High Performance Graphics (2010)
M. Doggett, S. Laine, and W. Hunt (Editors)

Hardware Implementation of Micropolygon Rasterization with Motion and Defocus Blur

J. S. Brunhaver, K. Fatahalian, and P. Hanrahan

Stanford University

Abstract

Current GPUs rasterize micropolygons (polygons approximately one pixel in size) inefficiently. Additionally, they do not natively support triangle rasterization with jittered sampling, defocus, or motion blur. We perform a microarchitectural study of fixed-function micropolygon rasterization using custom circuits. We present three rasterization designs: the first optimized for triangle micropolygons that are not blurred, a second for stochastic rasterization of micropolygons with motion and defocus blur, and third that is a hybrid combination of the two. Our designs achieve high area and power efficiency by using low-precision operations and rasterizing pairs of adjacent triangles in parallel. We demonstrate optimized designs synthesized in a 45 nm process showing that a micropolygon rasterization unit with a throughput of 3 billion micropolygons per second would consume 2.9 W and occupy 4.1 mm^2 which is 0.77% of the die area of a GeForce GTX 480 GPU.

Categories and Subject Descriptors (according to ACM CCS): I.3.1 [Computer Graphics]: Graphics processors—I.3.7 Three-Dimensional Graphics and Realism

1. Introduction

Rapid growth in the performance and programmability of GPUs has motivated researchers to take on the challenge of designing a graphics system capable of rendering detailed scenes with complex, high-resolution surfaces in real-time. These detailed surfaces are commonly represented by micropolygon meshes (polygons of less than a pixel in size).

There has been recent success in efficiently tessellating surfaces into micropolygons [PO08, EML09, FFB*09] as well as in shading micropolygons in a GPU pipeline [FBH*10]. Although geometry processing and shading typically dominate rendering cost, Fatahalian et al. note that micropolygon rasterization is computationally expensive because it cannot leverage optimizations that are common when rasterizing large triangles [FLB*09]. Analysis of micropolygon rendering pipelines implemented as software "compute-mode" applications running on GPUs indicates that rasterization can consume a significant fraction of total rendering time [ZHR*09] and requires a large fraction of a GPU's compute resources to process simple scenes [EL10] at real-time rates. The cost of micropolygon rasterization increases further when motion and defocus blur effects are desired.

Most modern GPU architectures use highly-optimized fixed-function hardware to perform triangle rasterization and, in contrast to previous micropolygon rasterization work, this paper pursues the design of a fixed-function rasterizer optimized for micropolygon workloads. We describe the design space of micropolygon rasterization, and present three optimized hardware designs. The first design implements stochastic rasterization of stationary micropolygons. The second implementation extends this design to support rasterization with motion and defocus blur. The third hardware unit is a hybrid of the two that is optimized to support a combination of blurred and non-blurred inputs.

2. Background

To provide context for the analysis presented in this paper, we will first provide some background on traditional rasterization and micropolygon rasterization.

2.1. Current GPU Rasterization

Traditional rasterization derives efficiency through the exploitation of the spatial coherence of screen samples relative to a scene's geometry. For example, early work utilized the coherence of pixels along a span in the Digital Differential Analyzer (DDA) implementation used in the RealityEngine [Ake93] to achieve high efficiency. Modern rasterization is usually performed in three steps.

First, polygon pre-processing is performed. This preprocessing is often referred to as "setup" and could include the generation of the edge equations required for point in polygon testing [Pin88, FGH*85].

Second, a set of screen space samples which are possibly covered by the polygon are determined. The goal here is to achieve a reasonable sample test efficiency (STE), the number of samples in polygon out of total samples tested, and to maximize the parallelism provided to the sample test portion. Modern systems compute the sample candidates using coarse screen tiles [FPE*89, MM00] and hierarchical techniques [Gre96, MWM01, SCS*09].

Third, each of these sample candidates is evaluated using individual point in polygon tests. Modern systems seek to maximize parallelism here to achieve performance. Traditionally this is done utilizing stamps [FGH*85], where the samples are ordered into blocks ranging from 4x4 to 128x128 samples [Pin88, FPE*89, SCS*09].

This approach to rasterization achieves high performance because large swaths of pixels are trivially in or trivially out of the polygon. In micropolygon rasterization, it does not make sense to test a large set of samples against a polygon as large sets of samples are unlikely to lie completely within a single polygon.

2.2. Micropolygon Rasterization

Fatahalian et al. provides a detailed description of the algorithms that we have implemented in hardware [FLB*09].

In this section we summarize the algorithms for rasterization without blur (NOBLUR) and with motion and defocus blur (BLUR) for clarity.

2.2.1. NOBLUR

The NOBLUR algorithm is summarized in the following pseudo code:

```
Cull back facing
BBOX = Compute MP bbox
foreach sample in BBOX
   test MP against sample
```

NOBLUR determines a screen space bounding box for the micropolygon and clamps the bounding box to the sample grid. It then iterates over those samples testing whether they fall inside the micropolygon using edge equations. This is similar to the algorithms presented in [Pin88].

2.2.2. BLUR

The BLUR algorithm supports motion blur and depth of field effects and is summarized in the following pseudo code:

```
foreach unique UVT tuple //N iterations
   MP_SHIFT = MP position at ui,vi,ti
   BBOX = compute MP_SHIFT bbox
   foreach TILE in BBOX
      SAMP = sample for ui,vi,ti and TILE
      test MP against SAMP
```

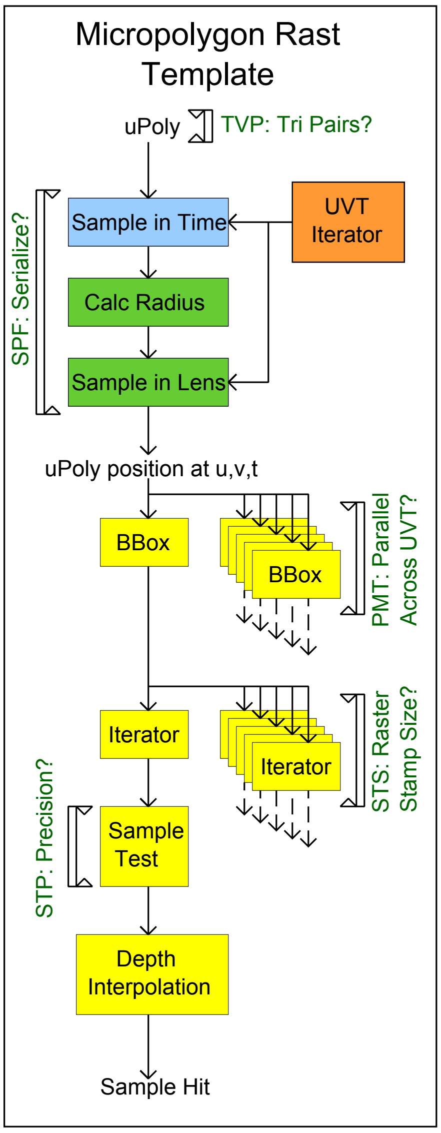
This algorithm estimates the rendering integral for a given frame by stochastically sampling over the lens parameters u and v and through time t. The algorithm iterates over N predefined u, v, t triplets. For each of these unique tuples the micropolygon is positioned appropriately. The micropolygon is first positioned at the sample time. After a micropolygon is positioned in time, the radius of the defocus blur kernel is calculated using the depth of each micropolygon vertex. The position of each vertex is then shifted in x by $u \cdot r$ and y by $v \cdot r$. This implementation simultaneously supports both motion blur and defocus and does not use time-dependent edge equations as proposed by [AMMH07].

N is actually larger than the number of samples per pixel, therefore the sample density in x and y per micropolygon positioned at u, v, t is reduced appropriately. For example, a rendering configuration where N=64 and MSAA x16 corresponds to a sample density of 1 sample for every 4 pixels for each positioned micropolygon. The sets of four pixels are referred to as tiles. The screen space bounding box for the positioned micropolygon is clamped to this grid of tiles. As before, each bounding box is iterated over and the point in triangle test per tile is performed. This sample test is jittered at each tile corresponding to u, v, t in addition to the tile's x and y position.

3. ASIC Rasterizer Microarchitecture

ASIC implementations of BLUR and NOBLUR are instances of the micropolygon rasterization unit template shown in Figure 1. Each instance is defined by the five configuration parameters shown at the bottom of the figure. Our third rasterization pipeline (HYBRID) is produced from another configuration of these parameters and efficiently supports rasterization of both motion blurred and non-blurred micropolygons. The calculations performed in each functional block are fixed-point (we assume that vertex positons have been converted to fixed point outside of this rasterization pipeline). This section describes the components of this template and their associated design parameters.

- The light blue, "sample in time" function block corresponds to the position in time operation described in the previous section. This $x_i(1-t) + x_{i+1}t$ operation is performed for each vertex.
- The green "calc radius" function block is equivalent to an inversion operation for each vertex's z value $(r = \frac{1}{z})$.



Microarchitectural Design Parameters	
TVP:	Rasterize triangle pairs
SPF:	Serialize positioning in u,v,t
PMT:	Parallelism across u,v,t tuples
STS:	Sample test stamp size
STP:	Sample test precision

Figure 1: *Microarchitecture: This block diagram presents the design for a micropolygon rasterization unit with support for jittered sampling, defocus, and motion blur. By removing features from the design it is possible to generate* NOBLUR *function units or units that support only defocus or motion blur. The parameters for the design space are indicated in green and represent optimization opportunities.*

- The green "sample in lens" function block corresponds to positioning the micropolygon with respect to the u,v values over the lens. The operation is equivalent to $u \cdot r$ in x and $v \cdot r$ in y.
- The yellow "bbox" function block determines the clamped bounding box for a positioned micropolygon or NOBLUR micropolygon. This is implemented as a set of parallel comparisons used to select minima and maxima. The minima and maxima are rounded in order to clamp to the x,y sample grid.
- The yellow "iterator" function block is a simple increment and compare implementation that also generates stall signals for the previous stages of the pipeline. It is equivalent to the for-each sample operator in the pseudo code.
- The yellow "sample test" function block calculates the edge distances required for the barycentric interpolation. Additionally, the sample test block determines whether the sample location falls within the micropolygon.
- The yellow "depth interpolation" function block is used to calculate the interpolated depth of the individual hit sample using the edge distances provided by the sample test block. The block is implemented as an inversion to calculate a normalization factor $n = 1/(d_1 + d_2 + d_3)$ and the multiply add using the norm factor $z_{interp} = \sum_0^2 d_i z_i n$.

Each of these units is required to implement a BLUR or HYBRID unit. The NOBLUR design exists as a subset of the BLUR design. NOBLUR requires only the yellow components: bbox, iterator, sample test, and depth interpolation. The NOBLUR design is impacted by the TVP, STS, and STP microarchitecture parameters.

- The first design parameter TVP determines whether to operate on individual triangles or to operate on two triangles simultaneously that share an edge. In order to operate on pairs of triangles, the unit tessellating primitives into triangles must communicate the micropolygon connectivity. With pairs of triangles, the operations which position vertices with respect to the u,v,t tuple must now accommodate four vertices as opposed to three. Additionally, the bounding box unit must now determine the bound for four points in space rather than three points. Finally the sample test unit will now perform five edge tests for two point in polygon tests per triangle pair rather than three edge tests for one point in polygon test per individual triangle.
- The second design parameter *SPF* is related to the u,v,t sample operators specific to the blur implementations. This parameter defines the parallelism in calculating the vertex positions. By allocating more per-vertex operation units per-cycle a higher micropolygon throughput could be achieved. The numerical evaluation of the parameter is with respect to serialization as it describes both the reduction in operation units and micropolygon positioned at u,v,t throughput.
- The third parameter *PMT* determines the number of u,v,t tuples to evaluate in parallel. This is specific to BLUR and

HYBRID designs and requires a new instantiation from bbox down to depth interpolation per parallel evaluation.

- The fourth parameter STS determines the number of sample tests performed in parallel per u, v, t positioned micropolygon. This parameter determines the size of the sample raster stamp. A 1x1 sample stamp is equivalent to testing one sample per cycle and imitates the pseudo code given for NOBLUR in section 2.2.1. A 2x2 sample stamp is equivalent to testing four samples per cycle. A 2x2 sample stamp iterates over a bounding box with a stride of two pixels. Additionally, the bounding box is now clamped to a 2x2 sample grid rather than a 1x1 sample grid.
- The fifth parameter STP determines the number of leading bits required for the sample test unit. This optimization attempts to take advantage of the fact that a tessellation unit upstream of the rasterization unit [FFB*09] will generate micropolygons whose width and height are less than 8 pixels widths with high probability $(> 1 - 1/10^6)$. For a sample test 8 bits of sub pixel precision are required [Bly06]. Additionally, we must be able to represent scene geometry in a 16k x 16k pixel render target [Bly06]. To meet these requirements 22 bits or 14.8 is required. If we assume a micropolygon pipeline, only 4.8 bits of precision are required to represent vertex position accurately. This optimization only impacts the precision required for the sample test unit, all other units use the required 14.8 precision.

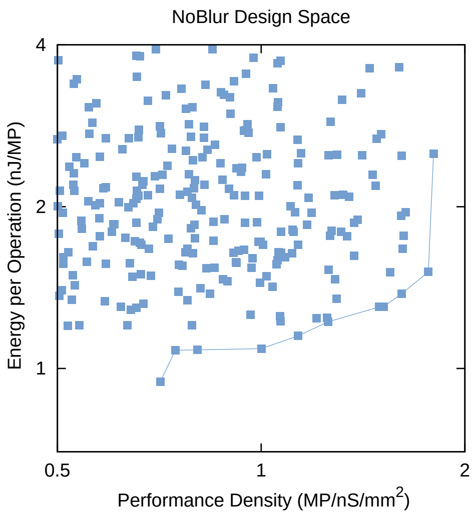


Figure 2: NOBLUR *Design Space: Each point represents an individual* NOBLUR *rasterization design at 16x MSAA. Along the frontier of this design space it is no longer possible to extract energy savings while simultaneously increasing performance. Instead, along this frontier energy efficiency can be traded for increased performance density as a form of design currency.*

4. Methodology

We seek to identify the microarchitecture parameters which result in energy and area efficient designs. To do this, we explore the physical design space for each of the microarchitecture implementations.

We vary the physical design parameters across low V_t, nominal V_t, and high V_t cell libraries at four domain voltages V_{dd} clustered around .9 Volts. V_t and V_{dd} are used as a knob to increase the performance per area, as individual gates of constant size become faster for lower V_t and high V_{dd}. We vary the clock speed in order to match the speed of the critical path at these V_t and V_{dd} pairs. Finally, pipe stages are added in order to reduce the path length. In order to accommodate variable delay, clock speed, and pipe stage count, we made use of Design Compilers data-path pipeline retiming tool. Additionally, the microarchitecture parameters given in Figure 1 impacted the physical design and were also considered where appropriate.

To determine the cycle time, power, and area for individual data path elements, each module was synthesized in a 45nm TSMC process with a pessimistic leakage model. This analysis was performed using the Synopsys Design Compiler tool at the granularity of the function blocks given in Figure 1. The data path of each function block was created using System Verilog and the Design Compiler DesignWare Library. The control paths for these elements are small compared to the data-path and is modeled by the cost of the control path elements. We incorporate the distribution of the control signals, which dominate the overall cost of this type of control circuit.

We performed synthesis using aggressive compilation routines and data path retiming tools. However, we did not perform any path fixing, manual retiming, or path duplication for any of the designs. Dynamic power was based on the characterization of the TSMC 45nm cell libraries coupled with a conservative estimate of switching factors for individual gate nodes and the margin required for wire routing. The estimates for dynamic power did not incorporate any power saving features like fine grained clock gating. We assumed that each function block would be fully used at any given time. Leakage power was determined using conservative cell characterizations. The area of the design was based on total cell area coupled with a place and route area estimation tool.

Each point shown in Figure 2 represents a unique instance of the template rasterizer from Figure 1. Each of these instances were created by varying the microarchitectural parameters and physical design parameters for each instance. The lower right hand portion of the plot represents desirable high performance and low energy designs, while the upper left represents undesirable low performance and high energy

Design Parameter	NoBlur	Blur	Hybrid
TVP:Triangle Pairs	Yes	Yes	Yes
SPF:Serial u,v,t Sample	n/a	No	Yes
PMT:Parallel over UVT	n/a	3	1
STS:Sample Stamp Size	2x2	1x1	2x2
STP:Sample Test Precision	4.8	4.8	4.8

Table 1: *Summary of Optimal Implementations: We indicate in this table the value of the parameters used to optimize each of the three designs:* NoBlur, Blur, *and* Hybrid.

designs. The goal here is to identify the efficient design frontier or the Pareto optimal designs as shown with the curve in Figure 2. Among the Pareto optimal designs there exist no other known designs which have a lower energy per operation and higher performance density [HQW*10, AMPH09]. Instead the two values are used as a form of currency to trade energy efficiency for greater performance density. The designs in the upper right hand corner of the plot are designs that have traded energy efficiency to gain performance density. The designs in the lower right hand corner of the plot correspond to designs that have traded performance density to gain energy efficiency.

Additional performance can be gained by replicating the rasterization function unit from Figure 1. The goal then is not to optimize individual unit power, unit area, and unit performance. Instead the goal is to maximize the triangle rasterization rate in a given die area while minimizing the energy per rasterized triangle.

5. Results

For our evaluation we used scenes composed of triangles with an average area of 0.54 pixels and rendered with 16x MSAA. For scenes with blur we used N=64 u,v,t tuples with a 2x2 pixel tile size. Note that the performance of Blur is invariant to the amount of motion in the scene.

Table 1 summarizes the optimal design decisions in this space with respect to the microarchitecture parameters.

5.1. NoBlur

The NoBlur design served as a good starting point as it would be integrated into all other designs. The implementation of NoBlur led us to two optimizations which were useful across all designs and one optimization that was useful for improving NoBlur power and area efficiency.

First, Figure 3 shows the Pareto curves for two constrained designs. The orange curves of circles represent designs which were constrained to operate only on pairs of triangles, while the blue curve of squares represents designs which were constrained to operate only on triangles. The triangle pairs curve never intersects the individual triangles

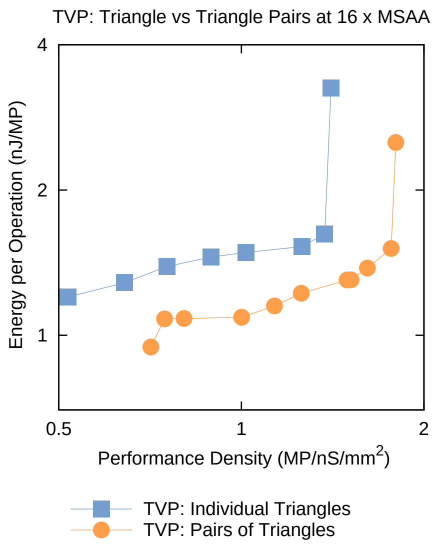


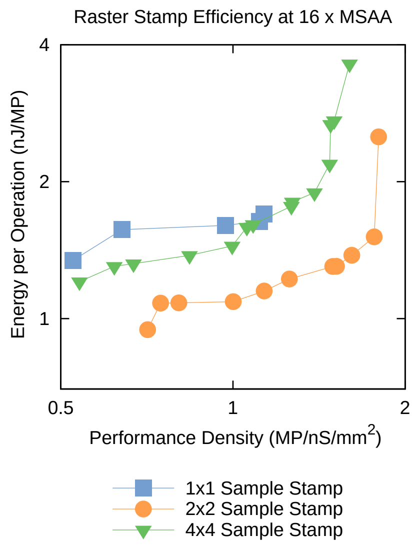
Figure 3: *Triangles vs. Pairs of Triangles: This figure shows the efficiency of two sets of designs. In one set the design was optimized under the constraint that it must use individual triangles as the base primitive (blue squares), while the other set was constrained to operate on pairs of triangles simultaneously (orange circles).*

curve, therefore it is always better to rasterize pairs of triangles. At the 1 W/mm^2 power density an implementation operating on triangles consumes 15% more resources than an implementation operating on pairs of triangles.

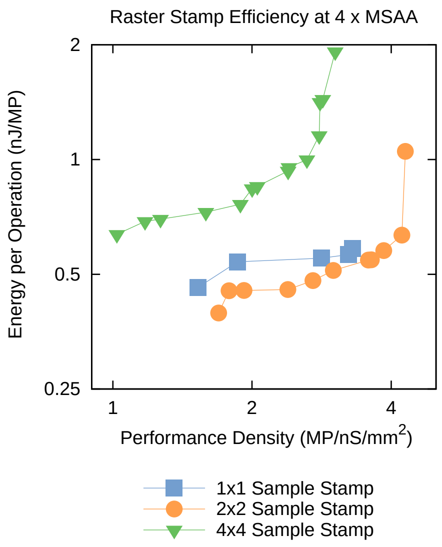
This improvement from operating on pairs of triangles simultaneously rather than individual triangles comes from two sources. First, a triangle pairs exhibit a higher sample test efficiency (more useful work is done per sample test) than individual triangles. Second, operating on two triangles simultaneously allows us to amortize the cost of: calculating the two shared vertices' positions in u,v,t and performing an edge test for the shared edge.

Second, we looked for opportunities to decrease the bit width of the computations in order to increase the efficiency of the design. This corresponds to the microarchitectural parameter STP. Figure 4 shows the resources required to accommodate a 14.8 or 22 bit width sample test in a NoBlur rasterizer is double the resources required to support 4.8 or 12 bit width sample testing.

We also found, that by decreasing the cost of the sample test, a function block 2x2 sample raster stamp became efficient due to the relative cost of an iterator, sample test, and interpolation unit relative to the cost of the bounding box operation. The designs in Figure 4 corresponding to 4.8 and 6.8 bit width sampling operation use a 2x2 sample raster stamp while the remainder use a single sample per cycle pipeline.



(a) MSAA x16



(b) MSAA x4

Figure 5: NoBlur *Raster Stamp Efficiency: The figure on the left shows the efficiency of three sets of designs at 16x MSAA. Each set has a different raster sample stamp dimension, corresponding to 1 sample per cycle, 4 sample per cycle, and 16 samples per cycle. The data points contained in each set represent the optimal designs given a constraint on sample raster stamp size. The figure on the right repeats this design space exploration for 4x MSAA.*

The shift to a smaller bit width sample unit for NoBlur rasterization allowed for an increase in samples evaluated per cycle with a 2x2 sample raster stamp. In Figure 5(a) the NoBlur 16x MSAA rasterization units which were constrained to use 2x2 sample raster stamps are represented in orange circles. These designs were consistently more efficient than the designs constrained to operate on 4x4 stamps, in green triangles, and 1x1 stamps, in blue squares. In Figure 5(b), the NoBlur 4x MSAA rasterization units which were constrained to use 2x2 sample raster stamps, in green circles, are consistently more efficient then either the 4x4 stamps, in blue, or 1x1 stamps, in red. The difference between 4x MSAA and 16x MSAA is that for 4x MSAA the difference in efficiency for 1x1 stamps and 2x2 stamps is small and at the most a 10% increase in cost. However for 16x MSAA the difference in cost between 2x2 stamps and 1x1 stamps is about 40%.

The cost savings, when shifting to 2x2 stamps from a 1x1 stamp, are derived from amortizing the cost of three bounding box units while only requiring a 25% increase in the number of samples to be evaluated. For example, for the design given in Table 2 25% of the cost of an iterator, sample test, and interpolater is less than three times the cost of a bounding box unit.

The curves in Figure 3 and Figure 5 don't trade positions along the efficient frontier. For these microarchitectural parameters, only one parameter choice is Pareto optimal.

Component	Power (mW)	Area (mm^2)
Sample in Time	52	.10
Sample in Lens	28	.043
Bbox	14	.012
Back-face Cull	10	.0010
Iterator	3	.0020
Sample Test	18	.0015
Interpolation	14	.013

Table 2: *Cost Breakdown: The power and area required for each of the base components of a 1 $\frac{W}{mm^2}$ optimal* Blur *design. This design had an operating voltage of .99V, used High V_t devices, and ran at a clock speed of 2GHz.*

Note that a 2x2 sample stamp differs from the algorithm used by [FLB*09] and shown in 2.2.1. For an ASIC, functions are statically implemented and the goal is to optimize the utilization of a function block relative to its total expense, not necessarily maximizing STE. In this case the reduction in utilization for each sample test lane in relation to its cost was less than the increase in utilization of the BBox and back face cull relative to their cost.

NVIDIA's GeForce GTX 480 GPU, which is fabricated in a similar 45nm process, has a die size of 529 mm^2 [NVI]. This GPU can process four triangles per clock at 700 MHz resulting in a throughput of 2.8 billion triangles per second.

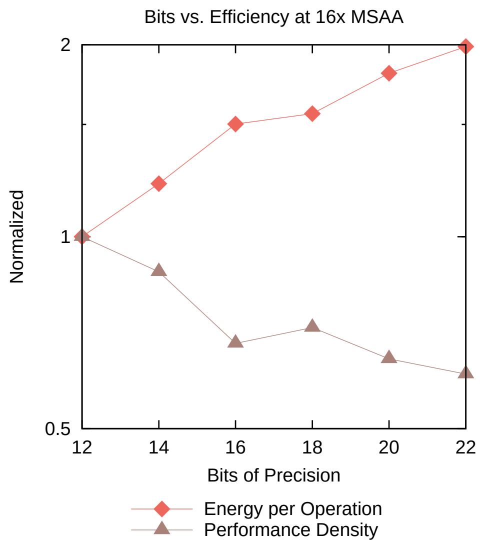


Figure 4: *Savings from Reduced Precision: This graph presents the increase in cost for a* NoBlur *design at 16x MSAA as the number of leading bits used in the sample test unit is increased from 4.8 to 14.8. The results are normalized to the performance density and energy per operation of a design using 4.8 or 12 bits of precision. The red diamond data points indicate the increase in energy per operation and the purple triangle data points indicate the decrease in performance density.*

Power Density →	.7 W/mm^2	1.0 W/mm^2	1.5 W/mm^2
NoBlur (W)	2.9 (1)	3.3 (1)	3.7 (1)
Motion blur (W)	20 (7)	24 (7)	27 (7)
Defocus blur (W)	19 (7)	23 (7)	29 (8)
Blur (W)	25 (9)	30 (9)	38 (11)
Hybrid-NoBlur (W)	3.9 (1)	5.0 (2)	6.4 (2)
Hybrid-Blur (W)	43 (15)	56 (17)	72 (19)

Table 3: *The Cost of* Blur *Relative to The Cost of* NoBlur*: This plot shows the costs for 6 different designs for three different power densities. The power density of a design is simply the multiplication of its performance density with its energy per operation. The area of the designs can be found by dividing the power by the power density. The values in the parenthesis indicates the multiplied cost for that design over* NoBlur *at the same power density. Note that* Hybrid*-NoBlur and* Hybrid*-Blur are the same design operating in different render modes (*NoBlur *and* Blur*).*

For a three billion micropolygon per second rasterization rate the NoBlur rasterization unit would consume 2.9 W and occupy 4.1 mm^2 which is 0.77% of the die area for a GeForce GTX 480 GPU.

Sustaining this rasterization rate with the data-parallel software NoBlur implementation described in [FLB*09] would require 160 billion 32-wide vector operations per second. No GPU today provides this level of software performance. Although fixed-function rasterization hardware may not provide direct speedup in systems bottlenecked by shading costs, it provides an opportunity to allocate more general-compute resources to the problem of shading.

5.2. Blur

We found that for a Blur design which supports blur, the ideal sample parallelism is to compute the position of three micropolygons position at u_i, v_i, t_i micropolygons simultaneously. In Figure 6 the design constrained to evaluate three positioned micropolygons in parallel (PMT=3), represented as orange circles, was consistently more efficient than any of the other designs. It is difficult to maintain full occupancy for all three of the evaluation units because the u, v, t sample units only emit one u, v, t positioned micropolygon per cycle while the average micropolygon only covers three tiles. Regardless of the reduced utilization, the increase in throughput is enough to amortize the significant cost in calculating vertex positions for u, v, t. Referring to Table 2, the evaluation of a positioned micropolygon will require 60 mW and .14 mm^2 while the u, v, t sample unit will require 80 mW and .030 mm^2. At this specific design point, the parallelism was desirable even at a lower utilization.

Unlike the NoBlur design the optimal Blur uses a stamp size that is 1x1 samples. It is also important to note that testing multiple samples per cycle using a sample stamp larger than 1x1 is inefficient because x and y samples are sparse and bounding boxes are small.

In Figure 7 the distance between the blue squares representing the set of optimal NoBlur designs and the red diamonds representing the set of optimal Blur designs is about 10x. Table 3, which is a summary of Figure 7, indicates the power and area required to rasterize 3 billion micropolygons per second at three power densities. Power density is simply the multiplication of the energy per micropolygon with the performance per millimeter. The area is derived from dividing the required micropolygon rate by the performance density of the design. The power is derived by multiplying the required micropolygon rate by the energy per operation. For a three billion micropolygon per second rasterization rate the Blur rasterization unit would consume 25 W and occupy 36 mm^2 which is 6.8% of the die area for a GeForce GTX 480 GPU.

5.3. Hybrid

To support both NoBlur and Blur render modes the first step would simply be to add a bypass around the sample u, v, t units to evaluate the unblurred micropolygons di-

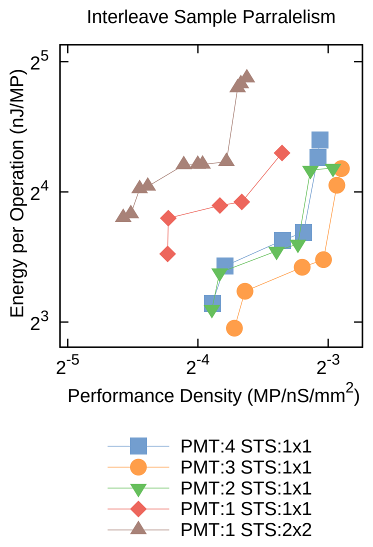


Figure 6: *Efficiency in Sample Parallelism for* BLUR*: This figure presents the efficiency of sample parallelism for* BLUR*. Each curve represents the Pareto optimal fixed-function units resulting from a design space constrained to use some form of sample parallelism. There are two forms of sample parallelism shown here. First, the purple triangles represent designs that sample positioned micropolygons using a 2x2 sample raster stamp and is noted as STS: 2x2. Second, the blue squares, orange circles, and green triangles represent the set of designs constrained to operate on 4,3, and 2 positioned micropolygons in parallel. Finally, the set of designs which were constrained to not use any sample parallelism is represented with the red diamonds.*

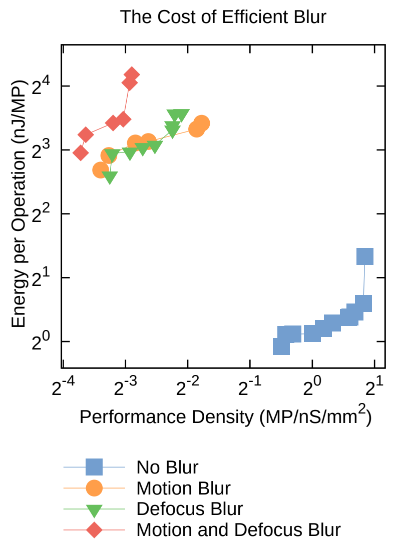


Figure 7: *The Cost of Efficient* BLUR*: This figure shows the efficiency for designs which optimize for a single render configuration. Each curve represents the Pareto optimal fixed-function units resulting from a design space constrained to be efficient for its specific render configuration.*

rectly. We found that at a power density of $1W/mm^2$ using a BLUR unit would require 4x to 5x more die resources than a NOBLUR unit to perform NOBLUR micropolygon rasterization at 16x MSAA. This increase in cost is due to two factors. First the BLUR unit did not make use of a 2x2 sample raster stamp. Second there is an overhead to the BLUR u,v,t sample functions that are unused.

Instead of optimizing based on the performance of one render configuration it is useful instead to optimize based on both BLUR and NOBLUR render configurations. The design of this HYBRID unit is different than the prior narrow focus designs.

Our goal was the design of a unit that equitably supports NOBLUR at 4x MSAA, NOBLUR at 16x MSAA, motion blur at N=64, and BLUR at N=64. Constraining our design to be optimal over all of these configurations, we found that it was efficient to do three things. First a HYBRID design should use a 2x2 raster sample stamp. Second it should reduce the parallelism of the sample in time and sample in lens operations such that it produces a single micropolygon positioned at u,v,t every other cycle. Third a HYBRID design should avoid parallelism across micropolygons positioned at u,v,t. The results are shown in Table 3 where the cost of NOBLUR and BLUR rasterization using a HYBRID unit are only 2x more than the cost of a NOBLUR unit and BLUR unit. A HYBRID special function unit capable of rasterizing three billion non-blurred (270 million motion and defocus blurred) micropolygons per second would consume 3.9 W and occupy 5.6 mm^2 which is 1.0% of the die area for a GeForce GTX 480 GPU.

6. Discussion

Although micropolygon rasterization is computationally expensive, it can be implemented very efficiently using fixed-function custom circuits. Our NOBLUR rasterizer requires only 2.9 W and 4.1 mm^2 to sustain the triangle rate of a GeForce GTX 480 GPU. This unit would consume 0.77% of the die area of this chip. Supporting motion and defocus blur at this triangle rate requires 8.6 times more area and power. Given the substantial costs of performing the equivalent computations in software, we believe that future graphics systems will continue to need hardware support for rasterization.

Our rasterization units are optimized assuming all triangles are bounded in area. In practice it is difficult to guarantee this bound during pipeline tessellation and, in addition, a real-time system will need to support workloads that contain a mixture of triangle sizes. Just as we examined a hybrid rasterizer implementation that supports blurred and non-blurred micropolygons, future work should investigate the design of a rasterization unit that efficiently processes both larger triangles and micropolygons.

Last, hardware acceleration of stochastic point-sampling is a brute force approach to computing motion and defocused blurred visibility. We encourage investigation of alternative algorithms for accurate camera simulation in real-time rendering.

Acknowledgments

Support for this research was provided by the Stanford Pervasive Parallelism Laboratory funded by Oracle, NVIDIA, IBM, NEC, AMD, and Intel. We would like to thank the anonymous reviewers, Solomon Boulos, Edward Luong, Jonathan Ragan-Kelley, Henry Moreton, Bill Mark, Kurt Akeley, Sameh Galal, Andrew Danowitz and Samantha Brunhaver for their enlightening conversations and helpful feedback.

References

[Ake93] AKELEY K.: Reality engine graphics. In *SIGGRAPH '93: Proceedings of the 20th annual conference on Computer graphics and interactive techniques* (1993), ACM, pp. 109–116.

[AMMH07] AKENINE-MÖLLER T., MUNKBERG J., HASSELGREN J.: Stochastic rasterization using time-continuous triangles. In *GH '07: Proceedings of the 22nd ACM SIGGRAPH/EUROGRAPHICS symposium on Graphics hardware* (2007), Eurographics Association, pp. 7–16.

[AMPH09] AZIZI O., MAHESRI A., PATEL S. J., HOROWITZ M.: Area-efficiency in cmp core design: Co-optimization of microarchitecture and physical design. *SIGARCH Comput. Archit. News 37*, 2 (2009), 56–65.

[Bly06] BLYTHE D.: The direct3d 10 system. *ACM Trans. Graph. 25*, 3 (2006), 724–734.

[EL10] EISENACHER C., LOOP C.: Data-parallel micropolygon rasterization. *Eurographics Proceedings 29*, 2 (2010).

[EML09] EISENACHER C., MEYER Q., LOOP C.: Real-time view-dependent rendering of parametric surfaces. In *I3D '09: Proceedings of the 2009 symposium on Interactive 3D graphics and games* (2009), ACM, pp. 137–143.

[FBH*10] FATAHALIAN K., BOULOS S., HEGARTY J., AKELEY K., MARK W. R., MORETON H., HANRAHAN P.: Reducing shading on GPUs using quad-fragment merging. In *SIGGRAPH '10: ACM SIGGRAPH 2010* (2010), ACM.

[FFB*09] FISHER M., FATAHALIAN K., BOULOS S., AKELEY K., MARK W. R., HANRAHAN P.: Diagsplit: Parallel, crack-free, adaptive tessellation for micropolygon rendering. In *SIGGRAPH Asia '09: ACM SIGGRAPH Asia 2009 papers* (2009), ACM, pp. 1–10.

[FGH*85] FUCHS H., GOLDFEATHER J., HULTQUIST J. P., SPACH S., AUSTIN J. D., BROOKS JR. F. P., EYLES J. G., POULTON J.: Fast spheres, shadows, textures, transparencies, and image enhancements in pixel-planes. In *SIGGRAPH '85: Proceedings of the 12th annual conference on Computer graphics and interactive techniques* (1985), ACM, pp. 111–120.

[FLB*09] FATAHALIAN K., LUONG E., BOULOS S., AKELEY K., MARK W. R., HANRAHAN P.: Data-parallel rasterization of micropolygons with defocus and motion blur. In *HPG '09: Proceedings of the Conference on High Performance Graphics 2009* (2009), ACM, pp. 59–68.

[FPE*89] FUCHS H., POULTON J., EYLES J., GREER T., GOLDFEATHER J., ELLSWORTH D., MOLNAR S., TURK G., TEBBS B., ISRAEL L.: Pixel-planes 5: a heterogeneous multiprocessor graphics system using processor-enhanced memories. In *SIGGRAPH '89: Proceedings of the 16th annual conference on Computer graphics and interactive techniques* (1989), ACM, pp. 79–88.

[Gre96] GREENE N.: Hierarchical polygon tiling with coverage masks. In *SIGGRAPH '96: Proceedings of the 23rd annual conference on Computer graphics and interactive techniques* (1996), ACM, pp. 65–74.

[HQW*10] HAMEED R., QADEER W., WACHS M., AZIZI O., LEE B. C., RICHARDSON S., KOZYRAKIS C., HOROWITZ M.: Understanding sources of inefficiency in general-purpose chips. In *ISCA '10: Proceedings of the 37th Annual International Symposium on Computer Architecture, to appear* (2010), IEEE Press.

[MM00] MCCORMACK J., MCNAMARA R.: Tiled polygon traversal using half-plane edge functions. In *HWWS '00: Proceedings of the ACM SIGGRAPH/EUROGRAPHICS workshop on Graphics hardware* (2000), ACM, pp. 15–21.

[MWM01] MCCOOL M. D., WALES C., MOULE K.: Incremental and hierarchical Hilbert order edge equation polygon rasterizatione. In *HWWS '01: Proceedings of the ACM SIGGRAPH/EUROGRAPHICS workshop on Graphics hardware* (2001), ACM, pp. 65–72.

[NVI] NVIDIA: NVIDIA GF100 white paper: World's fastest GPU delivering great gaming performance with true geometric realism. v1.4.

[Pin88] PINEDA J.: A parallel algorithm for polygon rasterization. *SIGGRAPH '88: Proceedings of the 15th annual conference on Computer graphics and interactive techniques 22*, 4 (1988), 17–20.

[PO08] PATNEY A., OWENS J. D.: Real-time Reyes-style adaptive surface subdivision. In *SIGGRAPH Asia '08: ACM SIGGRAPH Asia 2008 papers* (2008), ACM, pp. 1–8.

[SCS*09] SEILER L., CARMEAN D., SPRANGLE E., FORSYTH T., DUBEY P., JUNKINS S., LAKE A., CAVIN R., ESPASA R., GROCHOWSKI E., JUAN T., ABRASH M., SUGERMAN J., HANRAHAN P.: Larrabee: A many-core x86 architecture for visual computing. *Micro, IEEE 29*, 1 (Jan.-Feb. 2009), 10–21.

[ZHR*09] ZHOU K., HOU Q., REN Z., GONG M., SUN X., GUO B.: Renderants: Interactive Reyes rendering on GPUs. In *ACM Transactions on Graphics (SIGGRAPH Asia 2009)* (2009), ACM SIGGRAPH.