Data-Parallel Rasterization of Micropolygons with Defocus and Motion Blur

Kayvon Fatahalian* Edward Luong* Solomon Boulos* Kurt Akeley† William R. Mark‡ Pat Hanrahan*

Abstract

Current GPUs rasterize micropolygons (polygons approximately one pixel in size) inefficiently. We design and analyze the costs of three alternative data-parallel algorithms for rasterizing micropolygon workloads for the real-time domain. First, we demonstrate that efficient micropolygon rasterization requires parallelism across many polygons, not just within a single polygon. Second, we produce a data-parallel implementation of an existing stochastic rasterization algorithm by Pixar, which is able to produce motion blur and depth-of-field effects. Third, we provide an algorithm that leverages interleaved sampling for motion blur and camera defocus. This algorithm outperforms Pixar's algorithm when rendering objects undergoing moderate defocus or high motion and has the added benefit of predictable performance.

1 Introduction

Cinematic-quality rendering demands a high fidelity representation of complex surfaces. Smooth objects with high curvature or highly detailed objects, such as those that are rough or bumpy, cannot be faithfully represented by coarse polygonal meshes. Traditionally, the performance demands of real-time graphics have constrained the geometric complexity of scenes to be low. Most games work within a scene budget of less than a few hundred thousand polygons and rely on complex texturing, such as bump and normal mapping, to compensate for missing geometric detail. In contrast, high quality offline rendering systems, such as Pixar's RenderMan [Cook et al. 1987], represent complex surfaces accurately using *micropolygons* that are less than one pixel in size. It is common for a single offline frame to consist of hundreds of millions of micropolygons.

Barriers to increasing the geometric complexity of real-time scenes exist both within and outside the graphics pipeline. For example, high resolution geometry must be stored, animated, simulated, and transmitted to GPU computational units each frame. Within the pipeline, the extra geometry must be transformed and rasterized. However, the computational throughput of both CPU-side and GPU shader processing is rising dramatically with increasing core counts. Modern GPUs can dynamically shift computational resources between pipeline stages to accommodate increasing vertex loads, and include rapidly maturing support for programmatic geometric tessellation within the pipeline. These trends indicate that it will be feasible for a real-time system to provide micropolygon inputs to the graphics pipeline rasterizer in the near future.

This paper considers the problem of rasterizing micropolygons in a real-time system. Rasterizers in existing systems are highly tuned for polygons that cover tens of pixels; however they are inefficient for micropolygon workloads. We propose and evaluate three alternative algorithms for data-parallel micropolygon rasterization. The first considers only stationary geometry that is in perfect focus, and parallelizes across micropolygons, rather than across samples for a single micropolygon. The second is a data-parallel implementation of a previously published method by Pixar that supports motion blur and camera defocus effects. Our third implementation leverages interleaved sampling to decouple rasterization cost from scene characteristics such as motion or defocus blur.

*Stanford University: kayvonf, edluong, boulos, hanrahan @graphics.stanford.edu

†Microsoft Research: kakeley@microsoft.com

‡Intel Corporation: william.r.mark@intel.com

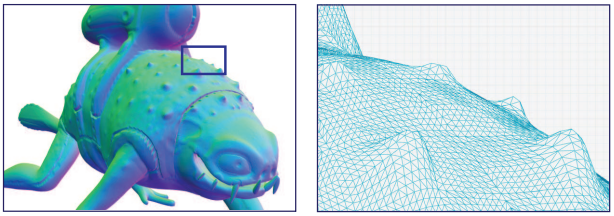

Figure 1: *Complex surfaces such as this frog's skin are represented accurately using micropolygons.*

2 Background

2.1 Traditional Rasterization

Whether implemented as fixed-function hardware or an optimized software implementation, computing the sample points covered by a polygon can be broken down into three major steps: performing per-polygon preprocessing ("polygon setup"), determining a conservative set of possibly-covered sample points, and performing individual point-in-polygon tests.

Setup encapsulates computations, such as clipping and computing edge equations, that are performed once per polygon and decrease the cost of the individual point-in-polygon tests. When polygons are large, the results of a single setup operation are reused for many sample tests. Setup need not be widely parallelized as its cost is amortized over many tests.

Rasterization algorithms use point-in-polygon tests to determine which screen sample points lie inside a polygon. Testing samples that lie outside the polygon is wasteful, because the polygon does not contribute to the image at these locations. We quantify this waste by considering the *sample test efficiency* (STE) of a rasterization scheme: the percentage of point-in-polygon tests that result in hits. Modern rasterizers compute polygon overlap with coarse screen tiles [Fuchs et al. 1989; McCormack and McNamara 2000] or use hierarchical techniques [Greene 1996; McCool et al. 2001; Seiler et al. 2008] that utilize multiple tile sizes as a means to efficiently identify a tight candidate set of samples.

Finally, samples must be tested to determine if they are inside the polygon. Modern rasterizers leverage efficient data-parallel execution by testing a block of samples (a "stamp") against a single polygon in parallel. These tests can be carried out using many execution units to achieve high throughput. This approach was introduced in Pixel Planes [Fuchs et al. 1985] which tested all image samples against a polygon in parallel. Other implementations use tile sizes ranging from 4x4 to 128x128 samples [Pineda 1988; Fuchs et al. 1989; Seiler et al. 2008]. Modern GPU rasterizers simultaneously perform as many as 64 simultaneous sample tests using data-parallel units [Houston 2008].

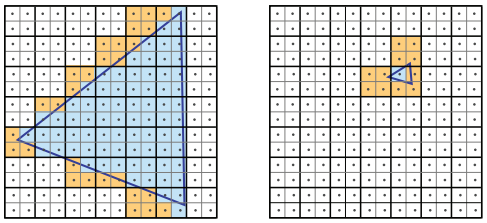

Figure 2: *Rasterization of polygons using a 2x2 sample stamp. Samples within the polygon are shown in blue. Samples tested during rasterization, but not covered by the polygon, are yellow. The micropolygon's area is small compared to that of the 2x2 stamp, resulting in low STE.*

In summary, modern rasterizers rely on per-polygon preprocessing, coarse-grained rejection of candidate samples, and wide data-parallelism for polygon-sample coverage tests. Unfortunately, these design decisions lead to inefficient implementations when polygons shrink to subpixel sizes. First, micropolygon scene representations contain tens of millions of polygons. The frequency (and therefore expense) of setup operations increases dramatically and is no longer amortized over many sample tests. Setup must be minimized or parallelized when possible. Second, hierarchical schemes for computing candidate sample sets are unnecessary. A micropolygon's screen bounding box describes a tight candidate set. Last, large stamp sizes are inefficient because the screen area covered by a block of samples is significantly larger than a micropolygon. The inefficiency of a 2x2 sample stamp (tiny by modern GPU standards) is illustrated in Figure 2, which highlights samples tested against (yellow) and covered by (blue) two polygons. STE is high for the polygon at left, but drops to 8% for the micropolygon at right. Large raster stamps give up efficiency near polygon edges in exchange for efficient data-parallel execution. When rendering micropolygons, all candidate samples are near a polygon edge.

2.2 Defocus and Motion Blur

Camera defocus and motion blur are commonplace in offline rendering but used sparingly in real-time systems due to the high cost of integrating polygon-screen coverage in space, time, and lens dimensions. To simulate these effects, we follow previous approaches [Cook et al. 1987; Akenine-Möller et al. 2007] and estimate this integral by stochastically point sampling [Cook et al. 1984; Cook 1986] polygons in 5-dimensional space (screen XY, lens position UV, time T). This presents two major challenges for high performance implementation. First, it is hard to localize a moving, defocused polygon in 5D, so generating a tight set of candidate sample points (maintaining high STE) is challenging. Second, performing point-in-polygon tests in higher dimensions is expensive. We focus on the first of these challenges in this paper.

There is also a large body of work that seeks to approximate these effects by post processing rendered output (see [Sung et al. 2002] and [Demers 2004] for a summary of motion blur and defocus blur approaches respectively). These approaches work well in some regions of a frame but often produce artifacts. While there will always be a use for fast approximations, we feel it is important to improve the performance of direct 5D integration.

2.3 Micropolygon Rendering Pipeline

We conduct our study of micropolygon rasterization in the context of a complete micropolygon rendering pipeline influenced by the design of the REYES rendering architecture [Cook et al. 1987]. Our pipeline accepts parametric surface patches as input and tessellates these patches into micropolygons. In contrast to modern GPUs, but like REYES, our system performs shading computations prior to rasterization at micropolygon vertices. A micropolygon-sample "hit" generates a fragment that is immediately blended into the render target (it does not undergo further shading).

Performing shading computations at micropolygon vertices is not fundamental to our study of micropolygon rasterization. However, micropolygon rasterization is tightly coupled to the properties of the pipeline's tessellation stage. As discussed later in this paper, rasterization performance relies heavily on adaptive tessellation that yields micropolygons that are approximately uniform in area, aspect ratio, and orientation. We follow the design of REYES and rely on tessellation to cull micropolygons that fall behind the eye plane, negating the need for expensive clipping prior to rasterization. We are simultaneously conducting a detailed study of high quality, adaptive tessellation in the context of a real-time system.

3 Algorithms

We introduce three algorithms for micropolygon rasterization. The first algorithm ignores defocus and motion blur effects. The second and third perform full 5D rasterization, yielding images containing both camera defocus and motion blur. We describe algorithmic issues of rasterizing micropolygons in this section, and defer the data-parallel formulation of each algorithm to Section 4.

3.1 2D Rasterization

Our algorithm for 2D rasterization (no motion blur, no defocus), NOMOTION (shown below), omits many optimizations common to modern rasterizers. NOMOTION does not use coarse or hierarchical rejection methods to compute a tight set of candidate samples. It directly computes an axis-aligned bounding box and tests all samples within this bound.

```
Cull backfacing
Compute edge equations [optional]
BBOX = Compute MP bbox
for each sample in BBOX
  test MP against sample
```

It is possible to avoid explicit precomputation and storage of edge equations by conducting point-in-polygon tests in a coordinate system that contains a sample point at the origin. These tests are slightly more expensive than tests using explicit edge equations, so edge precomputations in setup remain beneficial for larger micropolygons or high multi-sampling rates. NOMOTION also performs backface culling in micropolygon setup. This check is inexpensive compared to the cost of bounding and hit testing a micropolygon, and it can eliminate half a scene's hit testing work.

The size of a bounding box and, correspondingly, the STE of NOMOTION, is sensitive to the area, aspect ratio, and orientation of micropolygons. It is important for upstream tessellation systems to generate "good" micropolygons to maximize STE. On average, a quadrilateral fills the area of its bounding box better than a triangle, so NOMOTION benefits from directly processing both triangle and quadrilateral micropolygons. Tricky edge cases introduced by quads, such as concave or bow-tie polygons, produce only sub-pixel artifacts and can be treated with less rigor than when polygons are large. We perform a point-in-quadrilateral test by splitting the quad along the diagonal connecting vertices zero and two, then performing two point-in-triangle operations using five edge tests.

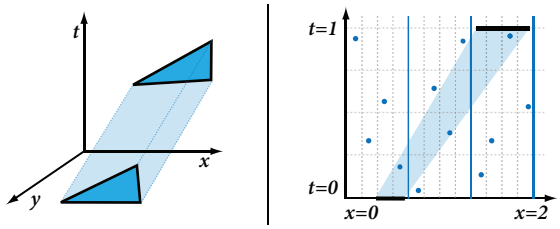

Figure 3: *Left: A polygon moving through (XY,T) space with linear motion. Right: A simplified illustration showing only one spatial dimension (X,T plane). Sample points are stratified in space and time. Only points lying inside the shaded region result in hits.*

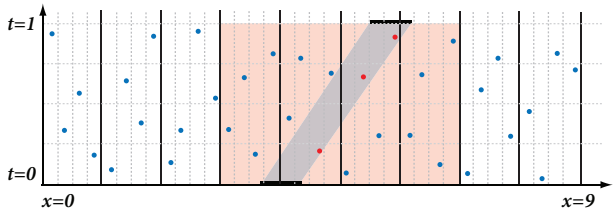

Figure 4: *Testing all samples within the spatial bounding box of a moving polygon results in low STE. Point in polygon tests are performed against all samples in the orange region. Only three of these samples (shown in red) are covered by the polygon.*

3.2 5D Rasterization: Motion and Defocus Blur

The NOMOTION algorithm finds all samples in screen space (2D XY space) that lie within a micropolygon. To render geometry with motion blur, we must find all samples in a 3D (XY,T) space that lie within the volume swept out by a moving polygon. Figure 3 visualizes a moving triangle in 3D (XY,T) space. A simplified view showing only the X spatial dimension is shown at right in the figure. The illustration shows the object's motion in (X,T) space during an exposure from t=0 to t=1. Samples within the shaded area (a total of three) are covered by the polygon. That is, at the time value associated with the sample, the micropolygon is covering the 2D screen position of the sample. In general, rasterizing a defocused, motion blurred polygon involves finding all samples in a 5D (XY,UV,T) space that fall within the volume representing the blurred polygon.

If the object in Figure 3 were moving faster, the shaded region would be more slanted, but the area of the region would remain the same. Because the number of samples covered is proportional to object area, the expected number of samples covered by a moving object is the same as if it was stationary (assuming its size does not change greatly during motion). Thus, for motion blurred rasterization to achieve STE similar to the stationary case, it must test approximately the same number of samples.

Recall that obtaining high STE for stationary polygons requires computing a tight spatial bound. Previous work transfers this idea to the 3D (XY,T) domain by bounding a triangle over an entire interval of time using either an axis-aligned or an oriented bounding box [Akenine-Möller et al. 2007]. All samples within the bound are tested against the polygon, even if the object is at a distant location at the time associated with a sample. Figure 4 highlights the samples tested by this scheme in orange. Because a rapidly moving polygon crosses a large region of space during the shutter interval, this approach can result in a significant increase in total sample tests. The problem is acute for micropolygons as even small object motion is significant in relation to their area; it is not uncommon for one-pixel micropolygons to move 30 pixels between frames.

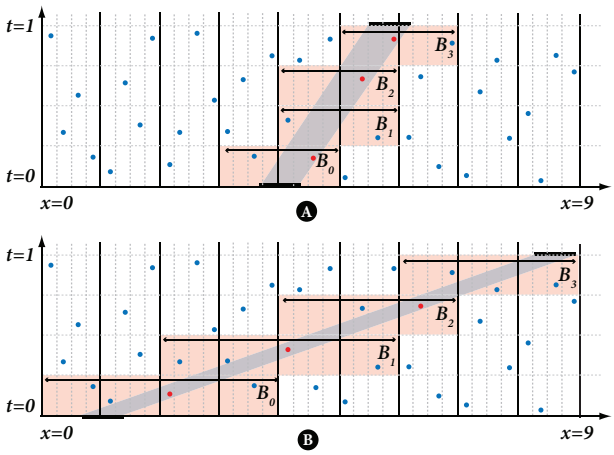

Figure 5: INTERVAL *uniformly partitions time into intervals, then bounds the spatial extent of the polygon in each interval. For each of the four intervals shown above, sample points lying within the time interval and within the polygon's spatial extent are tested against the polygon. Spatial bounds, B_i, are tight when a polygon is moving slowly (A) but loosen as the amount of motion increases (B).* INTERVAL *is most efficient for slow moving polygons.*

3.2.1 Interval Method

An approach described by Pixar [Cook et al. 1990] leverages stratified 5D sampling to quickly compute a tighter candidate sample set. We describe this algorithm under conditions of only motion blur (the (XY,T) case) before generalizing to full 5D rasterization.

Pixar's approach generates a unique set of S stratified time values for each region of space (the published embodiment generates S stratified samples per pixel). Stratification partitions the time domain into S intervals. For each micropolygon, the algorithm iterates over all intervals, computing the bounding box of the micropolygon for a given interval of time. Given this bound for each interval, it tests only the samples that fall within the interval's spatial extent. There is exactly one such sample per pixel due to the stratified nature of the samples. Because this algorithm iterates over intervals of time (and, in the 5D case, intervals of time and lens samples), we refer to it as INTERVAL. Pseudocode for INTERVAL is given below:

```
for each STRATUM   // S iterations
   BBOX = compute mp pixel bbox given STRATUM
   for each pixel P in BBOX
      test mp against sample from STRATUM in P
```

The behavior of INTERVAL in the (X,T) plane is illustrated in Figure 5. INTERVAL's STE depends on the spatial bound, B_i, of the polygon over the time range associated with each stratum. Therefore, STE depends on object velocity. When the polygon is moving slowly (Figure 5A), INTERVAL yields high STE as the polygon can be tightly localized in space (B_i's are small). For a stationary object, INTERVAL behaves similarly to NOMOTION. STE decreases as object motion becomes large (Figure 5B). For example, an object streaking across the screen can decrease the STE of INTERVAL sharply. Notice that the STE of INTERVAL not only depends on the magnitude of motion, but its direction (horizontal or vertical motion produce tighter bounds than diagonal motion).

INTERVAL extends gracefully to full 5D (XY,UV,T) rasterization by pairing UV and T strata. When constructing the 5D position of samples, INTERVAL always pairs values from the same UV stratum

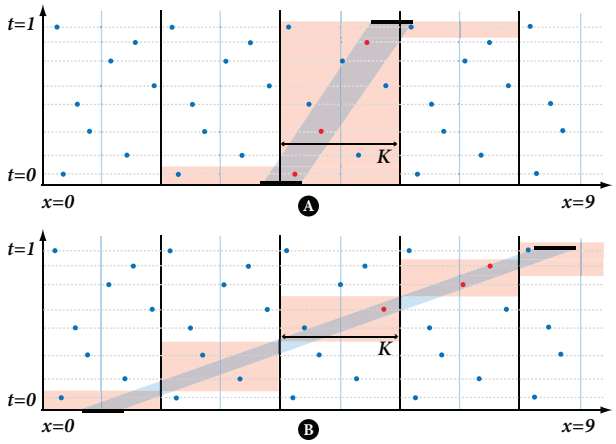

Figure 6: INTERLEAVEUVT *performs a separate rasterization step for each of N unique time values in the frame buffer (indicated by horizontal dotted lines). The micropolygon's position is determined exactly at these times, so STE is independent of object velocity. Approximately the same number of tests are performed against the slow (A) and fast (B) moving polygons. Solid lines in the figure indicate interleaving tile boundaries.*

with values from the same T stratum. The association of ranges in T and UV can be constructed in any manner, but is the same for all samples in the image. Because of this property, the range of sample UV and T values for the current stratum (outer loop of algorithm) is immediately computable given a stratum index. INTERVAL uses these bounds on sample UV and T values to bound the polygon spatially. It places no other constraints on the properties of UV and T values used or on the location of XY samples.

3.2.2 Interleave UVT Method

There are two properties of INTERVAL we seek to improve. First, we seek to increase efficiency under conditions of high motion and defocus blur, especially at lower multi-sampling rates typical of interactive graphics (4x-16x MSAA). Second, real-time systems benefit from a predictable frame rate, so it is desirable to decrease the sensitivity of rasterization performance to object velocity or defocus blur (object velocity, in particular, is difficult for a game designer to constrain).

Our solution achieves these goals using interleaved sampling [Mitchell 1991; Keller and Heidrich 2001] in the UV and T dimensions. The key idea of this approach is that every image sample point is assigned one of N unique UVT tuples $uvt_i = (u_i, v_i, t_i)$. Following Keller and Heidrich [2001] we consider the image as a grid of tiles each containing N sample points covering a K_x by K_y region of the screen. Within a tile, each sample point is assigned a unique tuple uvt_i. Thus, each of the N tuples is used exactly once per tile and all samples located at t_i in 5D space are also located at u_i and v_i. We exploit this property of the interleaved sampling pattern in the following algorithm for 5D rasterization (referred to as INTERLEAVEUVT):

```
for each unique UVT tuple (ui,vi,it)   // N iterations
   MP_POSITION = compute mp position at ui,vi,ti
   BBOX = compute tile-grid bbox from MP_POSITION
   for each TILE in BBOX
      test mp against 5D sample in TILE associated
      with tuple (ui,vi,ti)
```

The behavior of INTERLEAVEUVT in the simplified (X,T) case is illustrated in Figure 6. Notice that all image sample points are assigned one of only eight unique times (N=8). This assignment is

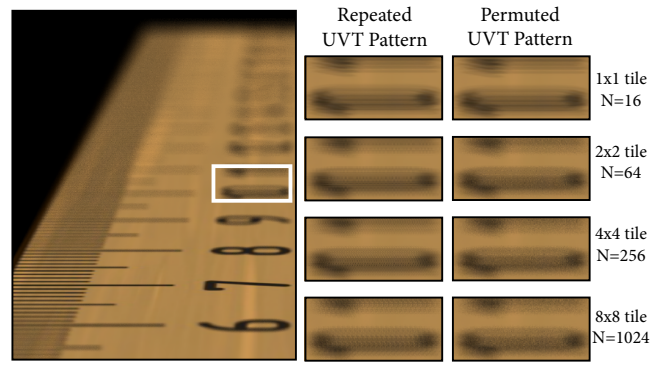

Figure 7: *Repeating the same pattern of UVT values over the image results in sampling artifacts that become more noticeable with increasing tile size (left column). We remove these artifacts by permuting the position of UVT values in each screen tile (right).*

repeated every two units in space (K_x=2). For each unique time, INTERLEAVEUVT computes the *exact* position of the micropolygon at the time (this is not a bound over an interval of time) and determines polygon overlap with screen tiles (our implementation uses axis-aligned bounding boxes to compute overlap with tiles). The micropolygon is then tested against the one sample within each overlapped tile that is located at the current time. This approach is similar to the accumulation buffer [Haeberli and Akeley 1990], except each accumulated image involves $1/N$ the samples of the final rendered image.

The STE of INTERLEAVEUVT is independent of the motion or defocus blur of rasterized geometry. Any micropolygon, regardless of whether it undergoes slow or fast motion, is tested against at least N samples (outer loop of INTERLEAVEUVT involves N iterations). In practice, a polygon will be tested against more than N samples due to potential overlap of multiple tiles for each tuple. On average, this overlap depends only on the size of the micropolygon and tiles.

The parameter N serves as a performance-quality knob for INTERLEAVEUVT. Large N yields potentially higher sampling quality (more unique time and lens values are used) at the expense of increased rasterization cost. Note that the spatial extent (K_x by K_y) of tiles is fixed once a cost budget of N tuples and a desired multi-sampling rate is selected. Said differently, for a given multi-sampling rate, the STE of INTERLEAVEUVT is directly proportional to the size of the interleaving tile.

We call attention to the relationship between INTERVAL's per-strata polygon spatial bounds (B_i in Figure 5) and INTERLEAVEUVT's tile size (K_x in Figure 6). Intuitively, the STE of INTERVAL and INTERLEAVEUVT can be compared by comparing the average B_i to K_x. For example, INTERLEAVEUVT is more efficient when the tile size is small in comparison to the amount of polygon translation over an interval of time. In practice, comparing B_i to K_x provides only a coarse estimate of STE (and overall cost) because a polygon may overlap multiple tiles. A detailed comparison of cost is provided in Section 5.

3.2.3 Sample Permutations

Repeating a tile of UVT values across the image (as illustrated in Figure 6) yields a sampling pattern where points assigned the same UVT value form a regular grid in XY space. This regularity yields sampling artifacts even if N is large enough to prevent strobing. Figure 7 (middle column) shows artifacts resulting from repeating 1x1, 2x2, and 4x4, and 8x8 pixel tiles of across the image. Artifacts

from repeating the larger tile patterns are arguably more objectionable because they are lower frequency.

We improve image quality without increasing N by varying the XY position of a UVT tuple in each tile. Our approach permutes the mapping of UVT values to tile-relative XY positions on a per tile basis (we associate each tile with one of 64 precomputed permutations). The permutations preserve stratification of UVT values within each pixel. There remains exactly one sample per image tile located at UVT_i, but these samples no longer form a regular grid in the image.

The zoomed images in the right column of Figure 7 show the effect of these permutations. Although the same tile size and number of UVT tuples is used, artifacts visible in the center column are replaced by less objectionable noise. Our current implementation permutes only the pixel offset of a UVT tuple within a tile (the same subpixel offset is always used for each UVT_i). This simplification produces satisfactory results, simplifies implementation, and permits compact representation of permutations (a permutation for a 2x2 pixel tile is encoded using only four, four bit numbers). We have not rigorously studied how to optimally construct mappings of UVT values to XY positions.

4 Implementation

In this section we describe generic data-parallel implementations of NOMOTION, INTERVAL, and INTERLEAVEUVT that are used to evaluate performance in Section 5. We do not target any specific architecture with our implementations. Instead, we assume an abstract set of data-parallel operations. In addition to standard floating-point and integer operations, this set includes memory scatters and gathers, and the ability to manipulate vector bit-masks.

Figure 8 formats pseudo-code for the three algorithms to highlight important similarities in structure. Each algorithm (optionally) begins with a small amount of per-micropolygon setup (*Setup*). All algorithms then bound the micropolygon at various points in time or aperture (*Bound*) and iterate over sets of samples within these bounds (*Test*). When samples are covered by the micropolygon, fragments are generated for downstream frame buffer processing (*ProcessHit*). We focus only on the costs directly associated with computing coverage and generating fragments and do not consider the costs of frame buffer operations.

There are two sources of conditional control flow common to all algorithms. First, iteration over candidate samples within a micropolygon bound depends on the micropolygon's spatial extent. Second, only samples covered by the micropolygon trigger fragment processing. When STE is low (as is the case, in particular for the full 5D algorithms), fragment processing will occur infrequently in comparison to sample tests: an STE of 10% will lead to $10\times$ as many sample tests as generated fragments.

4.1 No Motion

A micropolygon covers only a few image sample points when rasterized with low multi-sampling. Therefore, there is insufficient per-micropolygon sample testing work to utilize many parallel execution units. There is, however, no shortage of micropolygons, so we parallelize NOMOTION by operating on multiple micropolygons in parallel. Our implementation takes parallelism along this axis to its extreme. Given W data parallel execution units, we perform the same operation simultaneously on W unique micropolygons. This implementation does not perform work associated with a single polygon in parallel so computing tight subpixel bounding boxes (bounds are clamped to strata boundaries to maximize STE) does not decrease available parallelism. Parallelizing across micropolygons also results in a data-parallel implementation of setup.

Our parallelization scheme is susceptible to load imbalance across execution units because the number of samples tested against each micropolygon varies. Utilization during *Test* depends heavily on the ability of upstream pipeline stages to tessellate surfaces into micropolygons that are uniform in size and similar in orientation and aspect ratio.

Parallelizing across micropolygons introduces a number of implementation costs not present in raster-stamp-based approaches. First, in NOMOTION, fetching sample data requires a memory gather. In contrast, sample data for a stamp is obtained using aligned loads. Second, our scheme incurs increased memory footprint because data for W micropolygons must be kept accessible at once. However, we assume micropolygon shading occurs prior to the rasterization stage of the pipeline, so we only require position and color data at each vertex. Last, processing many micropolygons in parallel places a burden on downstream frame-buffer processing to reestablish the required order of generated fragments. Our micropolygon rasterizer does not generate large blocks of logically independent fragments. Fragments generated from micropolygons processed in parallel are mixed in the output stream.

We compute fragment color and depth in *ProcessHit* via smooth interpolation of per-vertex values. We have observed that when micropolygons are very small in size (near the Nyquist rate: 0.5 pixel edge length), fragment color can be computed using flat shading without producing visible artifacts. This optimization reduces the cost of *ProcessHit* by 38% and reduces storage requirements for micropolygon data. The edges of micropolygons used in our renderings are approximately one pixel in length, so the results discussed in Section 5 do not include this optimization.

4.2 Interval and Interleave UVT

INTERVAL and INTERLEAVEUVT test each micropolygon against a large set of candidate samples due to the difficulty of tightly bounding polygons in 5D space. As a result, unlike NOMOTION, many candidate samples can be tested against a single moving or defocused micropolygon in parallel.

INTERVAL's outer loop over intervals and INTERLEAVEUVT 's loop over UVT tuples (highlighted in blue in Figure 8) do not involve dynamic loop bounds. The number of intervals or UVT tuples is a property of the sampling scheme, so the number of iterations through these loops is micropolygon invariant. We select sampling rates such that loop bounds (S or N) are multiples of the data-parallel width of the machine and parallelize across micropolygon-sample coverage tests involving different intervals or UVT tuples.

Because the number of unique UVT tuples used by INTERLEAVEUVT is large (we use N=64 unique tuples in our final results), parallelization across tuples can make full use of many execution units while considering only a single micropolygon at a time. Values of S used in practice are smaller, so we achieve wide data-parallelism by running INTERVAL on a few micropolygons simultaneously when $W < S$. For example, to run INTERVAL with S=16 on a platform with 64 data-parallel execution units, we process four micropolygons at a time.

In both algorithms, utilization during *Test* depends on the variance of the dynamic loop bounds. For example, if all interval bounding boxes contain the same number of samples, *Test* will run at full utilization. The same result applies for INTERLEAVEUVT if the micropolygon overlaps the same number of tiles when positioned for each UVT tuple. Utilization decreases with increasing *Test* loop

	NoMotion (2D)	Interval (5D)	InterleaveUVT (5D)
	`for each MP`	`for each MP`	`for each MP`
Setup	`Backface cull // 9 ops`	`Backface cull // 17 ops`	`Backface cull // 17 ops`
Bound		`for each interval // S iters`	`for each UVT tuple // N iters`
			`Position MP given UVT // 36 ops`
	`Compute MP bbox // 29 ops`	`Compute MP bbox for interval // 70 ops`	`Compute MP tile bbox // 27 ops`
Test	`for all samples in bbox`	`for all samples in interval bbox`	`for all tiles in bbox`
	`Compute sample XY // 29 ops`	`Compute sample XYUVT // 33 ops`	`Compute sample XY // 23 ops`
		`Position MP given UVT // 36 ops`	
	`Test MP-sample coverage // 24 ops`	`Test MP-sample coverage // 24 ops`	`Test MP-sample coverage // 24 ops`
Process Hit	`If sample covered`	`If sample covered`	`If sample covered`
	`Interpolate color,Z // 31 ops`	`Interpolate color,Z // 31 ops`	`Interpolate color,Z // 31 ops`
	`Push to fragment queue`	`Push to fragment queue`	`Push to fragment queue`

Figure 8: *Our three rasterization algorithms formatted to show similarities in structure. We parallelize execution over iterations of loops highlighted in blue. Additional parallelization across micropolygons in* INTERVAL *occurs when there are fewer intervals than data-parallel execution units. Dynamic control flow exists in loops over samples (Test) and due to conditional processing of sample hits (ProcessHit).*

bound variance because units done with testing work wait idle until the micropolygon with the most work finishes.

Operation counts for the INTERVAL and INTERLEAVEUVT algorithms in Figure 8 correspond to our implementation in the full 5D sampling case, where both motion blur and defocus blur effects are produced. If only motion blur or defocus is desired, operations such as backface culling or positioning the micropolygon at the UVT position of a sample can be simplified. For example, a micropolygon's normal is not affected by defocus approximations, so performing backfacing checks on a defocus blurred micropolygon is the same as for 2D rasterization. The normal of a motion blurred micropolygon may flip while the shutter is open, so motion blur requires backfacing checks at both the beginning and end of the shutter interval.

We compute a micropolygon's position at a given time via linear interpolation of positions at the start and end of the shutter interval. In the full 5D sampling case, once the micropolygon has been positioned for a given time, circle of confusion radii are computed and its vertices are shifted according to the lens sample UV. When motion is not present, the circle of confusion can be computed as part of micropolygon setup and reused in each sample test.

5 Evaluation

We evaluate the rasterizer implementations presented in Section 4 in three primary areas. First, we characterize the overall cost of micropolygon rasterization as well as the breakdown of work among the four algorithm "phases". Second, we measure the impact of conditional execution on utilization of data-parallel execution units. Last, we measure the cost of enabling support for motion blur and camera defocus by studying the relative performance of INTERVAL and INTERLEAVEUVT under varying amounts of scene motion and defocus.

We perform experiments using the four animated scenes shown in Figure 9. All animations contain motion blur and are rendered at 1728×1080. TALKING also incorporates a defocus effect to draw viewer attention to the character facing the camera (TALKING requires full 5D sampling). Our rendering pipeline tessellates scene geometry, represented as Catmull-Clark patches, into micropolygons whose vertices are spaced by approximately one pixel on screen. Shading is performed prior to rasterization, on 2D grids of vertices similar to [Cook et al. 1987]. After shading, the rasterizer receives a stream of triangle micropolygons. Two triangles are created from the vertices forming each 2D grid cell, yielding triangles that are, on average, 0.5 pixels in area. All experiments are performed with backface culling enabled.

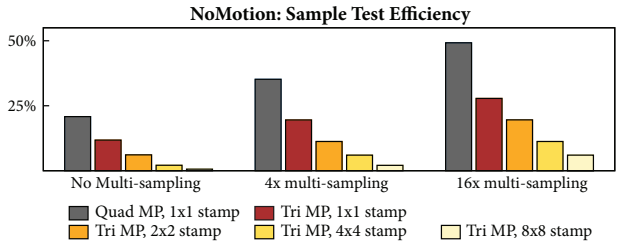

Figure 10: NOMOTION *(grey and red bars) processes many micropolygons simultaneously but tests micropolygons against a single sample at a time. Large raster stamps yield low STE.*

To conduct detailed analysis of algorithm performance, we created a library of fixed-width data-parallel operations. Library instrumentation counts the number of operations performed as well as the utilization of execution units by each operation (utilization is determined by explicitly-programmed masks). Using this library we implement versions of each algorithm that utilize 8, 16, 32, and 64-wide operations. In this evaluation, references to operation count refer to 16-wide operations unless otherwise stated.

5.1 No Motion

Figure 10 quantifies the STE benefit of parallelizing rasterization across micropolygons, rather than candidate samples for a single polygon. Recall that NOMOTION effectively uses a raster stamp size of one sample (1x1 stamp, red bars). When multi-sampling is disabled, the screen area of this "stamp" is already larger than a micropolygon (0.5 pixel area). Testing stamps of 4, 16, and 64 samples against a single micropolygon at once results in low STE, as illustrated by the gold bars in the figure. In our scenes, rasterizing micropolygons onto a 4x multi-sampled frame buffer (a common design point for modern GPUs) using an 8x8 sample stamp results in 2% STE. In contrast, NOMOTION's single sample stamp yields 18% STE under this configuration (on average, a micropolygon bounding box covers approximately 11 sample strata). Although NOMOTION's STE for triangle micropolygons increases to 26% under 16x multi-sampling, we note that the STE of rasterizing micropolygons is fundamentally lower than when polygons are large in size.

Figure 9: *Animation sequences used in algorithm evaluation from left to right:* BALLROLL, COLUMNPIVOT, TIGERJUMP, *and* TALKING. *All scenes feature motion blurred geometry.* TALKING *also features camera defocus and requires full 5D (XY,UV,T) rasterization.*

	Data-Parallel Execution Units			
	8	16	32	64
NO MULTI-SAMPLING				
Bound	.18 (.99)	.16 (.98)	.15 (.96)	.14 (.94)
Test	.67 (.77)	.65 (.71)	.64 (.65)	.63 (.60)
Process	.14 (.21)	.18 (.15)	.20 (.11)	.22 (.09)
Overall Util	.74	.66	.59	.54
Par Ops/MP	30.4	17.2	9.7	5.6
4X MULTI-SAMPLING				
Bound	.09 (.99)	.08 (.98)	.07 (.96)	.07 (.94)
Test	.69 (.80)	.67 (.74)	.66 (.67)	.66 (.62)
Process	.21 (.25)	.25 (.19)	.26 (.16)	.27 (.14)
Overall Util	.70	.62	.56	.51
Par Ops/MP	64.6	36.6	20.7	11.8
16X MULTI-SAMPLING				
Bound	.03 (.99)	.03 (.98)	.03 (.96)	.03 (.94)
Test	.71 (.83)	.69 (.76)	.69 (.69)	.68 (.63)
Process	.25 (.33)	.27 (.27)	.29 (.23)	.29 (.20)
Overall Util	.71	.63	.57	.52
Par Ops/MP	170.1	95.8	54.3	31.3

Table 1: *Fraction of total execution time and data-parallel unit utilization (in parentheses) of each stage of* NOMOTION *(Setup is negligible and not shown). Even at low multi-sampling, over 65% of operations are spent in Test. Average per-micropolygon cost (in data-parallel ops) is provided for each configuration.*

We can improve efficiency by rasterizing quadrilateral micropolygon inputs in addition to triangles. The black bars in Figure 10 show that STE can be as much as 2x greater when tessellated geometry is provided directly to the rasterizer as a stream of quadrilateral micropolygons. Quadrilateral primitives have twice the area of triangles and yield better coverage of an axis-aligned bounding box. Further, using quadrilateral micropolygons as a visibility primitive decreases the number of primitives reaching the rasterizer (we do not break cells of the tessellated grid into two triangles). We measure that directly rasterizing quadrilateral micropolygons decreases overall rasterization cost by 20%.

Table 1 breaks down the cost of each phase of NOMOTION, averaged over all scenes (*Setup* is omitted from the table as it constitutes less than 1% of execution time in all cases). Separate results for implementations using 8, 16, 32, and 64 data-parallel execution units are given. Even at low sampling rates, over 63% of execution time is spent in *Test*. We find that variance in the amount of testing work per micropolygon is low and that *Test* maintains high utilization of many data-parallel execution units. In the common case of 4x multi-sampling, *Test* sustains approximately 80% utilization when mapped onto eight data-parallel execution units. Scaling the implementation eight-fold to 64 units results in only a 18% drop in utilization. We conclude that parallelizing rasterization across micropolygons is not limited by variance in the amount of work per polygon. Maintaining polygon state (despite compact representation, 64 micropolygons require over 5 KB of storage), or the complexity of processing fragments from many micropolygons in subsequent frame buffer operations, are more likely to limit the scalability of NOMOTION.

The average cost (in units of data-parallel operations) of rasterizing a micropolygon using our implementation of NOMOTION is also given in Table 1. Although our implementation can benefit significantly from further optimization, these costs indicate that micropolygon rasterization is expensive, even in the absence of camera defocus and motion blur. For example, NOMOTION requires 11 billion 16-wide operations per second to rasterize ten million micropolygons (depth-complexity 2.5 rendering at 1728×1080 resolution) onto a 4x multi-sampled frame buffer at 30 Hz. We note that this cost is equivalent to 11 Larrabee units [Seiler et al. 2008]. These costs strongly suggest a need for fixed-function hardware acceleration.

5.2 Interval and Interleave UVT

The remainder of our evaluation focuses on rasterizing micropolygons with motion blur and defocus. Estimating time and lens integrals via stochastic sampling requires high pixel sampling rates to eliminate noise. Although the high super-sampling rates used in offline cinematic rendering will remain out of reach for real-time systems for some time, we have found that 16 samples per pixel is sufficient to reduce noise (in our opinion) to visually acceptable amounts. Further, our experiences indicate that only a small number of UVT tuples are necessary for INTERLEAVEUVT to generate compelling animations. In general, we are satisfied with INTERLEAVEUVT's output using 16x multi-sampling, and 64 unique UVT tuples (N=64, 2x2 pixel tile size). In this section, references to INTERLEAVEUVT imply N=64 unless otherwise stated. At these low sampling levels, artifacts such as temporal aliasing and noise are not fully eliminated from rendered output. We refer the reader to this paper's accompanying video to compare the output quality of various sampling configurations.

Switching rasterizer implementations from NOMOTION to INTERVAL or INTERLEAVEUVT has inherent cost, even when all scene geometry is stationary and in sharp focus. Recall from Figure 10 that NOMOTION yields 26% STE when rendering the test scenes at 16x multi-sampling. We temporarily disable object movement and defocus blur in the test scenes 9, making the visual output of all algorithms the same. Under these conditions, we find that INTERVAL's STE (10%) is twice as high as INTERLEAVEUVT's (5%), but less than half that of NOMOTION (26%). Simply enabling support for motion and defocus blur decreases STE by 2.6x. INTERVAL's STE is approximately equal to that of the 4x4 raster stamp in Figure 10 (a 4x4 sample stamp spans a 16x multi-sampled pixel in area, thus its bound as tight as the pixel bounds computed by INTERVAL). The overall cost of INTERVAL is 3.1x greater than NOMOTION. This difference is greater than the relative difference in STE because sample tests in the (XY,T) or full (XY,UV,T) sampling case cost more.

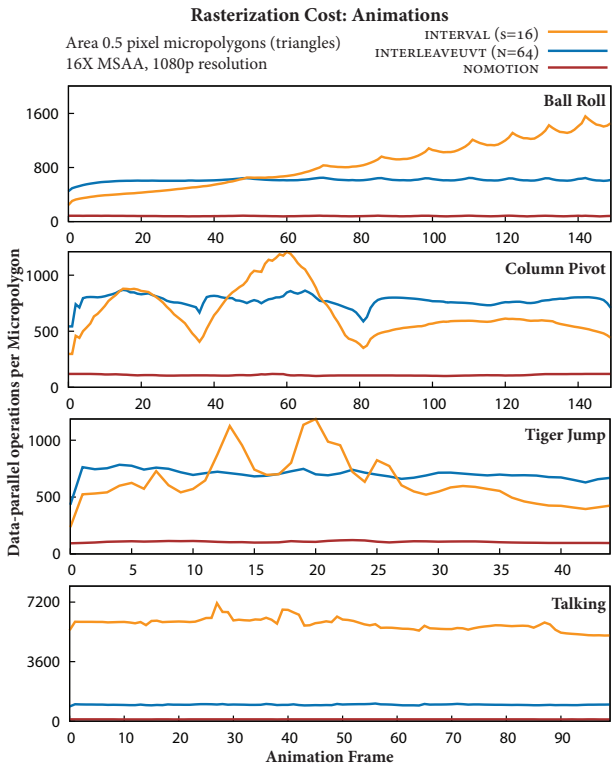

Figure 11: INTERVAL *exhibits significant variation in rasterization cost (measured in data parallel operations). In the* TIGERJUMP *animation sequence,* INTERVAL*'s per-micropolygon rasterization cost varies by nearly 4x over just 45 frames of animation.* INTERLEAVEUVT*'s cost is significantly less dependent on amount of scene motion or defocus.*

5.2.1 Animated Scene Costs

As stated in Section 3.2, as scene motion or defocus increases, we expect relative performance of INTERLEAVEUVT to improve with respect to INTERVAL. In Figure 11, we compare algorithm performance rendering each animation sequence at 1728×1080 resolution. We simulate a camera shutter that remains open for 1/48 of a second (a common exposure setting for film cameras).

As expected, INTERVAL's performance fluctuates more widely over the sequences than that of INTERLEAVEUVT. Even within a short sequence such as TIGERJUMP (1.5 sec), the performance of INTERVAL varies as much as 4x. The performance of INTERLEAVEUVT is not constant, but varies much less dramatically. We find that INTERLEAVEUVT's STE is nearly uniform over the sequences, and attribute variation in operation-count to two causes. First, INTERLEAVEUVT's utilization of parallel execution units increases moderately at low amounts of blur (this effect is discussed further in the next section). Second, under conditions of fast motion, we backface cull fewer micropolygons (polygons that begin the shutter interval back facing but flip as a result of motion cannot be culled) resulting in increased average per-micropolygon cost.

From our animations we conclude that object motion must be very fast (quickly moving hands in conversation, a camera jerk, an athletic jump) to equate algorithm performance. Although INTERVAL may introduce variation in rasterization costs, these costs are often lower than those of INTERLEAVEUVT. However, when defocus is present, INTERLEAVEUVT outperforms INTERVAL. The difference in cost can be extreme. For example, INTERLEAVEUVT is 7x faster than INTERVAL when rendering TALKING.

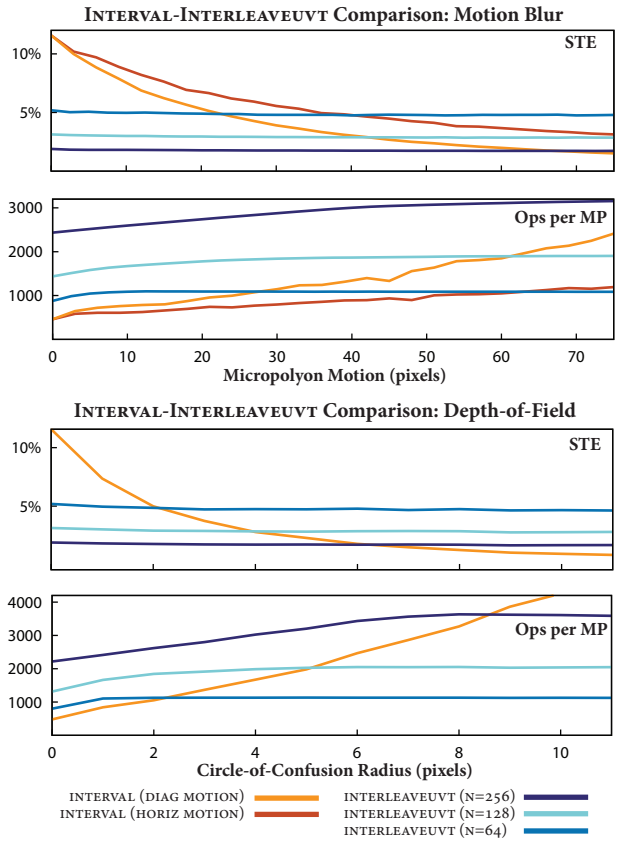

Figure 12: INTERVAL*'s STE (and performance) drops as motion blur (top) and defocus blur (bottom) increase. At high motion, but only small defocus,* INTERVAL*'s performance drops below that of* INTERLEAVEUVT*.* INTERLEAVEUVT*'s parameter N determines where the performance crossover point lies.*

5.2.2 Controlled Study

To gain further insight into the amount of blur required to equate INTERVAL and INTERLEAVEUVT performance, we construct a scene containing randomly positioned and randomly oriented triangle micropolygons of *exactly* 0.5 pixels in area. We measure the STE and per-micropolygon operation-count of both algorithms as the magnitude of micropolygon motion (Figure 12-top) or defocus blur (Figure 12-bottom) is steadily increased.

The results in Figure 12 are consistent with results from our animation scenes. The y-intercept of the STE curves is similar to the 5% and 10% STE measured when rendering animation scenes without scene movement and in sharp focus. In this controlled setup, we find that between 21 and 40 pixels of motion blur are required to equate INTERVAL's STE with that of INTERLEAVEUVT. This corresponds to fast object motion (an object motion crossing 40 pixels in 1/48 of a second will cross a 1728×1080 screen in 0.9 seconds).

INTERLEAVEUVT obtains the same STE as INTERVAL when a micropolygon's defocus blur radius is only two pixels. At high screen resolutions, modest camera defocus yields a blur radius significantly larger than this amount. Under these conditions, the cost of INTERVAL increases greatly. With only ten pixels of defocus blur, INTERLEAVEUVT can utilize 256 unique lens samples and still provide greater performance. This explains the extreme difference in algorithm cost in TALKING.

	Data Parallel Execution Units			
	8	16	32	64
INTERVAL				
Setup	.01 (.12)	.02 (.06)	.02 (.06)	.02 (.06)
Bound	.05 (1.0)	.05 (1.0)	.05 (1.0)	.04 (.99)
Test	.88 (.84)	.85 (.82)	.82 (.79)	.79 (.77)
Process	.05 (.15)	.08 (.09)	.11 (.06)	.15 (.04)
Overall Util	.80	.75	.70	.66
INTERLEAVEUVT N=64				
Setup	.02 (.12)	.03 (.06)	.05 (.03)	.09 (.02)
Bound	.24 (1.0)	.22 (1.0)	.20 (1.0)	.17 (1.0)
Test	.66 (.66)	.63 (.62)	.59 (.59)	.54 (.57)
Process	.08 (.15)	.12 (.08)	.16 (.05)	.20 (.04)
Overall Util	.69	.62	.56	.49
INTERLEAVEUVT N=256				
Setup	.01 (.12)	.01 (.06)	.02 (.03)	.03 (.02)
Bound	.34 (1.0)	.32 (1.0)	.28 (1.0)	.24 (1.0)
Test	.62 (.68)	.62 (.63)	.62 (.55)	.61 (.50)
Process	.03 (.13)	.05 (.07)	.08 (.04)	.12 (.02)
Overall Util	.77	.71	.62	.55

Table 2: *Fraction of total execution time and data-parallel unit utilization (in parentheses) of each stage of* INTERVAL *and* INTERLEAVEUVT *(N=64 and N=256 configurations shown).*

Figure 12's STE and operation-count curves differ in two notable ways. First, the INTERVAL-INTERLEAVEUVT crossover points shift upward when operation-count is considered (between 38 and 60 pixels of motion blur is necessary to equate algorithm cost). This is primarily due to higher utilization of data-parallel execution by INTERVAL. Second, notice that for smaller amounts of blur, INTERLEAVEUVT's STE curves are flat, but its operation-count curves are not. This effect is also due to utilization. When a micropolygon is not blurred, it has the same tile bounds for all UVT tuples; iteration in *Test* exhibits perfect utilization. As the polygon is blurred, variance in tile bounds increases, dropping utilization. Utilization stabilizes once blur becomes large with respect to the INTERLEAVEUVT tile size. This effect is partially responsible for the dips in the INTERLEAVEUVT curves in Figure 11.

5.2.3 Utilization

Table 2 provides detailed execution statistics for each phase of INTERVAL and INTERLEAVEUVT. As done in our NOMOTION analysis, we average statistics over a collection of frames from our animation scenes and provide results for implementations using 8, 16, 32, and 64 execution units.

Both INTERVAL and INTERLEAVEUVT realize lower STE than NOMOTION and correspondingly spend an even higher fraction of time performing sample tests (note that since positioning the micropolygon is hoisted out of the inner loop of INTERLEAVEUVT, much of the time spent in *Bound* can be considered "testing" work). INTERVAL's *Test* phase constitutes over 79% of execution time and maintains over 77% utilization when scaling to 64 execution units. Dividing UVT-space into equal intervals yields similarly sized spatial bounds for each interval, resulting in low variance in the number of iterations through the inner *Test* loop. Recall that scaling INTERVAL to many execution units requires several micropolygons to be processed at once. Micropolygons reaching the rasterizer in succession typically originate from the same region of a tessellated surface and undergo similar motion.

INTERLEAVEUVT achieves lower utilization in these critical regions because there is higher variance in micropolygon-tile overlap than in the size of INTERVAL's bounding boxes. Still, INTERLEAVEUVT remains amenable to wide data-parallel scale out. Although INTERLEAVEUVT does not fully utilize eight execution units (64%), its utilization drops off less than 4% each time the number of execution units is doubled. INTERLEAVEUVT spends a larger fraction of time in *Bound*, which always runs at full utilization. As a result, further optimization to increase utilization of *Test*, such as aggregating work into batches, will yield a performance benefit of at most 37%. We implemented (but do not report on) a more complex version of INTERLEAVEUVT that gains testing efficiency through these optimizations.

This analysis has focused on rasterization of triangle micropolygons to a 16x multi-sampled frame buffer. We have conducted similar studies of INTERVAL and INTERLEAVEUVT using variations in micropolygon area and sampling rate, and using quadrilaterals instead of triangles. While changing the INTERLEAVEUVT tile size or number of intervals used by INTERVAL has notable impacts on STE, we observe the data-parallel execution characteristics of each of these variations to be very similar to the results presented here.

6 Discussion

We have studied micropolygon rasterization with a focus on high-throughput, data-parallel implementation. We have shown that when rasterizing non-blurred micropolygons, many micropolygons must be processed simultaneously to efficiently utilize data-parallel processing (NOMOTION). We have provided an implementation of Pixar's INTERVAL algorithm that scales to wide data-parallel processing. Last, we have applied interleaved sampling to provide a data-parallel algorithm, INTERLEAVEUVT, whose performance, relative to our implementation of Pixar's algorithm, varies from less than half (in cases of minimal blur: small motion and no defocus) to greater than seven times (in cases of significant blur: large motion or almost any camera defocus). Given these properties, a micropolygon rendering system would benefit substantially from the ability to dynamically switch between these two approaches.

Although micropolygon rasterization makes good use of data-parallel processing resources, it still entails extremely high cost. NOMOTION will require hundreds of giga-operations per second to rasterize complex scenes at real-time rates. In terms of sample test efficiency, both INTERVAL and INTERLEAVEUVT are inefficient. Most of the sample tests performed by these algorithms do not result in fragments. Although the computational horsepower available to software is growing rapidly, real-time systems that adopt micropolygons as a fundamental rendering primitive should strongly consider fixed-function processing to accelerate rasterization work.

We have not studied the extent to which micropolygons place new demands on frame-buffer processing. Micropolygon rasterization generates fragments at fine granularity (it does not yield pixel quads or large stamps of hits) and, when motion and defocus blur are enabled, these fragments might be distributed widely across the frame buffer. An important area of future work investigates the efficiency of frame-buffer operations in a micropolygon system.

The ultimate goal of our research is the design of an efficient micropolygon rendering pipeline for real time rendering. We expect both large polygons and micropolygons to be common in future scenes, thus our approach seeks to evolve (not replace) the existing graphics pipeline abstraction and its corresponding implementations. Rasterization algorithms constitute only one aspect of this evolution. Ongoing work considers pipeline interface modifications for motion blur and defocus and addresses the challenges of adaptive tessellation and micropolygon shading.

Last, we observe that micropolygons, motion blur, and camera de-

focus establish a compelling context within which the trade-offs of computing visibility using rasterization or ray tracing should be re-examined. Optimizations to support motion and defocus blurred micropolygons require considerable re-design of current rasterization systems. We are interested to understand the extent to which a high-throughput ray tracer must evolve under these conditions.

Acknowledgments

Support for this research was provided by the Intel Foundation Ph.D. Fellowship Program, an Intel Larrabee Research Grant, and the National Science Foundation Graduate Research Fellowship Program. We would also like to thank Michael Doggett and Tim Purcell for valuable conversations.

References

AKENINE-MÖLLER, T., MUNKBERG, J., AND HASSELGREN, J. 2007. Stochastic rasterization using time-continuous triangles. In *Graphics Hardware 2007*, 7–16.

COOK, R. L., PORTER, T., AND CARPENTER, L. 1984. Distributed ray tracing. *Computer Graphics (Proceedings of SIGGRAPH '84) 18*, 3, 137–145.

COOK, R. L., CARPENTER, L., AND CATMULL, E. 1987. The reyes image rendering architecture. *Computer Graphics (Proceedings of SIGGRAPH '87) 21*, 4, 95–102.

COOK, R. L., PORTER, T. K., AND CARPENTER, L. C., 1990. Pseudo-random point sampling techniques in computer graphics. United States Patent 4,897,806, Jan.

COOK, R. L. 1986. Stochastic sampling in computer graphics. *ACM Transactions on Graphics 5*, 1, 51–72.

DEMERS, J. 2004. Depth of field: A survey of techniques. *GPU Gems*, 375–390.

FUCHS, H., GOLDFEATHER, J., HULTQUIST, J. P., SPACH, S., AUSTIN, J., FREDERICK P. BROOKS, J., EYLES, J., AND POULTON, J. 1985. Fast spheres, shadows, textures, transparencies, and image enhancements in pixel-planes. In *Computer Graphics (Proceedings of SIGGRAPH '85)*, 111–120.

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 processor-enhanced memories. *Computer Graphics (Proceedings of SIGGRAPH '89) 23*, 3, 79–88.

GREENE, N. 1996. Hierarchical polygon tiling with coverage masks. In *Computer Graphics (Proceedings of SIGGRAPH '96)*, 65–74.

HAEBERLI, P., AND AKELEY, K. 1990. The accumulation buffer: hardware support for high-quality rendering. In *Computer Graphics (Proceedings of SIGGRAPH '90)*, 309–318.

HOUSTON, M., 2008. Anatomy of AMD's terascale graphics engine. SIGGRAPH 2008 Class Notes: Beyond Programmable Shading: Fundamentals. `http://s08.idav.ucdavis.edu/houston-amd-terascale.pdf`.

KELLER, A., AND HEIDRICH, W. 2001. Interleaved sampling. In *Eurographics Workshop on Rendering*, 269–276.

MCCOOL, M. D., WALES, C., AND MOULE, K. 2001. Incremental and hierarchical hilbert order edge equation polygon rasterizatione. In *Graphics Hardware 2001*, 65–72.

MCCORMACK, J., AND MCNAMARA, R. 2000. Tiled polygon traversal using half-plane edge functions. In *Graphics Hardware 2000*, 15–21.

MITCHELL, D. 1991. Spectrally optimal sampling for distribution ray tracing. *Computer Graphics (Proceedings of SIGGRAPH '91) 25*, 4, 157–164.

PINEDA, J. 1988. A parallel algorithm for polygon rasterization. *Computer Graphics (Proceedings of SIGGRAPH '88) 22*, 4, 17–20.

SEILER, L., CARMEAN, D., SPRANGLE, E., FORSYTH, T., ABRASH, M., DUBEY, P., JUNKINS, S., LAKE, A., SUGERMAN, J., CAVIN, R., ESPASA, R., GROCHOWSKI, E., JUAN, T., AND HANRAHAN, P. 2008. Larrabee: a many-core x86 architecture for visual computing. In *ACM Transactions on Graphics (SIGGRAPH 2008)*, 1–15.

SUNG, K., PEARCE, A., AND WANG, C. 2002. Spatial-temporal antialiasing. *IEEE Transactions on Visualization and Computer Graphics 8*, 2, 144–153.