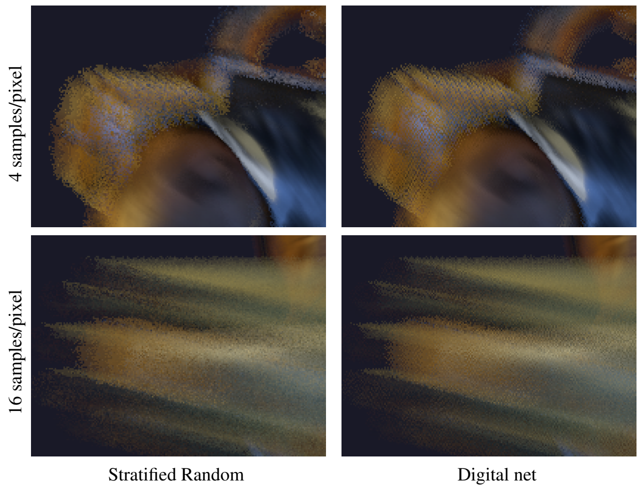

Figure 8: *Comparison between stratified random sampling (left) and sampling based on digital nets (right), at two different sampling rates. The regularity in the digital net based pattern gives a noticeably smoother appearance, while avoiding obvious aliasing.*

To avoid a repeating sample pattern, we apply *random digit scrambling* [Kollig and Keller 2002] to the generated xy-coordinates, as it gives good results at an extremely low cost. Conceptually, the sampling domain is hierarchically split in half along each spatial dimension, and the two halves randomly permuted. The same permutations are applied to all samples within a tile. In base 2, this operation can be performed by a bitwise XOR between the xy-coordinates and two independent random bit vectors. We compute the random vectors based on a hash of the tile position, which ensures consistency from frame to frame. The regular structure of the scrambled digital net gives low noise without any obvious aliasing artifacts. Note that random digit scrambling also largely preserves the properties of the projected samples (Figure 7), which is an important aspect. To increase the randomness, a sub-pixel jittering can be applied, but we have not found that to be necessary. Figure 8 shows the rendering quality compared to traditional stratified random sampling, where xy and t have been independently stratified per pixel.

Discussion An alternative to using procedurally generated samples is to store a lookup table of samples, ordered in t. This allows for more flexibility, but incurs an additional hardware cost. Inspired by Grünschloß and Keller [2009], we have experimented with randomized permutation-based search for $(0, m, 3)$-nets with larger minimum point distance than the above construction, but the results of this has been left for future work.

6 Shading

A core feature of our algorithm is that we visit a particular pixel at most *once* for a certain triangle. This is similar to McGuire et al.'s work [2010], but different from interleaved and interval rasterization [Keller and Heidrich 2001; Fatahalian et al. 2009], where a pixel may be visited many times for the same triangle. Using our traversal order, it is therefore possible to use multisampling anti-aliasing (MSAA) strategies with temporal filtering [Loviscach 2005; McGuire et al. 2010], which is not feasible for interleaved and interval rasterization since they slice the time dimension. As will be seen in Section 8, multisampling can give a considerable reduction in shader cost, which is a big advantage when implementing the traversal algorithm in an existing GPU pipeline. It should be noted that decoupled shading solutions [Ragan-Kelley et al. 2011;

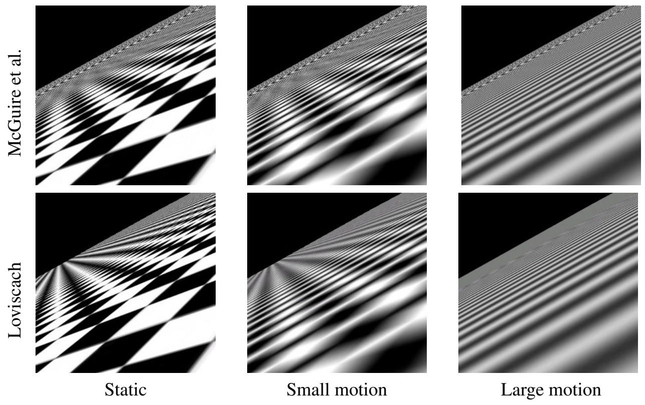

Figure 9: *Comparison between the filter kernel approximations proposed by McGuire et al. (top) and Loviscach (bottom). The results are rendered on an NVIDIA GeForce 290 GTX with* $16\times$ *anisotropic filtering. Note the severe aliasing in the top row. This is due to over-emphasizing the motion contribution, and the approximation of the screen space filter kernel as a fixed* $(1,1)$*-vector.*

Burns et al. 2010] show more promise to further reduce the shading cost. However, the multisampling approach can be implemented on current hardware with no, or very small, modifications as shown by McGuire et al [2010], which makes it a good first step towards a graphics hardware solution fully supporting motion blur. Although our shading system is very similar to previous work, we describe it briefly since being able to use multisampling is currently an important feature of our algorithm.

We base our temporal shader filtering on the work of Loviscach [2005] and McGuire et al. [2010]. The basic idea is to use anisotropic texture filtering to integrate textures over the motion footprint by modifying the derivatives. By integrating over time in the shader, it is possible to sample the shader only once per pixel, and write the result to all covered samples.

We assume that the only varying shader inputs are the barycentric coordinates $u(x,y,t), v(x,y,t)$, and disregard from explicit shader input variables representing the sample time. For texture filtering, we need to estimate the texture footprint. The integration domain is given by a fourth order rational function in t, but we choose to make the same approximations as Loviscach [2005], and use his approach for perturbing the screen space texture gradient axes to account for the temporal derivative. McGuire et al. [2010] present an approximation where the screen space gradients use a fixed axis in texture space. However, this method suffers from severe aliasing for some view directions, as can be seen in Figure 9.

We compute $\frac{\partial u}{\partial x}$, $\frac{\partial u}{\partial y}$ and $\frac{\partial u}{\partial t}$ by finite differences (same for the partial derivatives of v). For each quad of 2×2 pixels with at least one sample covered, we evaluate the shader at five points: each of the four pixel centers at $t = 0.5$ for the quad, in order to compute x and y derivatives using finite differences, and one additional point for one of the samples at $t = 1$ to compute a per-quad approximation of the temporal derivative. For a more fine-grained temporal derivative, one can shade the entire quad at two distinct times and compute per-pixel temporal derivatives, or shade on a per sample/pixel basis, which would shade four points to compute per-pixel $\frac{\partial u}{\partial x}$, $\frac{\partial u}{\partial y}$, and $\frac{\partial u}{\partial t}$ derivatives (and the partial derivatives of v). We use the quad approximation in all our tests. The shading approach integrates very well into existing pinhole camera rendering systems with MSAA support with relatively modest modifications to the hardware.

It should be noted that our approach for computing derivatives may lead to shading samples that lie outside the triangle. This can cause shading artifacts, but we have not found them to be significant in our test scenes. McGuire et al. [2010] used a different approach and picked the last covered time sample as the shading sample, but re-projected to $t = 0$. This has other implications such as overblurring due to too large filter kernels. Working out a strategy for correct shader filtering and derivative computations with stochastic sampling is an interesting problem that deserves a thorough evaluation. However, we leave that for future work.

With interleaved rasterization [Keller and Heidrich 2001; Fatahalian et al. 2009] in a pipeline with shading after visibility, there are no adjacent screen space samples with the same sample time. Without introducing large hardware changes (e.g., a shader cache) the spatial derivatives can be computed in two ways. One option is to compute derivatives per sample by executing the shader at three spatial positions. The other option is to compute derivatives using finite differences from four nearby samples with the same time, while taking the perturbed sample positions into account. This method produces coarser derivatives, as the samples with the same time are typically separated by two or more pixels. In either case, as we shade at all sample times, the temporal derivatives are less important and can be ignored without much loss in image quality. We chose to use the first alternative for the statistics presented in this paper, as the quality more closely matches that of our derivatives. Furthermore, coarse derivatives will degrade the quality even for static scenes compared to current graphics API specifications, which we want to avoid.

7 Implementation

Hierarchical Rasterizers To evaluate our algorithm, we have implemented four rasterizers called CONVEX, OUR, INTERVAL and HIERARCHICAL INTERLEAVE in a software rasterization pipeline that can execute DX11 traces, but lacks a general shading system. In order to evaluate shading and sampling quality, we have also designed a GPU rasterizer which performs stochastic rasterization in a pixel shader, similar to McGuire et al.'s work [2010].

CONVEX (based on McGuire et al. [2010]) traverses all tiles within an AABB overlapping the screen space convex hull ($\mathcal{CH}$) of the moving triangle. The original paper uses a two-step algorithm that triangulates the convex hull and rasterizes stochastically in the pixel shader (to run on current GPUs). In our software implementation, each tile is tested against the $\mathcal{CH}$ edges, and if it overlaps, all spatio-temporal samples within the tile are tested against the triangle. Both approaches perform hierarchical rasterization, but the latter may be more efficient in hardware, as a tile is visited only once. OUR algorithm is similar, but uses the tile tests with temporal bounds introduced in Section 4 instead of the $\mathcal{CH}$ edges. The temporal bounds significantly reduce the number of samples that must be tested. INTERVAL is based on Pixar's motion blur algorithm (described by Fatahalian et al. [2009]). The main difference compared to CONVEX and OUR is the iteration order, where INTERVAL has an outer loop over sample times instead of over tiles. This is the only algorithm that cannot be easily extended to hierarchical rasterization. We still use a tiled traversal approach for the sake of hierarchical depth culling, but we have no test for trivially rejecting a non-overlapping tile. HIERARCHICAL INTERLEAVE is a hierarchical 2DH version of Fatahalian et al.'s [2009] interleaved rasterizer, where the triangle is rasterized with an interleaved sampling pattern. Like INTERVAL, this algorithm has the outer loop over sample times rather than tiles. Note that for mixed triangle sizes, adding a hierarchical test to Fatahalian et al.'s [2009] interleaved rasterizer improves performance a lot, which is to be expected since the target of the original paper was micropolygon rendering.

	CONVEX	OUR
Iteration order	Tiles	Tiles
Screen space bbox	BBox of $\mathcal{CH}$	Bound tri over $[0, 1]$
Tile test	$\mathcal{CH}$ edges	Swept tri
Occl. & sample test	$t \in [0, 1]$	$t \in \hat{t}_{tot}$
Shading	MSAA	MSAA
Sampling pattern	Arbitrary	Ordered in t

	INTERVAL	HIER.INTERLEAVE
Iteration order	Sample times	Sample times
Screen space bbox	Bound tri over $[t_i, t_{i+k}]$	Bound tri at t_i
Tile test	None	Tri edges at t_i
Occl. & sample test	$t = [t_i, t_{i+k}]$	$t = t_i\ \forall i$
Shading	Supersampling	Supersampling
Sampling pattern	Ordered in t	Interleaved

Table 1: *Algorithm comparison. $\mathcal{CH}$ denotes the screen space convex hull of the moving triangle. For* INTERVAL*, we divide the N unique sample times in N/k intervals.*

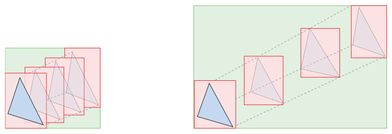

Figure 10: *Two moving triangles are rasterized at four fixed times for illustrative purposes. With little motion, the individual bounding boxes (red) overlap, which makes a screen space traversal in the swept bbox (green) preferable. At higher motion, the bounding boxes separate and* HIERARCHICAL INTERLEAVE *traversal is more efficient.*

All algorithms perform backface culling [Munkberg and Akenine-Möller 2011] and view-frustum culling (including temporal bounds). In all tests, we use the same high-quality interleaved sampling patterns, described in Section 5, with 64 fixed times. The reason for this is that the edge functions can be pre-computed for 64 times in the triangle setup and reused over the triangle, which gives substantially reduced costs for our test scenes. The inside test, which uses 2DH edge equations, is therefore identical for all four algorithms. The traversal strategies are summarized in Table 1.

Using a sample pattern with a fixed set of sample times, there is an amount of motion where a fast moving triangle has no spatial overlap between adjacent times, t_i and t_{i+1}. Figure 10 illustrates this. In this case, there is little temporal coherence to exploit, i.e., each tile has only one or a few covered samples. Therefore, we propose a fallback to HIERARCHICAL INTERLEAVE traversal whenever the individual bounding boxes no longer overlap. We use a simple heuristic to decide when this occurs. The dimensions of the bounding boxes at $t = 0$ and $t = 1$ are $w_0 \times h_0$ and $w_1 \times h_1$, respectively, and the swept bounding box $w_s \times h_s$. If we rasterize at N unique times, HIERARCHICAL INTERLEAVE traversal is chosen whenever $\min(w_0, w_1) < w_s/N$ or $\min(h_0, h_1) < h_s/N$. We use this fallback in all our measurements of OUR, CONVEX, and INTERVAL. It is primarily activated for very small, fast moving triangles.

At $16\times$ multisampling, we use three hierarchical levels with 16×16, 8×8, and 4×4 pixel tiles for all algorithms except HIERARCHICAL INTERLEAVE (which uses 32×32, 16×16 and 8×8 pixel tiles). Depth culling is performed on the coarsest and finest levels, and trivial reject or time overlap test are performed on the two finer levels. These configurations were determined by extensive evaluation of both arithmetic cost and bandwidth usage. We use coarser tile levels for HIERARCHICAL INTERLEAVE since the tile tests are executed for each sample time t_i. For HIERARCHICAL INTERLEAVE, a single tile test at 4×4 pixel tiles culls at most 4 samples with $t = t_i$. In contrast, up to 256 samples with $t \in [0, 1]$ can be culled for 4×4 pixel tiles with our tile tests.

Scene	Triangles	Resolution	Motion
StoneGiant1	0.26M	1280 × 720	Camera translation
StoneGiant2	4.1M	1920 × 1200	Camera rotation
Heaven	2.4M	1920 × 1080	Camera translation
SubD11	0.16M	1920 × 1080	Keyframed animation

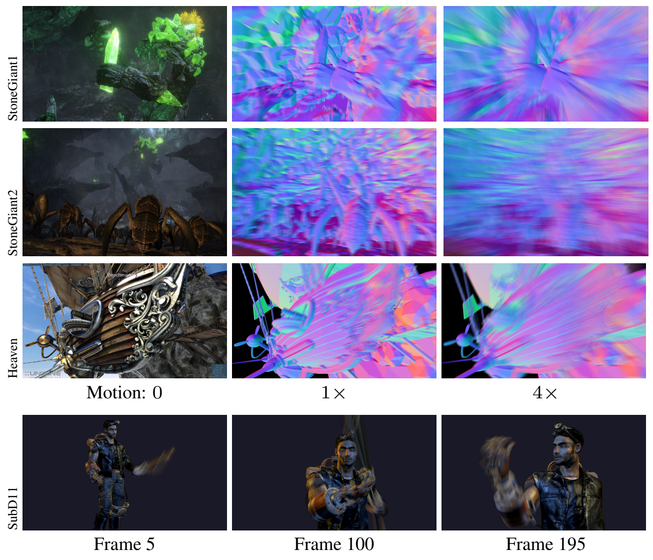

Figure 11: *Our test scenes with two frames from the StoneGiant demo (courtesy of BitSquid), one from the Heaven 2 demo (courtesy of Unigine Corp.), and the SubD11 animation from Microsoft DX SDK (June 2010). Motion blur has been added to all scenes.*

Time-Dependent Occlusion Culling A traditional *z-max* buffer stores one conservative z-max value for each screen space tile. We use a *time-dependent* z-max buffer [Akenine-Möller et al. 2007; Boulos et al. 2010], which contains multiple temporal depth values per tile for increased culling efficiency in the presence of motion blur. For better efficiency, our algorithm uses the tile's temporal overlap, $\hat{t}_{tot}$ (Section 4), to avoid performing occlusion queries for time intervals when the triangle does not overlap. Our simulator uses a fully associative cache backing the depth and z-max buffers with 64 bytes cache lines. In the following, we denote the number of different sample times represented in one cache line as s. A corresponding z-max value then represent s times over a screen space tile whose spatial extents is proportional to $1/s$ to make it fit in the cache line. A coarse z-max buffer is either constructed for each of the 64 times, and hence $s = 1$, or for a group of consecutive times (representing a smaller screen space area), where $s > 1$. In all cases, we have one z-max value for each cache line of sample depths.

The temporal coherence may be further exploited by constructing a temporal hierarchy [Boulos et al. 2010]. We explored this with a two-level spatio-temporal cache-backed hierarchy, but were not able to reduce bandwidth usage. This is partly due to the cache line grouping of values – large chunks of geometry must be successfully culled in order to avoid reading z-max data – and partly due to the added cost of keeping another level of z-max data in the cache. Also note that our tests are not using any depth compression. However, a two-level temporal hierarchy decreased the arithmetic cost by around 5–10%, and we use it for all algorithms.

8 Results

Our test scenes are presented in Figure 11 and include various types of motion, triangles sizes, and geometry distributions. All results were generated using our own simulation framework de-

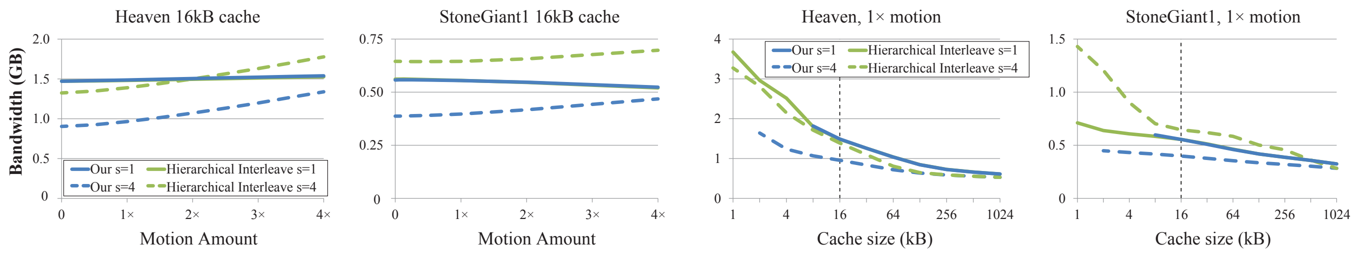

Figure 12: *Bandwidth usage for z and z-max at* 16 *samples per pixel. Left: varying amount of motion with a 16kB depth cache. A higher temporal z-max resolution (lower s) scales better with increasing motion, but has a higher constant cost.* $s = 4$ *is a suitable choice for* OUR*, as the crossover occurs at extreme motion (outside the graph). Right: varying cache size with fixed amount of motion. The minimum cache requirement for our algorithm is to accommodate all N time layers and the z-max storage. Our algorithm with* $s = 4$ *scales well to decently small cache sizes.* CONVEX *is not shown as it behaves similar to our algorithm, with the only difference that more z-max queries are performed.*

scribed in Section 7 with 16 samples per pixel. We present results for the depth buffer bandwidth, the number of shader executions, and the number of arithmetic operations required for rasterization. In most charts, we have also included a standard hierarchical rasterizer without motion blur. This is referred to as STATIC.

Depth Buffer Bandwidth Reducing memory bandwidth usage is incredibly important, and therefore, we start with a study on depth buffer bandwidth usage. Figure 12 (left) shows the depth buffer memory bandwidth usage from cache misses (including both sample depths and z-max values) when the number of sample times per cache line, s, is varied. Grouping more times into a cache line increases the penalty of larger motion, but lowers the bandwidth usage for parts of the scene moving slowly. For our algorithm, $s = 4$ is preferable for a wide range of motion. We use this number for all measurements for OUR, CONVEX, and INTERVAL since they have similar access patterns. Note, however, that our algorithm performs slightly fewer hierarchical occlusion queries than CONVEX since we use the results from the tile test to avoid testing some time intervals. More aggressive temporal grouping ($s = 16$) only pays off for very small motion, while no temporal grouping ($s = 1$) is more efficient for extreme motion. HIERARCHICAL INTERLEAVE scales differently, and the benefit for grouping sample times into the same cache line is low even for static scenes. This is an effect of the traversal order, where the triangle is fully traversed for one sample time before continuing with the next. For larger triangles, this may lead to eviction of the first cache line before starting traversal for the next sample time. Therefore, we use the $s = 1$ framebuffer layout for the HIERARCHICAL INTERLEAVE algorithm.

To determine a suitable cache size, we ran the bandwidth measurements with various cache sizes as presented in Figure 12 (right). Our algorithm needs a minimum cache size of 2kB (and in the case of $s = 1$, it needs 8kB) to avoid a 100% miss rate. However, above this minimum, it scales better than HIERARCHICAL INTERLEAVE traversal for decreasing cache sizes. In fact, HIERARCHICAL INTERLEAVE does not level out until the cache becomes very large (128kB–1MB). For the remaining comparisons, we use a 16kB cache for all algorithms. This corresponds to approximately 256 fully covered pixels worth of data, not counting z-max storage.

As shown in Figure 12 and 16 (top row), the depth buffer bandwidth (including both sample depths and z-max values) to external memory is relatively constant for the HIERARCHICAL INTERLEAVE traversal order with increasing motion, while it is increasing somewhat for OUR, CONVEX, and INTERVAL. Our algorithm becomes less efficient than HIERARCHICAL INTERLEAVE at some point. However, extreme motion is needed to reach the break-even point, and our algorithm consistently outperforms the competing algorithms except for the difficult StoneGiant2 scene with extreme motion (>4×). In Figure 13, we see that our algorithm uses less bandwidth than HIERARCHICAL INTERLEAVE for the SubD11 animation. In fact, in most frames, OUR uses only about 50% of that of HIERARCHICAL INTERLEAVE, even though the scene is animated at a low frame rate (24 fps) and contains frames with relatively large motion. Our algorithm also uses slightly less bandwidth than CONVEX since our tile tests with temporal bounds enable more efficient occlusion culling. This is particularly noticeable in the frames with largest motion. The depth buffer bandwidth of INTERVAL is similar to our algorithm but with one important difference. Since INTERVAL does not have a trivial reject test we need to perform depth culling for all tiles overlapping the triangle bounding box. This is not significant for small triangles or for scenes with only motion in the x- or y-directions. However, for SubD11 which contains large sliver triangles, this leads to many unnecessary depth culling queries and increases bandwidth significantly.

Shading Efficiency Figure 14 shows the number of shader executions per frame for the SubD11 animation. For OUR and CONVEX, we use multisampling, while HIERARCHICAL INTERLEAVE traversal has to resort to supersampling due to the traversal order (i.e., for each time, there is only one sample per tile in the interleaved sampling pattern). For INTERVAL, we use 16 time intervals. This implies that within each time interval, there is one sample per pixel in the interleaved sampling pattern. As can be seen, the benefit of multisampling is very high for SubD11. Also, INTERVAL is more efficient than HIERARCHICAL INTERLEAVE since shading is computed once per interval (16×) rather than per sample time (64×), and quad-fragment shading and finite differences can be used.

At extreme motion relative to the primitive size, we effectively revert to supersampling. This happens at >4× motion for the StoneGiant2 scene, which is highly tessellated (see the middle row in Figure 16). It is questionable whether this extreme motion will be usable for real-time rendering (> 60 fps). In all cases, the shading overhead compared to STATIC is often quite high. Given these observations, we conclude that better long term solutions for shading may be cache or object-space based approaches, as discussed in Section 6. However, we believe that a multisampling approach for motion blur is likely to be adopted by hardware vendors as an intermediate step towards a pipeline with decoupled shading.

Rasterization Cost We have instrumented our code with cost estimations for the critical stages of the rasterization algorithm: triangle setup, tile test, sample test, and interpolation setup. We only account for the cost of the more complex operations such as ADD, MUL, RCP, etc., and disregard from the cost of bit-twiddling and control logic. Therefore, our results should not be seen as absolute costs for an implementation, but rather demonstrate the general

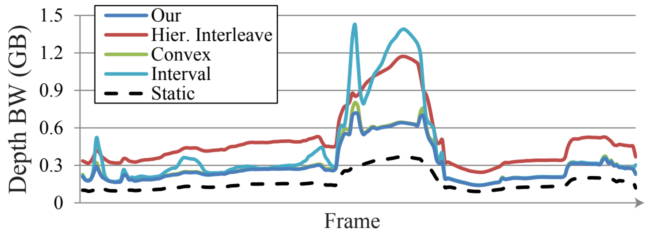

Figure 13: *Depth buffer bandwidth for the SubD11 animation.*

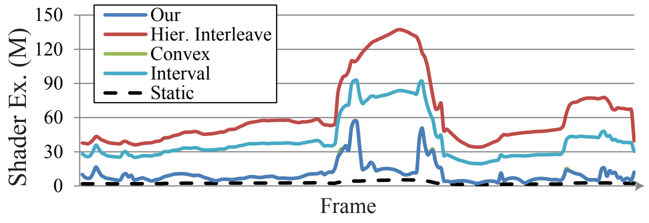

Figure 14: *Number of shader executions for SubD11.*

trend of the algorithms and how they relate. We intentionally do not use sample test efficiency [Fatahalian et al. 2009] as an efficiency measure, since it only includes a part of the rasterization cost. For example, sample test efficiency would benefit of using as small tiles as possible, but in practice, the best tradeoff is found with medium sized tiles where the sum of the tile test cost and sample test cost is minimized.

In Figure 15, we show a breakdown of the costs for the different stages of the rasterizer for the SubD11 animation. HIERARCHICAL INTERLEAVE has a significant triangle setup cost, due to that each triangle is bounded at $N = 64$ discrete times. Also, the interpolation setup is more expensive, as the shading is supersampled. Additionally, the sample test work is increased due to a larger screen space tile size. We have experimented with finer tile sizes, but that resulted in a substantial increase in the tile test cost, making the overall cost increase. INTERVAL performs quite poorly for this test scene. The reason for this is that the model contains large sliver triangles and since the INTERVAL rasterizer lacks a hierarchical tile-overlap test, it performs many unnecessary sample tests. For CONVEX, the cost of sample testing dominates, and varies widely. Our algorithm has more expensive per-tile tests, but they manage to cull a larger part of subsequent work. The total arithmetic cost using our two tile tests (moving box and moving edge) is presented in Table 2. We also measured the efficiency of our linearized edge test compared to directly solving the quadratic edge equation per tile. On the entire SubD11 animation (the scene with most complex motion), the linearized test results in 8% more inside test, but reduces the total arithmetic cost by 13% thanks to less expensive per-tile computations. On the Heaven scene, the linearized test results in 4% more inside test, but a total cost reduction of 24%.

	Static BBox	Box	Edge	**Box+Edge**	
SubD11	8.5(58)	6.2(32)	1.8(4.9)	1.7(4.5)	$\times 10^9$
Heaven	160(200)	17(21)	17(22)	14(18)	$\times 10^9$

Table 2: *Efficiency of our tile tests in terms of average total arithmetic cost per frame (maximum in parenthesis). The combination of the two tests gives a cost efficient and robust tile test.*

In Figure 16 (bottom row), we show how the four different motion blur algorithms scale with increased motion. StoneGiant1 and Heaven both have motion blur from a camera translation and show similar trends despite a large difference in tessellation. For modest motion ($< 1\times$), our algorithm has about half the traversal cost of HIERARCHICAL INTERLEAVE, and has roughly the same cost at extreme motion. INTERVAL scales similar to our algorithm but has lower overall performance. CONVEX is efficient for small motion, but the arithmetic cost grows very quickly for larger motion. StoneGiant2 is a highly tessellated frame with a camera rotation, so that each triangle gets similarly long motion trails. There are about a million triangles in the two front-most spiders alone. This is a worst-case scenario for a screen space *hierarchical* traversal, as a large fraction of the scene geometry is near pixel-sized. Here, HIERARCHICAL INTERLEAVE handles extreme motion robustly at a significant cost (around 40 Gops per frame). Our approach is competitive up to about $2\times$ motion. At extreme motion levels, nearly all triangles are completely separated (see Figure 10), so there is no gain in using a screen space traversal algorithm.

9 Conclusions

This paper has described the first efficient algorithm for hierarchical rasterization of motion blur. The algorithm builds on novel tile tests that compute the temporal overlap between a screen space tile and a moving primitive. We have shown that these temporal bounds are important to reduce the volume of tested samples and enable efficient hierarchical occlusion culling. We have further devised a high quality sampling method that uses the temporal bounds to quickly generate samples with good statistical properties. Our method is based on reordering of a known three-dimensional digital (t, m, s)-net to better fit the requirements for motion blur rasterization. Finally, we have provided extensive measurements of depth buffer bandwidth usage, arithmetic intensity, and shader efficiency for MSAA on modern complex workloads, which we have not seen in other studies.

In this paper, we have taken one step towards efficient motion blurred rendering on graphics processors. Our focus has been on a rather non-intrusive change to current GPUs. One area that needs more work is efficient shading, which makes the next natural step to add a decoupled shading cache [Ragan-Kelley et al. 2011; Burns et al. 2010]. This is left for future work at this point. In addition, it would be interesting to transform the algorithms into using efficient fixed-point math robustly. We hope that our work will help drive a continued interest in stochastic rasterization as a realistic method to achieve high-quality motion blur in future graphics pipelines.

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

AKENINE-MÖLLER, T., AND AILA, T. 2005. Conservative and Tiled Rasterization Using a Modified Triangle Set-Up. *Journal of Graphics Tools, 10*, 3, 1–8.

AKENINE-MÖLLER, T., AND STRÖM, J. 2003. Graphics for the Masses: A Hardware Rasterization Architecture for Mobile Phones. *ACM Transactions on Graphics, 22*, 3, 801–808.

AKENINE-MÖLLER, T., MUNKBERG, J., AND HASSELGREN, J. 2007. Stochastic Rasterization using Time-Continuous Triangles. In *Graphics Hardware*, 7–16.

BOULOS, S., LUONG, E., FATAHALIAN, K., MORETON, H., AND HANRAHAN, P. 2010. Space-Time Hierarchical Occlu-

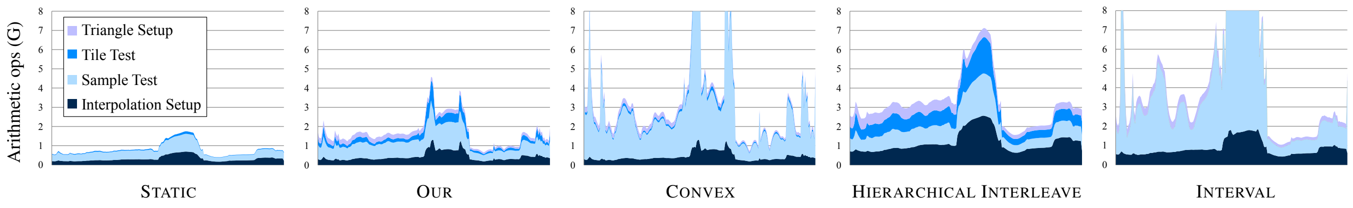

Figure 15: *Arithmetic cost breakdown for different traversal algorithms on the* SUBD11 *animation at* $16\times$ *samples per pixel. Note that the* CONVEX *traversal has substantially higher number of sample tests, due to the lack of temporal bounds per tile. Due to the lack of hierarchical testing,* INTERVAL *and* INTERLEAVE *also suffer from excessive sample testing. Hence, we only use the improved* HIERARCHICAL INTERLEAVE *in our measurements.*

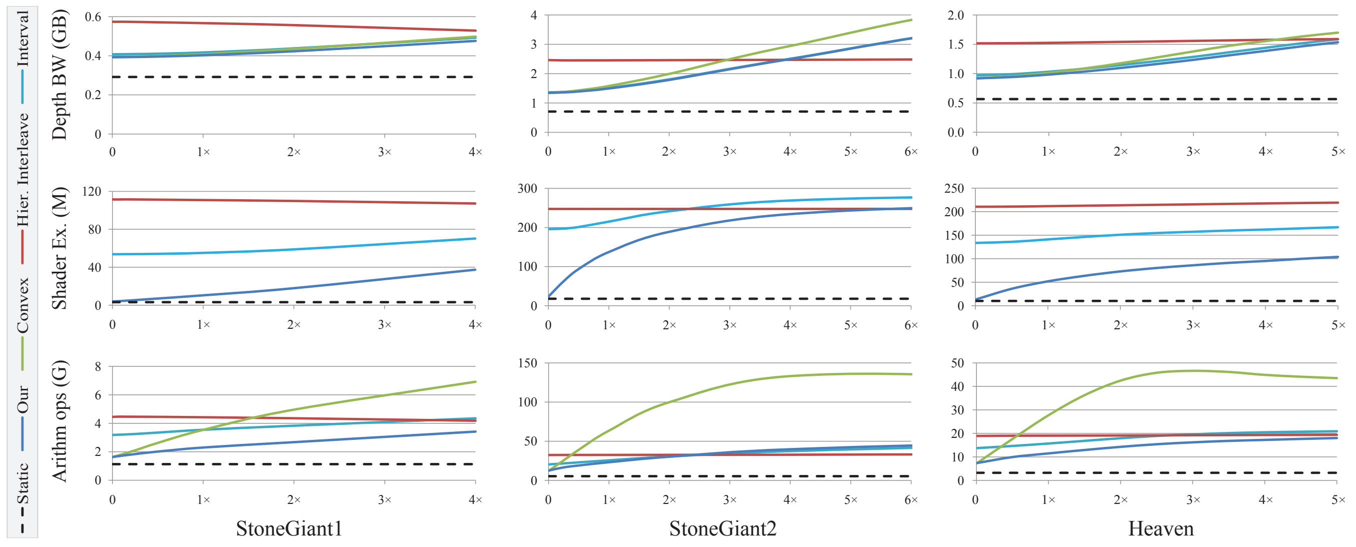

Figure 16: *The total depth buffer bandwidth, #shader executions, and arithmetic cost for the StoneGiant and Heaven scenes, as functions of the motion amount, where* $1\times$ *denotes a modest motion amount and* $4\times$ *represents extreme motion. See screenshots in Figure 11. Note that* CONVEX *has the same number of shader executions as* OUR

sion Culling for Micropolygon Rendering with Motion Blur. In *High-Performance Graphics*, 11–18.

BRAWER, R., AND PIROVINO, M. 1992. The Linear Algebra of the Pascal Matrix. *Linear Algebra and its Applications, 174*, 13–23.

BRUNHAVER, J., FATAHALIAN, K., AND HANRAHAN, P. 2010. Hardware Implementation of Micropolygon Rasterization with Motion and Defocus Blur. In *High-Performance Graphics*, 1–9.

BURNS, C. A., FATAHALIAN, K., AND MARK, W. R. 2010. A Lazy Object-Space Shading Architecture with Decoupled Sampling. In *High-Performance Graphics*, 19–28.

COOK, R. L., PORTER, T., AND CARPENTER, L. 1984. Distributed Ray Tracing. In *Computer Graphics (Proceedings of SIGGRAPH 84)*, ACM, vol. 18, 137–145.

COOK, R. L., CARPENTER, L., AND CATMULL, E. 1987. The Reyes Image Rendering Architecture. In *Computer Graphics (Proceedings of SIGGRAPH 87)*, ACM, vol. 21, 95–102.

FATAHALIAN, K., LUONG, E., BOULOS, S., AKELEY, K., MARK, W. R., AND HANRAHAN, P. 2009. Data-Parallel Rasterization of Micropolygons with Defocus and Motion Blur. In *High-Performance Graphics*, 59–68.

FUCHS, H., POULTON, J., EYLES, J., GREER, T., GOLDFEATHER, J., ELLSWORTH, D., MOLNAR, S., TURK, G., TEBBS, B., AND ISRAEL, L. 1989. Pixel-Planes 5: A Heterogeneous Multiprocessor Graphics System using pProcessor-Enhanced Memories. In *Computer Graphics (Proceedings of SIGGRAPH 89)*, ACM, vol. 23, 79–88.

GREENE, N., KASS, M., AND MILLER, G. 1993. Hierarchical Z-Buffer Visibility. In *Proceedings of SIGGRAPH 1993*, ACM, 231–238.

GREENE, N. 1994. Detecting Intersection of a Rectangular Solid and a Convex Polyhedron. In *Graphics Gems IV*, Academic Press Professional, Inc., 74–82.

GRIBEL, C. J., DOGGETT, M., AND AKENINE-MÖLLER, T. 2010. Analytical Motion Blur Rasterization with Compression. In *High-Performance Graphics*, 163–172.

GRÜNSCHLOSS L., AND KELLER, A. 2009. (t, m, s)-Nets and Maximized Minimum Distance, Part II. In *Monte Carlo and Quasi-Monte Carlo Methods 2008*. Springer Berlin Heidelberg, 395–409.

HAEBERLI, P., AND AKELEY, K. 1990. The Accumulation Buffer: Hardware Support for High-Quality Rendering. In *Computer Graphics (Proceedings of SIGGRAPH 90)*, ACM, vol. 24, 309–318.

KELLER, A., AND HEIDRICH, W. 2001. Interleaved Sampling. In *Proceedings of the 12th Eurographics Workshop on Rendering Techniques*, Springer-Verlag, 269–276.

KLAVŽAR, S. 2006. Counting Hypercubes in Hypercubes. *Discrete Mathematics, 306*, 22, 2964–2967.

KOLLIG, T., AND KELLER, A. 2002. Efficient Multidimensional

Sampling. *Computer Graphics Forum, 21*, 3.

KOREIN, J., AND BADLER, N. 1983. Temporal Anti-Aliasing in Computer Generated Animation. In *Computer Graphics (Proceedings of ACM SIGGRAPH 83)*, ACM, vol. 17, 377–388.

LOVISCACH, J. 2005. Motion Blur for Textures by Means of Anisotropic Filtering. In *Rendering Techniques 2005*, 105–110.

MCCOOL, M. D., WALES, C., AND MOULE, K. 2001. Incremental and Hierarchical Hilbert Order Edge Equation Polygon Rasterization. In *Graphics Hardware*, 65–72.

MCCORMACK, J., AND MCNAMARA, R. 2000. Tiled Polygon Traversal using Half-Plane Edge Functions. In *Graphics hardware*, 15–21.

MCGUIRE, M., ENDERTON, E., SHIRLEY, P., AND LUEBKE, D. 2010. Real-Time Stochastic Rasterization on Conventional GPU Architectures. In *High Performance Graphics*, 173–182.

MOREIN, S. 2000. ATI Radeon HyperZ Technology. In *Graphics Hardware, Hot3D Proceedings*.

MUNKBERG, J., AND AKENINE-MÖLLER, T. 2011. Backface Culling for Motion Blur and Depth of Field. *journal of graphics, gpu, and game tools (to appear)*.

NIEDERREITER, H. 1992. *Random Number Generation and Quasi-Monte Carlo Methods*. SIAM.

OLANO, M., AND GREER, T. 1997. Triangle Scan Conversion using 2D Homogeneous Coordinates. In *Graphics Hardware*, 89–95.

PINEDA, J. 1988. A Parallel Algorithm for Polygon Rasterization. In *Computer Graphics (Proceedings of SIGGRAPH 88)*, ACM, vol. 22, 17–20.

RAGAN-KELLEY, J., LEHTINEN, J., CHEN, J., DOGGETT, M., AND DURAND, F. 2011. Decoupled Sampling for Graphics Pipelines. *ACM Transactions on Graphics (to appear), 30*, 3.

A Derivation of Our Generator Matrices

We base our sampling method on the $(0, m, 3)$-net proposed by Grünschloß and Keller [2009]. Their first two generator matrices, giving the Larcher-Pillichshammer (LP) points, are defined as:

$$C_1 = \begin{pmatrix} 0 & & 1 \\ & \cdot^{\cdot^{\cdot}} & \\ 1 & & 0 \end{pmatrix} \quad \text{and} \quad C_2 = \left(\begin{cases} 1 & \text{if } k \le l, \\ 0 & \text{else} \end{cases} \right)_{k,l=1}^{m}, \tag{13}$$

and the third component is generated by the matrix:

$$C_3 = \left(\binom{l}{k} \bmod 2 \right)_{k,l=1}^{m}. \tag{14}$$

To make the generated point set better suited for motion blur rasterization with our algorithm, we propose a new construction based on the matrices $C_i' = C_i D$, where $D = C_3^{-1} C_1$. D is chosen so that $C_3' = C_3 C_3^{-1} C_1 = C_1$, which generates points ordered in the third component, while keeping the first two components as the LP-points. Note that C_3 is an upper triangular matrix as $\binom{l}{k} = 0$ if $k > l$, and its determinant is thus the product of the diagonal entries, $\det(C_3) = \prod_{i=1}^{m} \binom{i}{i} = 1$, which shows it is of full rank and therefore invertible.

We start by computing the inverse C_3^{-1}. Note that C_3 is closely related to the Pascal matrix $P_n(i,j) = \binom{i-1}{j-1}, 1 \le i,j \le n$. Its inverse, P_n^{-1} is known [Brawer and Pirovino 1992] and has elements $P_n^{-1}(i,j) = (-1)^{i-j}\binom{i-1}{j-1}$. In $\mathbb{F}_2$, $-1 = 1$, and hence the term $(-1)^{i-j}$ disappears and $P_n^{-1} = P_n \pmod 2$. The elements of C_3 are the same as the lower-right submatrix of size $m \times m$ of P_{m+1}^T mod 2. This result lead us to believe that C_3 is its own inverse in $\mathbb{F}_2$, i.e., $C_3^{-1} = C_3$. As a proof, we show that the elements of $C_3 C_3^{-1} = C_3^2 = I$ are:

$$C_3^2(k,l) = \sum_{i=1}^{m} \binom{i}{k}\binom{l}{i} \bmod 2 = \ldots = \begin{cases} 1 & \text{if } k = l, \\ 0 & \text{if } k \ne l, \end{cases} \tag{15}$$

for $1 \le k, l \le m$. First, note that nonzero terms in the sum can only occur when both binomial coefficients are nonzero, i.e., when $k \le i \le l$, due to $\binom{l}{k} = 0$ if $k > l$. Hence, in the lower left, $k > l$, all elements are zero. For $k \le l$, the sum can be expanded using a double counting combinatorial proof [Klavžar 2006]:

$$\sum_{i=1}^{m} \binom{i}{k}\binom{l}{i} = \sum_{i=k}^{l} \binom{i}{k}\binom{l}{i} = 2^{l-k}\binom{l}{k}. \tag{16}$$

In the upper right, $k < l$, $C_3^2(k,l) = 2^{l-k}\binom{l}{k} \bmod 2 = 0$, as all elements are multiples of 2. Along the diagonal, $k = l$, the sum reduces to $2^0\binom{k}{k} = 1$, which concludes the proof. Finally, our new matrix, C_1', for computing the first component of the points is given by:

$$C_1' = C_1 D = C_1 C_3 C_1 = \left(\binom{m+1-l}{m+1-k} \bmod 2 \right)_{k,l=1}^{m}, \tag{17}$$

as pre and post-multiplication by the *exchange* matrix, C_1, results in flipping C_3 vertically and horizontally. To compute, C_2', we start with $C_2 C_3$, which is given by:

$$C_2 C_3(k,l) = \ldots = \begin{cases} \sum_{i=k}^{l} \binom{l}{i} \bmod 2 & \text{if } k \le l, \\ 0 & \text{if } k > l, \end{cases} \tag{18}$$

as the rows of C_2 are zero for columns $i < k$, and $\binom{l}{i} = 0$ if $i > l$. Through experiments, we have found that $\sum_{i=k}^{l} \binom{l}{i} = \binom{l-1}{k-1} \pmod 2$, $k \le l$. The formal proof of this can be found by deduction. Finally, C_2' is given by flipping this matrix horizontally, i.e., replacing the column index l by $m+1-l$, as follows:

$$C_2' = C_2 D = C_2 C_3 C_1 = \left(\binom{m-l}{k-1} \bmod 2 \right)_{k,l=1}^{m}. \tag{19}$$

This completes the derivation of our new set of generator matrices.