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

A Lazy Object-Space Shading Architecture With Decoupled Sampling

Christopher A. Burns[1] Kayvon Fatahalian[2] William R. Mark[1]

[1]Intel Labs
[2]Stanford University

Abstract

We modify the Reyes object-space shading approach to address two inefficiencies that result from performing shading calculations at micropolygon grid vertices prior to rasterization. Our system samples shading of surface sub-patches uniformly in the object's parametric domain, but the location of shading samples need not correspond with the location of mesh vertices. Thus we perform object-space shading that efficiently supports motion and defocus blur, but do not require micropolygons to achieve a shading rate of one sample per pixel. Second, our system resolves surface visibility prior to shading, then lazily shades 2x2 sample blocks that are known to contribute to the resulting fragments. We find that in comparison to a Reyes micropolygon rendering pipeline, decoupling geometric sampling rate from shading rate permits the use of meshes containing an order of magnitude fewer vertices with minimal loss of image quality in our test scenes. Shading on-demand after rasterization reduces shader invocations by over two times in comparison to pre-visibility object-space shading.

Categories and Subject Descriptors (according to ACM CCS): I.3.3 [Computer Graphics]: Picture/Image Generation—Line and curve generation

1. Introduction

Over 20 years ago, the Reyes rendering system demonstrated an efficient approach for rendering anti-aliased motion blur and depth-of-field effects. Reyes achieves its efficiency by performing shading computations in object space prior to visibility testing, so that each expensive shading computation is reused by many visibility samples, even in the presence of motion blur and defocus blur. However, the Reyes system has two limitations that would be desirable to overcome for interactive use on modern hardware: the decision to shade at triangle vertices requires that surfaces always be tessellated to pixel-size micropolygons, which increases visibility-testing costs; and the system performs occlusion culling at a coarse granularity only, resulting in many unnecessary shading computations. In this paper, we extend the Reyes object-space shading technique to overcome these limitations. Our approach builds on techniques originally developed for GPU pipelines that shade in image space.

In Reyes, input primitives are split into small sections called *grids*, which are diced into micropolygons approximately one or one half pixel in size. The micropolygons are shaded at the vertices in object space prior to visibility. Instead of shading on grids of vertices, our system shades on 2x2 blocks of surface samples that are uniformly distributed in object space within a grid. This change allows our system to improve performance in two ways:

- The system shades approximately once per pixel without requiring that triangles be tessellated to one-pixel (micropolygon) size. Triangle size is set independently from shading rate, with the size based only on the required geometric detail. Using fewer but larger triangles reduces the cost of visibility testing.
- The system shades each 2x2 block of pixels on demand, after performing fine visibility testing. By doing this, we avoid excess shading associated with conservative approximations of visibility. Because the 2x2 block of shading samples contains many fewer shading samples than a typical Reyes grid, we further reduce the number of samples that are shaded but not used. The net result is that our system shades fewer samples.

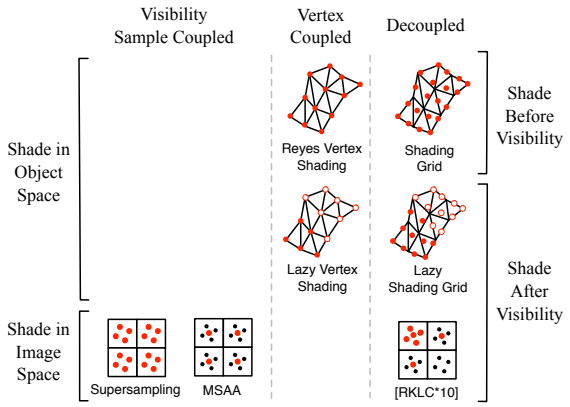

Figure 1: *A shading architecture design space. Red dots indicate shading samples, black dots are visibility samples, and hollow dots are unevaluated samples. Shading can be computed in object space or image space, and may be coupled with vertices or visibility samples, respectively. This paper presents the "Lazy Shading Grid".*

	(1) Fine Occlusion Culling	(2) Shading Reuse Across Pixels	(3) Shading Reuse at Triangle Edges	(4) Supports Non-micro Triangles
GPU + Fragment Merge [FBH*10]	●		●	●
GPU + Memoization Cache [RKLC*10]	●	●		●
Reyes [CCC87]		●	●	
Lazy Shading Grid	●	●	●	●

Figure 2: *Ours is the only shading system with these four performance-related features, which we discuss in Section 2. Shading reuse across pixels is especially important to support motion blur and depth-of-field effects efficiency.*

Decoupling shading from triangle vertices presents benefits to the Reyes pipeline analogous to the advantages that adaptive multisampling has over supersampling in an image-space shading system. Surface tessellation fidelity can now be driven solely by required geometric detail, which is generally coarser than the required density of shading samples. We show that this change can halve the cost of visibility testing using stochastic rasterization, as compared to Reyes micropolygon shading, with very little loss in image quality for our test scenes.

To reduce unnecessary shader invocations, we implement and evaluate a mechanism to defer shading until precise visibility information is known. By shading *lazily*, we avoid shading portions of grids that are occluded, back-facing, or outside the current view frustum. In scenes with many silhouettes, small occluders, and high depth complexity the savings can exceed 2x. For rendering systems that tile the frame buffer and process tiles independently in parallel, the benefits are even greater, as we show in our evaluation.

This paper focuses on algorithmic improvements to object-space shading pipelines, rather than an optimized implementation. A recent line of research has investigated micropolygon rendering using image-space shading [FBH*10]. We build on some key ideas from this previous work, but consider our object-space approach to be an alternative to these image-space approaches; for evolutionary reasons some system designers will have an a-priori preference for either object-space or image-space shading, and we are not attempting to argue that shading in object space is inherently better or worse than shading in image space. We begin with an analysis of the design space and previous work.

2. The Design Space

The design space of shading architectures, illustrated in Figure 1, can be divided into two groups according to the coordinate space in which surface shading is sampled. Image space shading systems compute surface color at image coordinates after geometry has been projected and visibility determined, while object-space techniques compute sample locations in object-space, usually prior to visibility testing. Figure 2 complements this taxonomy with a feature comparison checklist for the most relevant related systems.

2.1. Object-Space Shading

Object-space shading architectures are typified by the Reyes architecture [CCC87], which tessellates surfaces into micropolygons that are shaded prior to visibility. The principle advantage of the Reyes architecture is that shading results for a surface can be reused for many pixels, efficiently supporting complex blur effects important for cinematic rendering such as motion blur and camera defocus (Fig. 2, col. 2).

There are two serious disadvantages to this approach. Since shading is performed at triangle (or quad) vertices, triangle size must be kept small to support the requirement for a high density of shading samples, even when a surface is flat and larger triangles would provide sufficient geometric detail (Fig. 2, col. 4). Small triangles impose a significant burden on the rasterizer [FLB*09]. Second, surfaces are shaded before precise visibility information is available, preventing precise occlusion culling (Fig. 2, col. 1). This results in several forms of extraneous shading relative to a fragment-shading system. To the authors' knowledge, post-visibility object-space shading has not been explored in the literature. Despite these drawbacks, there is increased interest in Reyes as a viable real-time graphics architecture with the addition of dynamic tessellation to the DX11 pipeline [Mic10], and recent efforts to implement the Reyes pipeline on current GPU hardware [ZHR*09] [PO08].

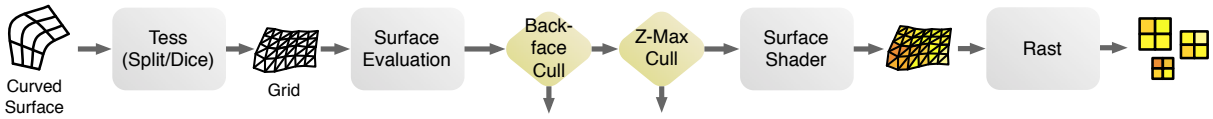

Figure 3: *Data flow diagram of the standard Reyes micropolygon rendering pipeline. The tessellator produces a stream of small triangle meshes called* grids *from an input curved surface representation. The surface attributes are evaluated at the triangle vertices and optionally displaced. After culling, the triangle vertices are lit and shaded, and finally rasterized.*

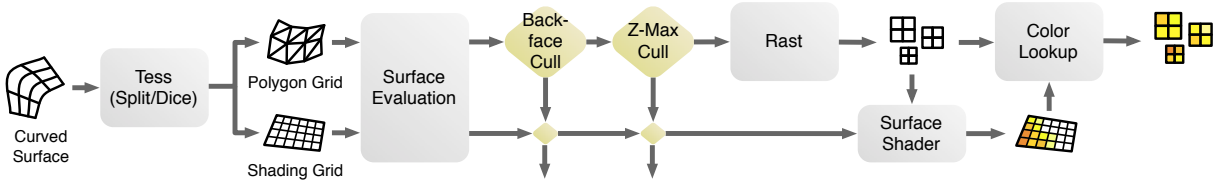

Figure 4: *Data flow diagram of the Reyes pipeline modified to support both the shading grid and lazy shader execution. Rasterization on the polygon grid generates a list of fragments that determine precisely which samples in the shading grid require shading. The fragments look up their final color from the shading grid. Note that both the polygon grid and the shading grid pass through the surface evaluation stage.*

2.2. Image-Space Shading

All modern GPU-accelerated real-time rendering systems couple shading samples to image space visibility samples. The simplest of these approaches is *supersampling*, which executes the surface shader for every visibility sample taken. The technique is inefficient because anti-aliased visibility generally requires a higher sampling rate than does shading. Accordingly, most GPUs today compute one shading sample per pixel regardless of the visibility sampling rate (shaders are expected to band-limit themselves), an optimization known as *multi-sampled anti-aliasing* (MSAA) [Ake93].

The earliest attempt to support blur effects within these systems required rendering the scene multiple times for different lens coordinates u and v and/or time values t, and compositing these results together with an *accumulation buffer* [HA90]. Shading is decoupled from spatial coordinates (x,y), but remains coupled to the values of u,v and t. Stochastic rasterization as proposed by Akenine-Möller et al. retains this partial coupling and requires shader execution per pixel proportional to the number of temporal/lens samples used to resolve motion blur or defocus blur [AMMH07].

Fully decoupling sampling under motion and defocus blur in a way that permits reuse of previously computed shading samples requires managing a complex many-to-one mapping of visibility samples to shading samples. Ragan-Kelley et al. show that this is possible in the context of a stochastic GPU fragment shading pipeline [RKLC*10]. In their decoupling, a memoization cache stores shading results computed at a 5D visibility sample (x,y,u,v,t) that is mapped to a canonical coordinate frame $(x',y',0,0,0)$. When additional shading samples are requested that map to the same coordinate $(x',y',0,0,0)$, the cached value is used if available. This arrangement permits flexible per-primitive shading rates, and reuse of shading samples across pixels in a GPU fragment shading pipeline (Fig. 2, col. 2). Their formalization of the decoupled sampling concept applies to object-space shading methods as well. If Reyes vertex shading is viewed as mapping visibility samples to triangle vertices, our method uses an alternative object-space mapping that avoids vertex coupling. While the resulting benefits of our approach and that of Ragan-Kelley et al. are similar, we achieve them by evolving the Reyes object-space shading architecture rather than by evolving sort-last fragment shading systems.

The main benefit of decoupling sampling from visibility in a GPU is that it permits reuse across pixels under conditions in which objects are blurred. However, current GPUs (and the system implemented by Ragan-Kelley et al.) shade triangles individually and do not track continuity information between adjacent triangles. As a result there is no reuse of shading results at triangle boundaries (Fig. 2, col. 3). GPUs shade on blocks of four pixels (quads) at a time to support finite difference calculations. Each quad is shaded once for each triangle touching it because shading results are only reused within a single triangle. Recent work has proposed a quad-merge pipeline stage to recover the lost continuity information [FBH*10] in a standard GPU fragment pipeline, without blur effects. Their work shows the penalty for not reusing shading samples across triangles is severe in the case of micropolygons half a pixel in area, but still costs a factor of two for triangles 5-7 pixels in area.

Successfully combining the fragment merging technique with either the memoization cache or stochastic rasterization

has the potential to deliver an image-space shading architecture that efficiently supports blur effects with small triangles, but this combination is non-trivial and has not yet been done. The object-space shading architecture presented in this paper provides all the advantages listed in Figure 2 via improvements to the basic Reyes pipeline.

3. The Standard Reyes Pipeline

The canonical micropolygon rendering pipeline is known as the Reyes algorithm [CCC87] and is commonly used for offline cinematic rendering workloads (Figure 3). The renderer receives as input a collection of curved surfaces (though almost any boundary representation can be supported). These surfaces pass through a two-stage tessellation process known as split-dice. During *split*, patches are recursively subdivided into approximately flat *sub-patches* small enough to dice into a micropolygon grid, referred to simply as a *grid*. The polygons must be small enough to meet the user-requested shading rate, typically one half a pixel in area.

The grid passes through a surface evaluation stage (known as the *domain shader* in the DX11 pipeline) that evaluates the surface position, normals, texture coordinates, and other attributes at the vertex locations. This stage also optionally evaluates a displacement shader at all vertices. Surface shading is performed at the vertices prior to rasterization, which is typically done with many jittered samples per pixel. Color is interpolated from the shaded vertices, and fragments are z-tested and blended with the framebuffer.

4. The Shading Grid

To decouple shading samples from vertices (Fig. 2, col. 4), we add an object called a *shading grid* as output from the tessellation stage alongside the usual grid, which we rename the *polygon grid* to emphasize that it contains only surface attributes necessary for visibility. The shading grid is an implicit grid of UV locations on which surface shading takes place. The dimensions of this grid are computed to provide approximately one shade point per pixel. The polygon grid is not shaded, but passes straight to the rasterizer. When a triangle in the polygon grid is rasterized, output fragment colors are sampled and interpolated from the shading grid.

We think of the shading grid as a small multi-channel texture map that is streamed from the tessellator, through the shaders, and discarded after rasterized fragments consume the results. Thus it should always remain in cache, avoiding the long latencies normally associated with texture lookups. Our reference implementation is in software, though hardware texture sampling units could be exploited in a performance implementation.

4.1. Smooth Derivative Values

Shaders often require the derivatives of certain surface parameters with respect to neighboring shading samples. In practice, shading samples are executed in 2x2 blocks and finite differencing is used to compute these derivatives. For a parameter p shaded on a grid in object-space, we wish to compute $\frac{\partial p}{\partial u}$ and $\frac{\partial p}{\partial v}$. However, discrete changes in the object-space sampling rate at grid boundaries can sometimes result in visible artifacts. The solution employed by modern Reyes implementations is described as estimating the "ideal" micropolygon size at a surface location and using that instead of the actual micropolygon size [AG00]. Since the ideal micropolygon edge is one pixel long in screen-space, this is equivalent to computing screen-space derivatives $\frac{\partial p}{\partial x}$ and $\frac{\partial p}{\partial y}$.

The shading grid simplifies this computation. Shading samples are distributed uniformly in object-space across the face of a sub-patch, and finite differencing within 2x2 sample quads provides the derivatives $\frac{\partial p}{\partial u}$ and $\frac{\partial p}{\partial v}$ for surface attribute p. To compute "smooth" derivatives (i.e. screen-space derivatives) we employ the chain rule and evaluate the partial derivatives $\frac{\partial u}{\partial x}$, $\frac{\partial u}{\partial y}$, and $\frac{\partial v}{\partial x}$, $\frac{\partial v}{\partial y}$. The complete derivation is provided in Appendix A.

4.2. Shading Grid Resolution

To accurately achieve the target shading rate, we must estimate the projected screen-space area of a sub-patch prior to rasterization. We use this estimate to determine the resolution of the shading grid. This is similar to the task of determining the tessellation rate of the polygon grid, with the exception that we can ignore potential "T"-junctions at boundaries, since only shading is computed on this grid, not visibility. Therefore we arrange samples in an implicit regular grid spanning the entire sub-patch domain.

During the split phase of tessellation, the tessellator evaluates the projected, displaced screen position of a small series of samples along a patch edge. The screen-space length of this edge is used to determine whether the (sub-)patch needs further splitting, or can be diced. We also use this arc length to approximate the number of shading samples needed along that edge to achieve the target shading rate for that primitive. We round these estimates to the next largest even number so that we can shade at a 2x2 block granularity to allow derivative calculation as described in the preceding section.

However, relying on the screen-space edge lengths alone will result in significant over-shading for sub-patches that are distorted by perspective, or that project to the camera at an oblique angle. This problem is identified by Fisher et al. [FFB*09] and is solved by "interior scaling". They estimate the screen-space area of the sub-patch and evaluate an equation expressing triangle count as a function of that area. We follow the same principle, except with two modifications. First, we estimate the sub-patch area by differential geometry calculations, rather than by assuming it to be four times the largest of its four quadrant areas as in Fisher et al. We empirically determined that our approach gave more consistent results for the shading grid than the approach of

Fisher et al. Second, we compute two scale factors instead of one, to better handle rectangular sub-patches (though the splitter attempts to generate square grids, dimensions may vary by up to a factor of two). We now describe how these two factors are computed.

4.3. Anisotropic Grid Scaling

We can estimate the screen-space area of a shading grid "texel" in a manner similar to the way renderers compute mipmap LOD levels. We will evaluate the differential geometry properties of the sub-patch at each of the four corners, and use them to compute a "best-fit" shading grid resolution in each parametric coordinate direction that will conservatively shade the entire patch at a rate nowhere less than the target shading rate.

At each corner of the shading grid, we compute the partial derivatives $\frac{\partial u}{\partial x}, \frac{\partial u}{\partial y}, \frac{\partial v}{\partial x}, \frac{\partial v}{\partial y}$ as described in Appendix A. These quantities relate the size of a shading sample to the size of a screen pixel. We use these to compute a pair of values indicating the screen-space size of a texel at each corner of the sub-patch:

$$S_u = \sqrt{\left(\frac{\partial u}{\partial x}\right)^2 + \left(\frac{\partial u}{\partial y}\right)^2} \qquad S_v = \sqrt{\left(\frac{\partial v}{\partial x}\right)^2 + \left(\frac{\partial v}{\partial y}\right)^2}$$

Due to curvature, the perspective transform, and displacement mapping, the four pairs of quantities S_u and S_v will usually differ. We take a conservative approach and choose the largest value, and use the reciprocal of this quantity to adjust the estimated shading grid resolution. Note that this does not guarantee a local lower bound on shading rate unless the sub-patch is flat - high frequency variation across the patch could potentially result in a large variation in shading rate in some localized areas within the sub-patch. However the assumption of an approximately linear sub-patch is a reasonable one given the design of the split-dice algorithm. Without using grid scaling, shader executions increase by between 30 and 50% for most scenes, with zone plate and furball exhibiting penalties of more than 100%.

The extra work required to compute the anisotropic scale factors at each corner of the patch is small relative to the cost of over-shading. For sub-patches that are ultimately diced and shaded, the screen-space derivatives at the corners of the patch are required for smooth screen-space derivatives, as explained in Section 4.1. For sub-patches interior to the split tree (those that require further splitting), it only represents a few additional surface evaluations per patch, depending on how careful the implementation is to reuse previously computed values at shared corners and edges.

4.4. High-frequency Displacement

Displacement mapping can distort our estimate of the shading grid resolution. It sometimes means that our assumption of an approximately flat sub-patch fails in an arbitrary and unpredictable way. In particular, high frequency detail can introduce wildly inaccurate results since the scale factors are calculated from the surface orientation locally at the corners.

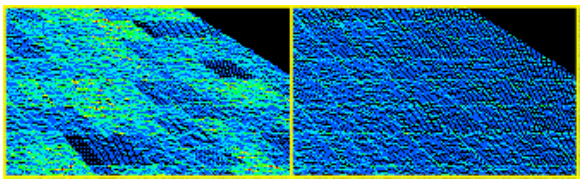

Figure 5: *High frequency displacement mapping can obstruct estimation of shading grid resolution if not properly filtered, shown on the left. Dark blue pixels indicate one shader execution, and warmer colors indicate additional shader executions. Displacement detail is filtered in the right image as described in Section 4.4, resulting in more accurate estimation of shading grid resolution.*

To address this issue, we enlarge the displacement sampling filter width when computing the grid scale factors described in Section 4.3. Our filter is half the width of the sub-patch's longest edge, effectively filtering out displacement frequencies that do not meaningfully contribute to the general orientation of the patch. The resulting improvement of this filter bias is shown in Figure 5. Note that both shading samples and grid vertices are ultimately computed using the unfiltered displacement shader, and this approximation does not limit the detail visible in the final rendered image. Geometric detail is only limited by the size of the triangles output from the tessellator, which in our system is user-specified on a per-primitive basis.

5. Lazy Shader Execution

The second pipeline modification we implement is post-visibility shader execution. The traditional Reyes pipeline shades the entire grid if any portion of it is conservatively estimated to be visible. This results in over-shading at silhouettes, frustum boundaries, and where occluding geometry is fine-grained (Fig. 2, col. 1).

During rasterization, visibility samples that pass an early Z test are stored in a *fragment buffer*. Shading grid samples required to color these fragments are tagged as "requested" if they are not already shaded. Processors shade the requested samples in 2x2 blocks (quads) to support finite difference calculations: if any sample in a quad is requested, the entire quad is shaded. After shading, waiting fragments interpolate color from their UV location in the shading grid and proceed to the frame buffer for blending.

The fragment buffer stores sample hits generated by the rasterizer. Data include depth, parametric UV coordinates, and a coverage mask if the rasterizer delivers fragments in

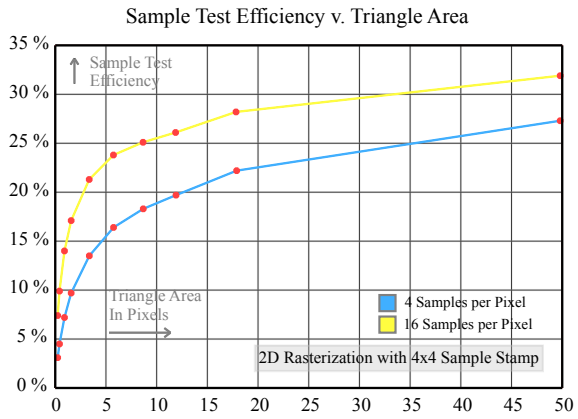

Figure 6: *Rasterization efficiency increases rapidly as triangles grow from sub-pixel sizes to about 5 pixels in area, but more slowly thereafter. These data points were generated from the big guy scene, though all tested scenes produced very similar curves.*

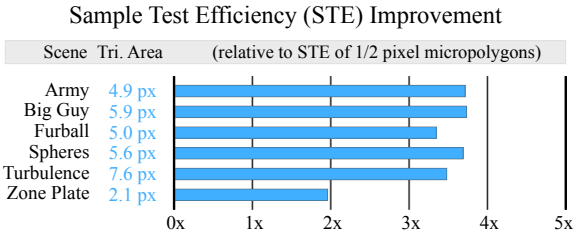

Figure 7: *Observed improvement in STE for our test scenes as a direct result of increased triangle size. Sample testing accounts for approximately 65% of both stochastic and stamp rasterization costs [FLB*09].*

blocks. The UV coordinates are used to request and later query samples in the shading grid. We use a fragment buffer large enough to cover 1024 pixels, the largest allowable shading grid in our configuration.

The lazy shading method described here could be implemented without the shading grid, i.e. while shading at the vertices of a micropolygon grid. Identifying which vertices are requested for a given fragment becomes slightly simpler, and fragments store triangle index and barycentric coordinates instead of grid UV coordinates to facilitate lookup after vertex shading. Derivative calculation becomes slightly more complicated, however.

6. Evaluation

We evaluate image quality and algorithmic performance with three pipelines. A Reyes vertex shading pipeline as shown in Figure 3, a modified pipeline that uses the shading grid, and a lazy shading grid pipeline that shades after rasterization as shown in Figure 4. Entirely backfacing grids are discarded after surface evaluation and displacement. This is followed by a Z-max occlusion culling stage. A low resolution Z-buffer with a guaranteed conservative Z value for every 8x8 pixel block is maintained and used to cull occluded grids. This type of culling is commonly employed in today's real-time systems and is simple to implement [AMHH08].

We chose a variety of scenes that exhibit a range of depth complexity and geometric detail, including displacement mapped geometry. The spheres, army, and furball scenes illustrate coarse, medium, and fine grain occlusion, respectively. The turbulence and zone plate scenes consist of a single surface with displacement mapping. The zone plate is particularly challenging since the displacement introduces significant self-occlusion and many silhouette edges.

In our results, we use a maximum shading grid size of 1024 samples. This choice follows the RenderAnts system [ZHR*09], but differs from the RenderMan specification default of 256 samples [AG00]. Like Zhou et al. we empirically determined that the benefits of larger grid sizes for a real-time system outweigh their costs, but it is particularly so in light of our enhancements. The benefits include less redundant work at grid boundaries, and increased available parallelism within a grid. The costs of larger grids in a non-lazy Reyes shading system include increased over-shading at silhouette edges and tile boundaries, and less effective coarse Z culling of grids. *Our modifications directly attack these disadvantages, making larger grids even more desirable.*

We also implement a sort-middle pipeline, where input primitives are bucketed into screen-space tiles, which are then processed independently and in parallel. Many real-time systems are sort-middle [AMHH08], and we will show that lazy shading can significantly improve the efficiency of object-space shading in a sort-middle pipeline. More flexible approaches to sorting and scheduling such as that used in RenderAnts [ZHR*09] merit additional research, but future scaling of many-core hardware architectures are likely to reward systems with less synchronization.

6.1. Shading Grid Evaluation

The purpose of the shading grid is to decouple the shading samples from the vertices, thereby allowing the system to use larger triangles where the geometry is less detailed. Our observational experiments with the scenes shown in Figure 10 indicate that a maximum triangle edge length of 4.0 pixels is more than sufficient except in the case of zone plate, where we use 3.0 because of the high frequency displacement shader. An ideal tessellator would adaptively choose the triangle size, concentrating smaller triangles at silhouette edges and curves, and using larger triangles in smoother areas. The development of such a "smart" tessellator is beyond the scope of this paper, and we use the DiagSplit split-dice

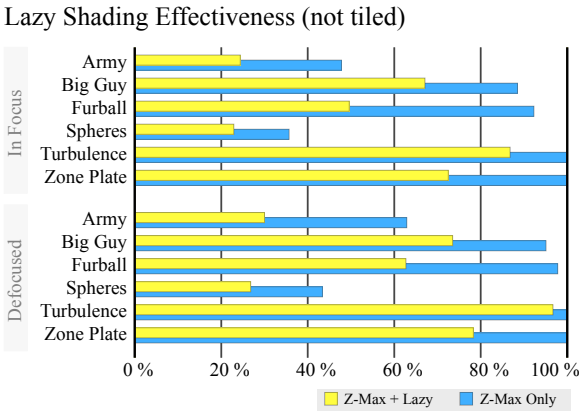

Figure 8: *Lazy shading reduces shader execution for occluded and partially occluded grids. Z-Max culling only culls entire grids against a conservative low-resolution Z-buffer. Scenes such as Army and Furball benefit the most from fine grained occlusion culling. 100% is defined as shading all front-facing geometry (shorter bars are better). We used a maximum shading grid size of 1024 samples, as described in Section 6.*

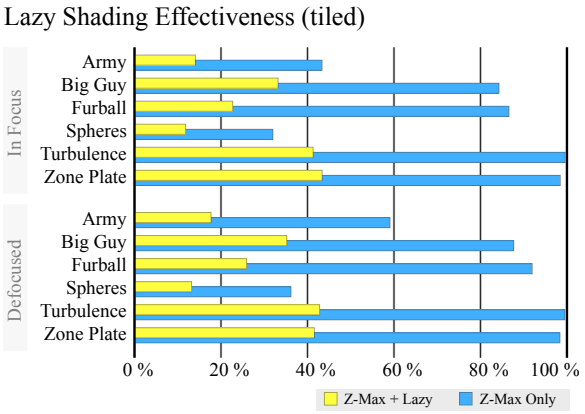

Figure 9: *Same as Figure 8, except with sort-middle screen space tiling (tiles are 64x64 pixels). Lazy shading is more effective relative to Z-Max culling in this case because of bin-spread, discussed in Section 6.2.*

tessellation algorithm by Fisher et al. [FFB*09]. Our configuration results in a triangle count reduction in all scenes between 8x-10x, and an increase in average triangle area from 0.4 pixels to between 4.9 and 7.0 pixels.

A ten-fold increase in triangle size reduces visibility costs in two ways. First, fewer triangles means fewer per-triangle costs such as bounding box computation. Second, larger triangles result in improved sample test efficiency (STE), defined as the ratio of samples covered to samples tested. Figure 6 illustrates the relationship between triangle size and STE, and Figure 7 shows the observed STE improvement for our test scenes. Sample testing and bounding box computation comprise between two thirds to three quarters of rasterization costs by operation count [FLB*09]. This implies that a 3.5x improvement in STE, as we show for most scenes in Fig 7, reduces visibility costs by nearly 2x. While our choice of tessellation rate (4.0 pixel edge length) is somewhat arbitrary, we have been very conservative in our quality judgements, especially in view of the reduced performance benefits of further increasing triangle size.

Like Reyes vertex shading and Ragan-Kelley et al., the shading grid allows the user to specify an arbitrary shading rate, expressed in shading samples per pixel. We use a value of 1.0, though other values may be useful in certain shaders or other conditions. For example, it may be acceptable to reduce the shading resolution for blurred objets. Note that unlike with image-space shading systems, this parameter is only a target, not a guaranteed result. Some grids may have fewer, or more than, one shading sample per pixel on average. The quality of the resulting shading is similar to that of vertex shading, with slightly more variation between surfaces, which we attribute to the constraint that grid dimensions are rounded-to-even, and the slightly larger average grid size. Occasionally we have observed a few significantly under-shaded grids in anomalous cases, but the slightly blurry results in these cases are temporally transient and unlikely to be a problem for most real-time applications. This problem is not unique to our method, but related to the general problem of assigning shading points in object space according to a desired screen-space distribution.

There are two new potential sources of artifacts that are worth pointing out, even though we did not observe them. First, the shading grid generates shading samples at 3D locations which lie on the limit surface, but not necessarily on the surface defined by the tessellated polygon grid. It is unclear whether secondary visibility calculations performed at shading samples will therefore fail in unexpected ways. It may, for example, exaggerate depth bias issues with shadow maps, or self-intersections for secondary rays cast from shading locations. The second possibility results from the fact that samples along shading grid boundaries are not "aligned" in the way that border vertices must align to avoid cracks in the mesh. In principle (and in our experience) this is not a problem, but it is imaginable that some shaders may exaggerate numerical inconsistencies in surface attributes along these edges.

6.2. Lazy Shading Evaluation

The lazy shading mechanism is designed to give object-space shading architectures the ability to avoid shading occluded or otherwise invisible geometry at a granularity finer than an entire grid, thus resolving one of its largest disad-

vantages relative to fragment shading architectures. The first significant cause of overshading in Reyes is partial occlusion of grids, either due to silhouette edges or depth complexity. Coarse occlusion culling can be performed without shading lazily via Z-max occlusion culling [AMHH08]. This form of culling is inherently conservative: roughly a quarter to a third of shaded grids in our tested scenes were fully occluded but escaped Z-max culling. This culling pass is ineffective for scenes with many silhouette edges or small thin occluders (zone plate, furball). The zone plate scene is a special case where the displacement function creates significant occlusion and many silhouette edges. Since the Z sort occurs prior to tessellation and displacement, grids are not rendered in Z order and occlusion culling is ineffective altogether. Figure 8 illustrates the effectiveness of lazy shading on the tested scenes.

The second major cause of over-shading that we analyze is an artifact of screen-space tiling. The original Reyes system tiled the frame buffer and rendered each tile serially to minimize the renderer's memory footprint. Grids which overlapped a tile boundary were shaded once and "pushed" to neighboring tiles and reused. In modern real-time tiled renderers, however, tiles are processed independently and in parallel without the synchronization burdens of passing shaded grids between neighboring tiles. This results in redundant shading of grids that overlap tile boundaries. The percentage increase in shading due to screen-space tiling is called *bin-spread*. A comparison of Figures 9 and 8 suggests that tiling tends to cause more over-shading than occlusion. Without lazy shading, a real-time tile-based renderer will suffer from significant over-shading at tile boundaries.

Table 1 quantifies the amount of over-shading caused by 64x64 pixel tiling against 1024 sample grids. Shading lazily can reduce the bin-spread overhead to around 12% for all scenes in the absence of screen-space blur effects. When surfaces are blurred and visible in multiple buckets, however, redundant shading is forced by the requirement that the tiles consist of independent workloads. The remaining bin-spread is proportional to the size and quantity of blur in the scene, and lazy shading cannot address this inefficiency. The army, big guy, and spheres scene have blur widths similar to the tile width, resulting in around 2x bin-spread even with lazy shading (See Ragan-Kelley et al. for a detailed examination of the relationship between blur effects and tile size [RKLC*10]). This penalty is significant, and alternative sorting and scheduling algorithms for real-time object-space shading renderers may be worth further investigation.

Shading Bin Spread, 64x64 pixel tiles				
Scene	Sharp		Depth-of-Field	
	Eager	Lazy	Eager	Lazy
Army	73.3%	10.1%	203.1%	90.4%
Big Guy	118.3%	13.5%	317.3%	116.6%
Furball	120.6%	7.8%	167.0%	17.4%
Spheres	96.7%	12.8%	205.5%	80.2%
Turbulence	136.1%	12.8%	224.1%	44.0%
Zone Plate	86.0%	13.0%	170.0%	45.9%

Table 1: *Lazy shading significantly reduces the costs of* bin-spread *in a tile based renderer. Bin spread is defined as the percentage increase in shader execution when the screen is tiled versus execution for a single full-screen tile. Blur effects worsen the problem by increasing the screen-space area of surfaces, causing them to spill into neighboring bins.*

7. Costs and Limitations

There are several storage and computation overheads associated with the changes we've made to the Reyes pipeline. These costs are reasonable relative to the existing demands of the system and the benefits derived from them for typical scenes. We also describe a few of the practical limitations of our approach.

The limit surface is evaluated both at the shading samples in the shading grid, and at the vertex locations in the polygon grid. With a shading rate of 1.0, the ratio of shading samples to vertices is roughly on the order of the number of pixels per triangle, approximately 6:1 for our scenes, implying a 16.7% increase in work. Furthermore, not all surface attributes are evaluated at the vertices of the polygon mesh (only those required to compute vertex position). The additional storage for this polygon grid is small relative to the size of the shading grid, based on a similar analysis.

The more interesting overhead is the shading grid color lookup following rasterization and shading. In a vertex shading system, the color is interpolated from the vertices using the barycentric coordinates of the screen-sample. In our system, the parametric u, v coordinates are interpolated from the vertices, and then used to perform an interpolated lookup from the shading grid. As mentioned, the shading grid should be small enough so that it remains in on-chip cache to avoid latency. Furthermore, this operation is well understood and is frequently implemented in hardware even as GPUs trend toward more general computing capabilities [SCS*08]. The precise break-even point where the computational cost of shading grid resampling exceeds the benefits of larger triangles depends largely on the implementation.

The lazy shading mechanism requires additional storage for the fragment buffer. In our software implementation, with 4 samples per pixel and 4 pixels per stamp, each entry is 74 bytes (per sample depth, coverage mask, and a single UV pair). For a 1024 pixel buffer, this comes to just under 19 kilobytes. This is an additional cost relative to what a traditional Reyes system requires, but it is similar to what real-time fragment shading renderers already require, as often hundreds of fragments (which include all the above data

plus surface attributes for the shader) are in-flight simultaneously and require storage on chip.

The utility of the shading grid is predicated on two assumptions. First, we assume that the limit surface can be directly evaluated at an arbitrary parametric coordinate. Some subdivision surfaces lack this property and would be difficult to efficiently support. Second, we assume the split phase of tessellation produces quadrilateral domains. A triangular domain may be supported by creating a degenerate edge in the shading grid, but this would exhibit poor shading efficiency and most likely create aliasing problems near the degenerate edge. We finally note that the benefit of larger triangles as a result of decoupling shading from vertices cannot be capitalized on when per-pixel detail is necessary to adequately render a surface. This may be the case with very detailed displacement, highly curved surfaces, or when the input primitives themselves are very small.

8. Future Work and Conclusion

We have presented a shading architecture which satisfies four important performance criteria that no other current system can completely offer. Our modified Reyes pipeline allows fine-scale occlusion culling and relaxed triangulation requirements, matching the advantages fragment shading systems have had over Reyes. We also preserve Reyes' natural advantages in handling camera blur effects and avoiding redundant shading operations at small triangle boundaries - advantages which no current fragment shading architecture fully provides.

While we present our analysis on an algorithmic level, we do so with an eye toward interactive rendering on future many-core parallel hardware. As future work, an implementation on a suitable hardware platform in the spirit of previous efforts to implement Reyes for interactive rendering would be worthwhile. Additionally, an adaptive tessellator that uses small triangles only where they are needed (taking into consideration silhouettes and displacement) would better exploit our decoupled shading mechanism.

Our results further the goal of eventually developing a real-time rendering pipeline that efficiently supports accurate depth-of-field and motion blur, as well as high geometric complexity. They are most likely to be of interest to those seeking to improve the performance of the Reyes algorithm and adapt it to the needs of parallel real-time applications. Ongoing research seeks to achieve largely the same goals by addressing inefficiencies in current fragment shading systems [FBH*10] [RKLC*10]. A shading solution fit for modern parallel hardware that meets our efficiency criteria may ultimately have elements of both Reyes and GPU pipelines.

Acknowledgements

The authors would like to thank everyone who took the time to carefully read early drafts of this work, especially Greg Johnson, Manfred Ernst, and Warren Hunt and the rest of the Visual Applications Research group at Intel Labs. We also especially thank Solomon Boulos, Pat Hanrahan and the Stanford Graphics Lab for advice and for providing access to crucial source code and test scenes.

References

[AG00] APODACA A. A., GRITZ L.: *Advanced RenderMan: Creating CGI for Motion Pictures*. Morgan Kauffman, 2000.

[Ake93] AKELEY K.: RealityEngine graphics. In *Proceedings of SIGGRAPH 93* (1993), Computer Graphics Proceedings, Annual Conference Series, ACM, ACM Press / ACM SIGGRAPH, pp. 109–116.

[AMHH08] AKENINE-MÖLLER T., HAINES E., HOFFMAN N.: *Real-Time Rendering*, 3 ed. A. K. Peters, Ltd., 2008.

[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.

[CCC87] COOK R., CARPENTER L., CATMULL E.: The Reyes image rendering architecture. In *Computer Graphics (Proceedings of SIGGRAPH 87)* (1987), vol. 27, ACM, pp. 95–102.

[FBH*10] FATAHALIAN K., BOULOS S., HEGARTY J., AKELEY K., MARK W. R., HANRAHAN P.: Reducing shading on gpus using quad-fragment merging. In *SIGGRAPH '10: Proceedings of the 37th annual conference on Computer graphics and interactive techniques* (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. *ACM Transactions on Graphics 28*, 5 (2009), 1–10.

[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.

[HA90] HAEBERLI P., AKELEY K.: The accumulation buffer: hardware support for high-quality rendering. In *SIGGRAPH '90: Proceedings of the 17th annual conference on Computer graphics and interactive techniques* (1990), vol. 24, ACM, pp. 309–318.

[Mic10] MICROSOFT: Programming guide for Direct3D 11, 2010. http://msdn.microsoft.com/en-us/library/ff476345.

[PO08] PATNEY A., OWENS J. D.: Real-time Reyes-style adaptive surface subdivision. *ACM Transactions on Graphics 27*, 5 (2008), 1–8.

[RKLC*10] RAGAN-KELLEY J., LEHTINEN J., CHEN J., DOGGETT M., DURAND F.: *Decoupled sampling for real-time graphics pipelines*. Tech. Rep. MIT-CSAIL-TR-2010-015, MIT Computer Science and Artificial Intelligence Laboratory Technical Report Series, 2010.

[SCS*08] 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., HANRAHAN P.: Larrabee: a many-core x86 architecture for visual computing. *ACM Transactions on Graphics 27*, 3 (2008), 1–15.

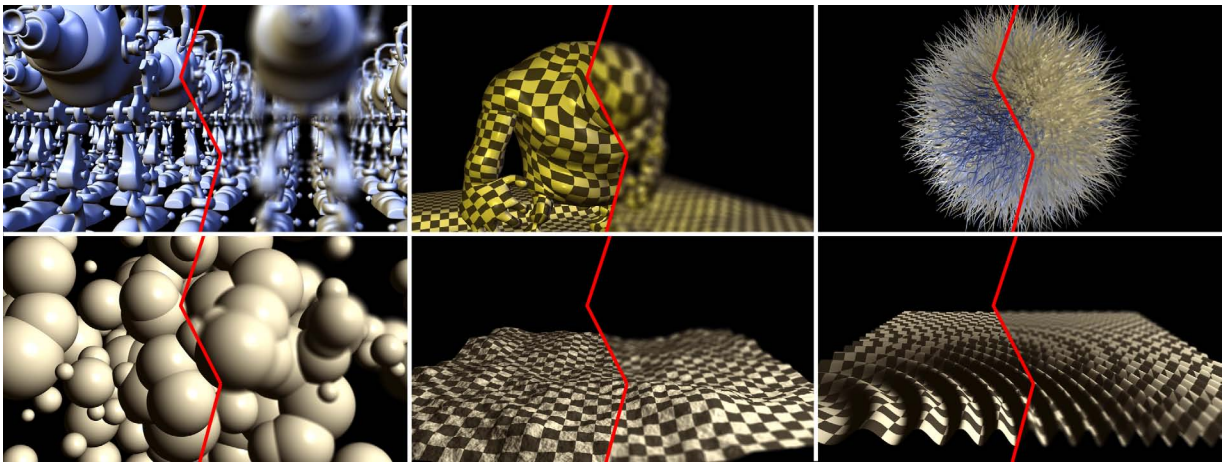

Figure 10: *Suite of test scenes used. All images were rendered at 1920x1080 resolution and 64x64 pixel screen tiles. The right half of each image displays the camera defocus effect as used in reported statistics. From left to right, top to bottom: army, big guy, furball, spheres, turbulence, zone plate. Sharp renderings use 4 samples per pixel, and defocus renderings use the interleaved stochastic rasterization algorithm from Fatahalian et al. with 16 samples per pixel [FLB*09].*

[ZHR*09] ZHOU K., HOU Q., REN Z., GONG M., SUN X., GUO B.: RenderAnts: interactive reyes rendering on gpus. vol. 28, ACM, pp. 1–11.

Appendix A: Screen-space derivatives derivation

Here we present the method by which the screen-space partial derivatives of a surface attribute p are computed in object-space prior to rasterization.

We assume the surface attribute p is a function of surface parameters u and v, and the camera projection determines the functions $x(u,v)$ and $y(u,v)$ which give the screen location (x,y) of an arbitrary surface location (u,v). Therefore, the screen-space partial derivatives of $p(x(u,v),y(u,v))$ are given by the chain rule:

$$\frac{\partial p}{\partial x} = \frac{\partial p}{\partial u}\frac{\partial u}{\partial x} + \frac{\partial p}{\partial v}\frac{\partial v}{\partial x} \qquad \frac{\partial p}{\partial y} = \frac{\partial p}{\partial u}\frac{\partial u}{\partial y} + \frac{\partial p}{\partial v}\frac{\partial v}{\partial y} \tag{1}$$

We use finite differencing in object-space to determine partial derivatives with respect to coordinates (u,v). Since $x(u,v)$ and $y(u,v)$ are not generally invertible functions, our strategy is to convert partials with x or y in the denominator to expressions only containing partials with u or v in the denominator, and perform finite differencing in an object-space coordinate grid. If we take a small step in the u coordinate direction and measure the difference in screen-space coordinates, we compute $\Delta s_u = (\frac{\partial x}{\partial u}, \frac{\partial y}{\partial u})$. This is the first column of the jacobian matrix:

$$J = \begin{bmatrix} \frac{\partial x}{\partial u} & \frac{\partial x}{\partial v} \\ \frac{\partial y}{\partial u} & \frac{\partial y}{\partial v} \end{bmatrix}$$

The dot-product of this vector Δs_u with the gradient of u with respect to screen-space coordinates (x,y) gives the u component of the vector in object space, which we know to be 1. A similar calculation involving a finite difference in the v direction (second column of J) gives us a system of two equations with unknowns $\frac{\partial u}{\partial x}$ and $\frac{\partial u}{\partial y}$:

$$\Delta s_u \cdot \nabla u = \frac{\partial x}{\partial u}\frac{\partial u}{\partial x} + \frac{\partial y}{\partial u}\frac{\partial u}{\partial y} = 1 \tag{2}$$

$$\Delta s_v \cdot \nabla u = \frac{\partial x}{\partial v}\frac{\partial u}{\partial x} + \frac{\partial y}{\partial v}\frac{\partial u}{\partial y} = 0 \tag{3}$$

Solving these equations gives:

$$\frac{\partial u}{\partial x} = \frac{\partial y}{\partial v}(det(J))^{-1} \qquad \frac{\partial u}{\partial y} = -\frac{\partial y}{\partial u}(det(J))^{-1} \tag{4}$$

A similar system of equations can be set up using ∇v, providing solutions to solve for $\frac{\partial v}{\partial x}$ and $\frac{\partial v}{\partial y}$. Using these results, combined with the chain rule in Equations (1), and simplifying, we have our final expressions for the screen-space derivatives:

$$\frac{\partial p}{\partial x} = \frac{\partial p}{\partial u}\left[\frac{\partial y}{\partial v}(det(J))^{-1}\right] + \frac{\partial p}{\partial v}\left[\frac{\partial y}{\partial u}(-det(J))^{-1}\right] \tag{5}$$

$$\frac{\partial p}{\partial y} = \frac{\partial p}{\partial u}\left[-\frac{\partial y}{\partial u}(det(J))^{-1}\right] + \frac{\partial p}{\partial v}\left[\frac{\partial y}{\partial v}(det(J))^{-1}\right] \tag{6}$$