Decoupled Sampling for Graphics Pipelines

JONATHAN RAGAN-KELLEY and JAAKKO LEHTINEN and JIAWEN CHEN
MIT CSAIL
and
MICHAEL DOGGETT
Lund University
and
FRÉDO DURAND
MIT CSAIL

We propose a generalized approach to decoupling shading from visibility sampling in graphics pipelines, which we call *decoupled sampling*. Decoupled sampling enables stochastic supersampling of motion and defocus blur at reduced shading cost, as well as controllable or adaptive shading rates which trade off shading quality for performance. It can be thought of as a generalization of multisample antialiasing (MSAA) to support complex and dynamic mappings from visibility to shading samples, as introduced by motion and defocus blur and adaptive shading. It works by defining a many-to-one hash from visibility to shading samples, and using a buffer to memoize shading samples and exploit reuse across visibility samples. Decoupled sampling is inspired by the Reyes rendering architecture, but like traditional graphics pipelines, it shades *fragments* rather than micropolygon vertices, decoupling shading from the geometry sampling rate. Also unlike Reyes, decoupled sampling only shades fragments after precise computation of visibility, reducing over-shading.

We present extensions of two modern graphics pipelines to support decoupled sampling: a GPU-style sort-last fragment architecture, and a Larrabee-style sort-middle pipeline. We study the architectural implications of decoupled sampling and blur, and derive end-to-end performance estimates on real applications through an instrumented functional simulator. We demonstrate high-quality motion and defocus blur, as well as variable and adaptive shading rates.

Categories and Subject Descriptors: I.3.7 [**Computer Graphics**]: Three-Dimensional Graphics and Realism; I.3.6 [**Computer Graphics**]: Methodology and Techniques; I.3.1 [**Computer Graphics**]: Graphics Hardware—*Graphics Processors*

Additional Key Words and Phrases: Antialiasing, Defocus Blur, Depth of Field, Graphics Hardware, Graphics Pipeline, Motion Blur, Reyes

1. INTRODUCTION

In modern real-time rendering, shading is very expensive. This is mirrored in hardware: an increasing majority of GPU resources are dedicated to complex shader and texture units, while much of the rest of the graphics pipeline—including rasterization and triangle setup—is small by comparison. Effects such as motion and defocus blur that require heavy sampling over a 5D domain (pixel area, lens aperture, and shutter interval) can therefore be very expensive if the shading cost increases linearly with the number of samples, as is the case with stochastic rasterization or an accumulation buffer. As a result, these effects usually must be approximated with heuristics for real-time applications. However, while high quality antialiasing, motion, and defocus blur do require taking many samples of the visibility function over a 5D domain, shading usually does not vary dramatically over the shutter interval or lens viewpoints, and can be prefiltered.

In this paper, we introduce *decoupled sampling*, which separates the shading rate from visibility and geometry sampling for motion blur, defocus blur, and variable shading rates in graphics pipelines. We seek to shade at a lower rate—for example, approximately once per pixel—but sample visibility densely to enable supersampling effects at a reduced cost. Decoupled sampling is inspired by multisample antialiasing (MSAA) and RenderMan's Reyes architecture [Akeley 1993; Cook et al. 1987], which re-use the same shaded color for multiple visibility samples. Multisampling computes the color of a triangle once per pixel but supersamples visibility (Fig. 2), achieving antialiasing without increasing shading cost. However, MSAA is limited to edge antialiasing, and a core contribution of this paper is to extend this principle to motion blur, defocus blur, and variable-rate or non-screen-space shading. These effects are challenging for MSAA because the correspondence between shaded values and visibility samples becomes irregular. For edge antialiasing, a color shaded at a given pixel applies only to the visibility samples inside that pixel's footprint, so there is a fixed number of visibility samples per shading sample, and their correspondence is both local (within the pixel) and static (fixed, independent of the input). In contrast, for an effect such as motion blur, a fragment-sized region of a triangle can be smeared over the screen, and the correspondence depends on the motion.

Like Reyes, we seek to leverage the assumption that a scene point's color is roughly constant over the shutter interval, and from all views on the lens. This assumption is key to enabling reuse of shaded values. But unlike fragment shading graphics pipelines, Reyes dices the geometry into micropolygons and shades them before fully computing visibility. Shading before visibility leads to overshading, where micropolygons are shaded but do not contribute to the image because they fall between visibility samples or are occluded. Modern implementations of Reyes use occlusion culling to reduce overshading due to visibility, but this culling is inherently conservative. Reyes also requires sampling geometry as finely as shading, even for simple or flat surfaces: it decouples shading from visibility, but *couples* the shading rate to the geometry sampling rate. This is potentially wasteful: just splitting and dicing into micropolygons is complex and expensive, and rasterizing micropolygons is dramatically more expensive than rasterizing fewer large triangles [Fatahalian et al. 2009].

We set out to decouple shading from both visibility and geometry sampling rates, in the presence of complex motion and defocus blur, while shading only after resolving precise visibility. In decoupled sampling, we explicitly map visibility samples to their corresponding shading values using a deterministic mapping func-

Fig. 1. Efficient defocus blur using *decoupled sampling* in a recent Direct3D game, with 27 visibility samples per pixel. The heat map encodes the amount of shading required over the image. Existing supersampling techniques would require 27 shading samples (red) over the entire image, which is prohibitively expensive, while decoupled sampling achieves similar quality with an overall average shading cost of only 1.67 samples per pixel, similar to current edge multisampling antialiasing (MSAA) with no defocus or motion blur. The visible tile structure in the heat-map is due to the rasterizer scanning pattern.

tion from the visibility domain (sampling over screen space, time, and aperture) to a well-defined shading grid. This function accounts for motion and defocus blur, shading rate variation, and any other properties which influence the relationship. Shading is performed on demand, triggered by visible samples, but at a lower rate than visibility. Memoization—remembering already-computed shading values and reusing them when shading is requested again at exactly the same location—enables reuse across many visibility samples. Rasterization and shading operate directly on the input scene polygons, which limits rasterization cost relative to micropolygon rendering, but does not intrinsically enable displacement mapping. Similar to MSAA and Reyes, decoupled sampling stores the framebuffer at the full supersampled resolution, and the savings relative to brute force supersampling come from the fact that multiple of those samples receive the same shading result. Together, this enables supersampling effects such as defocus and motion blur at greatly reduced shading cost, and also supports controllable and adaptive per-primitive shading rates to trade shading cost for selective super- or sub-sampling of shader evaluation, depending on shader frequency content.

We present implementations of decoupled sampling that extend two graphics pipeline architectures: a sort-last fragment pipeline, similar to current GPUs, and the tile-based sort-middle pipeline of Intel's Larrabee [Molnar et al. 1994; Eldridge 2001; Seiler et al. 2008]. Our implementation and evaluation focused on three primary levels of validation:

- A feature-complete functional simulator of Direct3D 9 augmented with decoupled sampling and stochastic rasterization to validate the feasibility and resulting image quality of the algorithms on real content, and its interaction with the subtleties of a real API.
- Implementation in simulated parallel pipelines to confront major architectural issues and validate the feasibility and correctness of the proposed designs.
- Microbenchmarks, performed on real content with a combination of instrumentation and microarchitectural simulation in our functional simulator, to isolate and study the effects of the key design choices in our proposed real-world architectures.

2. RELATED WORK

2.1 Antialiasing, Motion, and Defocus Blur

Supersampling. High-quality antialiasing, motion, and defocus blur can be computed by supersampled rendering, e.g. using an accumulation buffer [Haeberli and Akeley 1990] or stochastic rasterization [Cook et al. 1984; Cook 1986; Akenine-Möller et al. 2007]. Stochastic rasterization with motion and defocus blur is expensive, but recent results suggest that hardware implementation will make it feasible in the near future [Fatahalian et al. 2009; Brunhaver et al. 2010]. Similarly, recent progress has improved the efficiency of hierarchical Z culling in the presence of blur [Boulos et al. 2010]. Accumulation buffer rendering works by rendering samples in time and on the lens as successive rendering passes. This linearly scales the load across the entire pipeline and is generally too expensive for modern games. Stochastic rasterization supersamples only the fragment stages (not the per-vertex computation) of the graphics pipeline, saving substantial overhead, but it still scales linearly in fragment shading.

2D Blur Effects. Motion and defocus blur can also be approximated via a family of 2D blur techniques applied directly to a conventional, non-blurred framebuffer [Hammon 2007; Rosado 2007]. These techniques require only a small framebuffer and perform well on current hardware, but they suffer from approximations and ambiguity in occlusion. Breaking the scene into layers reduces occlusion artifacts, but cannot eliminate them with modest numbers of layers [Max and Lerner 1985]. Recent layer-based algorithms [Lee et al. 2009] can perform well, but at the cost of substantial algorithmic complexity and numerous heuristics. In the limit, making layered rendering fully support occlusion would require a layer per primitive for shading. In effect this is exactly what decoupled sampling does, but on-demand and in a fine-grained fashion within the rendering pipeline, rather than as a series of global passes through memory. To date, accurate stochastic visibility sampling remains the method of choice for production-quality rendering.

2.2 Decoupled Shading and Reuse

Reyes. Decoupled sampling is inspired by Reyes's separation of shading and visibility rates. However, rather than using micropolygons, it rasterizes the input geometry directly, and shading is driven by visibility samples, as in conventional graphics hardware. Reyes pipelines have been implemented on stream processors and GPUs [Owens et al. 2002; Zhou et al. 2009]. The problem of parallel generation of micropolygons by splitting and dicing has been recently treated by Patney and Owens [2008] and Fisher et al. [2009]. In contrast, we focus on enabling effects that require high visibility sampling rates and on enabling shading reuse in the presence of blur.

Note that, in contrast to a simple single-layered framebuffer, Pixar's RenderMan implementation uses an A-buffer, which stores a sorted list of visible micropolygons per sample. This enables order-independent transparency, but requires complex, variable-rate storage. We view the problem of order-independent transparency as orthogonal to the shading architecture and stochastic visibility sampling; our approach can benefit from similar multi-layer framebuffer algorithms.

Lightspeed's indirect framebuffer eliminates overshading and enables interactive rendering of motion blur, defocus blur, and transparency for relighting by preprocessing Reyes' full visibility structure for a fixed scene and viewpoint [Ragan-Kelley et al. 2007]. Shadermaps [Jones et al. 2000] precompute time- and view-independent parts of a shader to save computation over an entire animation.

Recent work also has applied our model of explicitly decoupled sampling in the context of object-space shading, improving on Reyes by decoupling shading from both visibility and geometry sampling, and evaluating shading only after precise visibility [Burns et al. 2010]. This is decoupled sampling with a mapping function defined between visibility samples and an *object-space* instead of image-space shading grid.

Multisampling. MSAA can be seen as a specialized instance of decoupled sampling where only subpixel x, y locations are sampled [Akeley 1993]. It does not support motion blur, defocus, or flexible shading domains since it requires a static, one-to-n relationship between shading and visibility samples. Motion and defocus blur can cause a one-pixel-sized region of a triangle, which MSAA tries to treat as a single shading sample, to influence variable numbers of visibility samples, potentially across many pixels. Fragment merging uses buffering to opportunistically share shading between micropolygons without blur, but it still builds on the traditional fixed screen space mapping of MSAA [Fatahalian et al. 2010].

Caching. A different way of reducing shading cost is to reuse shaded values opportunistically from frame to frame. The Microsoft Talisman architecture [Torborg and Kajiya 1996] sought heavy reuse by using 2D warping to update the full 3D view at a lower frame rate. In contrast, we seek full 3D rendering but save on shading costs. Reprojection caching techniques, including the Reverse Reprojection Cache [Nehab et al. 2007; Sitthi-Amorn et al. 2008] and the approximation cache in Hasselgren's Multiview Rasterization Architecture [2006], reuse values from previous frames or views to approximate shading. When a cache miss occurs, they recompute shading in the new view, but do not update the cache for nearby samples to reuse. In decoupled sampling, when a cache miss occurs, a shaded value is computed and the cache index is updated immediately for subsequent samples to reuse. Shading is performed in a single, well-defined discrete domain, so results are deterministic regardless of whether they hit or miss in the cache.

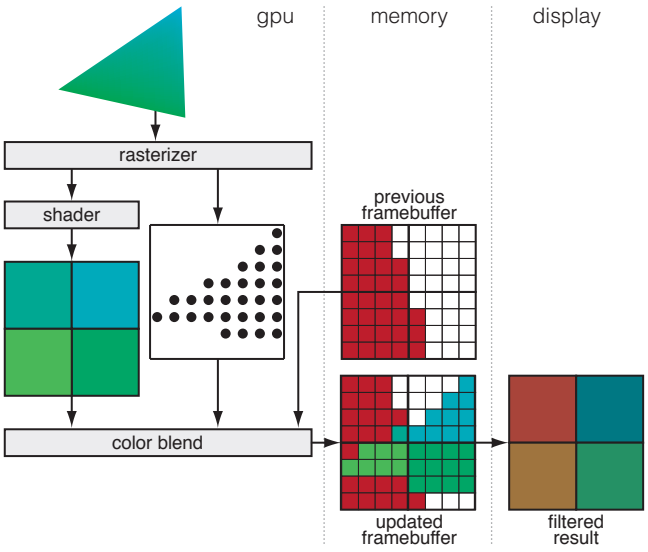

Fig. 2. Multisample antialiasing in a conventional GPU pipeline. Edge-tests are computed and visibility is processed (in the framebuffer) at super-sampled resolution, while shading is computed at pixel resolution. The final display is filtered down to pixel resolution from the supersampled framebuffer. This decouples shading from visibility rates for the special case of a regular and static relationship between shading and visibility, where both are aligned in screen-space, but this regular correspondence breaks down under effects like motion and defocus blur.

This is essential for avoiding artifacts due to discontinuities in the subpixel spacing at which shading was performed (Fig. 3). Decoupled sampling is also robust to occlusion and transparency, because it only reuses shading within a primitive. Further, Talisman and reprojection caching store and fetch from the entire previous frame buffer which must be streamed in from texture memory. In contrast, decoupled sampling processes all the samples for a triangle together in a single pass and performs memoization dynamically inside the pipeline. In particular, our memoization cache can be small and kept on-chip because it only stores data for the triangles that are in flight. It does not share computed values across frames.

Variable Shading Rates. Yang et al. [2008] apply global level-of-detail per frame in a general real-time rendering system by rendering a subsampled image and using edge-preserving upsampling to the final resolution. Several earlier authors have also described adaptive subsampling and intelligent upsampling in particular use cases. Direct3D 10.1 can dynamically adapt shading frequency by selecting between per pixel or per visibility sample shading in MSAA. Decoupled sampling extends this and adds more flexibility, allowing for shading over arbitrary domains like object space, and for continuous adaptation of the shading rate, where sample correspondences are complex.

3. DECOUPLED SAMPLING

We first describe decoupled sampling in its simplest form, before discussing specific implementations in the context of high-performance parallel pipelines. We define *visibility samples* as the points which are tested against the triangle edges and the Z-buffer, and *shading samples* as the points at which surface color is computed.

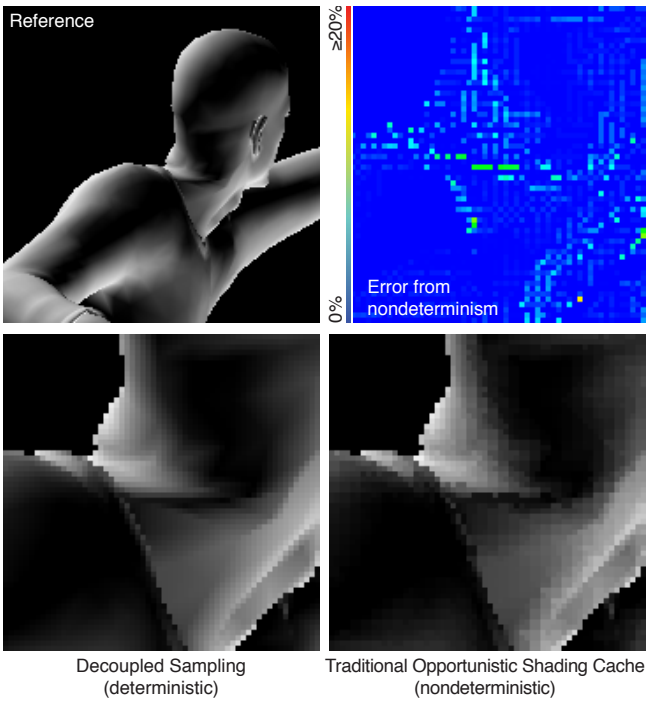

Fig. 3. Decoupled shading vs. nondeterministic caching (Reprojection Caching, Multiview Rasterization). Shading at the exact location of a shading cache miss, as done by nondeterministic caching schemes, introduces visible errors. Decoupled sampling produces high-quality results by computing shading at locations defined by a smooth, deterministic mapping from visibility samples to a well-defined shading grid. Meanwhile, opportunistic caching results in an unpredictable and non-smooth mapping of shading samples into screen space, producing order-dependent shading discontinuities.

3.1 Algorithm

Decoupled sampling builds on standard rasterization-driven graphics pipelines, as shown by the following pseudocode.

```
1  for each triangle
2    setup, compute edge equations
3    for each visibility sample (x, y, u, v, t)
4      if inside triangle AND passes Z // visibility driven
5        // decoupled domain
6        map (x, y, u, v, t) to shading index S
7        if S is in memoization cache
8          color ← cache[S] // reuse
9        else
10         // shade on-demand, at discrete location
11         color ← compute shading at S
12         cache[S] ← color // dynamic cache update
13       framebuffer[x,y,u,v,t] ← color // fully supersampled
```

First, an input triangle is rasterized against visibility samples, which might include extra dimensions for time t and lens uv (line 3–4). We assume an API extended to provide per-vertex motion vectors as well as lens aperture size and focusing distance. Decoupled sampling is mostly agnostic to the details of rasterization, and various strategies can be used, as described in our results.

To achieve decoupling, a visibility sample that passes rasterization and early-Z is mapped into the shading domain of the current triangle (line 6). For example, shading can be performed an expected once per pixel by mapping visibility samples to the same barycentric location at the beginning of the exposure, in the view from the center of the lens, and snapped to the middle of nearest pixel. This provides a many-to-one mapping to a discrete grid, and many visibility samples map to the same shading index. (Section 3.2 discusses the possible mappings.)

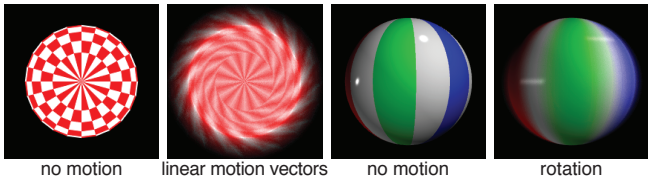

Fig. 4. Simple failure cases for our two key approximations: assuming linear motion over the shutter interval, and shading at a single discrete time and aperture location per frame. Note the dilation and fuzzy edge of the spinning red checkered sphere: with precise motion it should remain round and hard-edged. Similarly, smeared specular highlights on the spinning beachball should remain stationary.

This provides us with a deterministic shading index S, which is tested against a *memoization cache* (line 7). The value for S might have already been requested by another visibility sample mapping to the same index, in which case we can simply reuse it and avoid recomputation (line 8). If it is not present, we trigger a shading request (line 11), and once the value is computed, we update the cache immediately (line 12). When using a fixed-size cache to approximate memoization, we need to free old values to make room for new ones when the output buffer is full. Our implementations free the least recently used item, where time of use is updated each time the entry is looked up. Note that, when there is a miss in the cache, decoupled sampling shades at the discrete shading index S, not at the location of the visibility sample. This gives smooth and deterministic results, by always shading on a single grid defined by the mapping function, rather than shading opportunistically at the visibility sample location which generated the miss.

Finally, the framebuffer at the visibility sample location is updated with the shaded value. Note that the framebuffer is stored at the full supersampled resolution, but might contain the same shaded value multiple times due to reuse in line 8, as in MSAA.

For intuition about reuse, one can consider all the visibility samples that share the same shading value: a given shading value gets smeared across the screen according to motion blur or defocus. This is in stark contrast to MSAA, where a shaded value only affects visibility samples within a single pixel. But decoupled sampling is a *pull* or *gather* process, where visibility samples request shading values. This is in contrast with micropolygon shading, where colors are computed on the surface first, and then *scattered* into all visibility samples they may touch.

3.2 Decoupling Mapping

Our standard decoupling mapping S simply shades once at the beginning (or middle) of the shutter interval, as seen from the center of the lens. For this, the rasterizer provides barycentric coordinates defined on the clip-space triangle for each visibility sample. These clip-space barycentrics are used to compute the corresponding clip-space position on the $t = u = v = 0$ triangle, which is then projected onto the screen. We discretize the obtained image coordinates to the nearest pixel center. Note that, in the absence of motion and defocus blur, this gives the same result as MSAA. This mapping from clip-space barycentrics to shading coordinates is a simple 2D projection that can be represented by a 3×3 matrix. In

this case, simply using pixel coordinates as cache tags, there is ambiguity between primitives overlapping the same pixels. The cache tags must also be extended to include the unique ID of each primitive to unambiguously specify the value on that surface.

Another useful mapping function shades on a parametric grid in object space. This allows object-space shading, but decoupled from the geometry sampling rate, and with reduced overshading due to only shading after precise visibility. This is evaluated by Burns et al. [2010].

Degeneracies. The mapping from barycentrics to shading locations needs to be chosen so as to avoid degeneracies. For instance, if the triangle is behind the eye in the beginning of the time interval, its image as seen from the lens center at $t = 0$ is not a valid shading domain. Similar cases include triangles that degenerate to line segments as seen from the lens center. In such cases we choose an alternate mapping. In practice, we use $t = 1$ if the corresponding triangle is non-degenerate, or directly shade on a regular grid in barycentric space in extreme cases where no single reference view is sufficient, since this space is always well-defined. This can be seen as the 3D/5D equivalent of shading at an alternate sample location if the pixel center is outside the triangle in 2D MSAA. It should be noted that there is no overhead caused by degenerate mapping configurations: the decision is made once per triangle, and does not affect rasterization, only the projective mapping from barycentrics to shading coordinates. While we can give less guarantees on shading rate in these cases, the shading results are still computed correctly. These cases are rare and do not show in our results in any significant way.

Approximations. Like Reyes, our technique assumes that shading does not vary much across the aperture and during the exposure. While this is reasonable for many configurations, it has some limitations. For instance, highly view-dependent reflections on moving objects are not handled correctly; the highlight due to a stationary light on a spinning, glossy ball should stay the same over the whole frame (Fig. 4, right). Similar to MSAA, once-per-pixel shading may also yield artifacts for complex non-linear shaders such as bump-mapped specular objects (Fig. 5). Also, per-vertex motion vectors cannot encode curved motion. This is notable for spinning objects (Fig. 4, left). However, this limitation is orthogonal to decoupled sampling, and it can be addressed either by adding time as a dimension in the decoupling mapping and shading cache indices (so that the shading domain becomes a 3D volume), or like RenderMan by combining decoupled sampling with accumulation buffering.

3.3 Extensions

Bilinear Reconstruction. So far we have defined the cache look-up as a nearest-neighbor filter: visibility samples take the value of the closest corresponding shading location. We also support bilinear reconstruction. If the shading domain is 2D (e.g. pixel grid at $t = 0$ in the center view of the lens), we simply interpolate between the four closest shading locations. Note that this means that a cache miss in line 8 of the pseudocode now triggers the computation of four values.[1] This can be easily generalized to a shading domain of higher dimensions, for instance trilinear look-ups with shading values varying in both space and time. (See the figure in supplementary material for an example.)

Controllable shading rate. The mapping between visibility samples and shading locations can be adjusted by the programmer on a per-primitive or per-shader basis. In particular, it can be desirable to adjust the shading rate according to the spatial complexity of a shader: a matte shader can be sampled only once per pixel, but a shiny surface with high-frequency bump mapping can require shading computation at full supersampled resolution to reduce aliasing (Fig. 5). Decoupled sampling supports controllable shading rate by simply modifying the mapping function S in line 5, allowing smooth and continuous control of shading granularity. For example, varying shading rate with screen-space fragment shading simply requires changing the discretization of the mapping function to vary its quantization frequency.

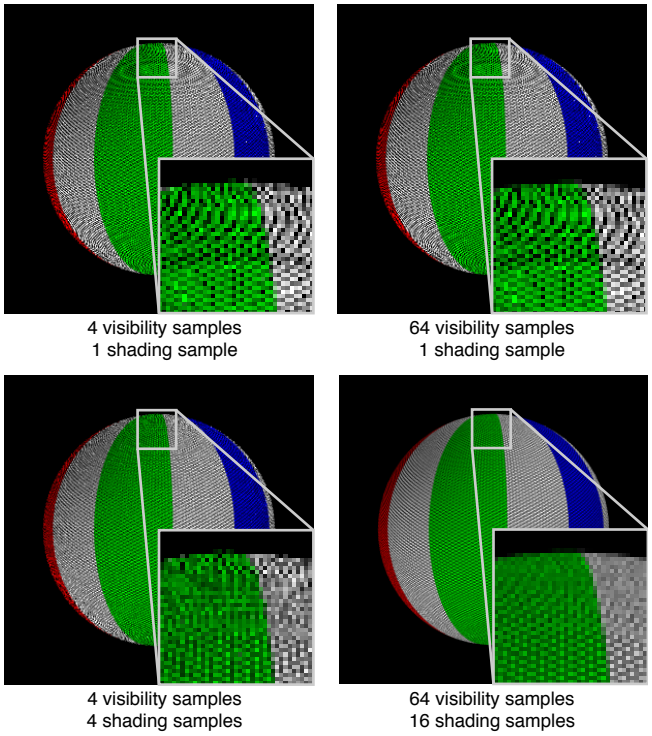

Fig. 5. An illustration of controllable shading rate in the case of an aliasing bump map. ***Top row:*** At only 1 shading sample per pixel, visibility sampling rate has no effect on the aliasing. ***Bottom row:*** Increasing the shading sampling rate alleviates aliasing problems. The shading can be supersampled at its own frequency independent of visibility. The shading rate can be controlled per primitive or per shader.

Adaptive shading rate. Shading rate can also be adapted automatically based on the configuration of the current primitive or object. In the context of defocus blur, we have experimented with a shading rate that depends on defocus blur such that the interval between shading samples is proportional to the circle of confusion. The circle is computed conservatively at the vertices such that the minimum blur dictates the shading rate for the whole primitive. When the whole triangle is out of focus, the shading grid is coarser than the pixel grid. Combined with bilinear reconstruction, this can yield significant savings at little impact on the image. The image in Fig. 6) was computed at little error at an average of 1.3 shader invocations per pixel using adaptive shading grid resolution, compared to the average shading rate of 2.05 when using a fixed 1:1 mapping. Note that alpha-tested primitives cannot be effectively subsampled because the shading affects visibility.

Similar extensions also apply to the Reyes algorithm. The key difference between Reyes and explicit decoupled sampling is that

[1] Note that in practice samples are shaded in quadruplets anyway because of finite difference derivative computations. See Section 5.

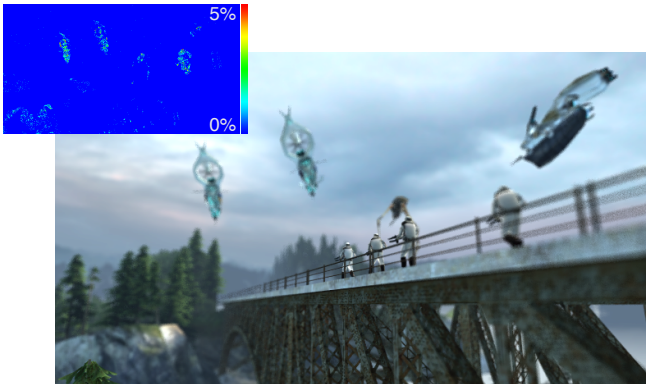

Fig. 6. This image from Half-Life 2 Episode 2 was rendered with 27 visibility samples per pixel using adaptive shader subsampling and bilinear reconstruction from a 256-quad shading cache. The shading rate was an average 1.3 times per pixel. The error plot shows relative image difference to the same frame rendered using non-adaptive 1:1 mapping of screen pixels to shading locations, which had an average shading rate of 2.05 per pixel.

decoupled sampling lazily evaluates shading only as required by visibility, rather than precomputing grids of potentially relevant samples. This is most significant in the case of shading at more than one time during the shutter interval or point on the lens. In these cases, Reyes requires splitting, dicing, and shading all geometry at each shading time/lens sample, while decoupled sampling just lazily fills in the samples in a logical 3D shading grid, which is never directly evaluated.

3.4 Discussion

Decoupled sampling is visibility driven: shading values are computed only when requested by a visible visibility sample. This is in contrast to Reyes, where micropolygons are shaded before rasterization, with only conservative knowledge of visibility.

In contrast to opportunistic schemes that seek to reuse shading across primitives and frames, we reuse shading samples only within a single primitive during a single frame. This makes our approach robust to occlusion and transparency, because other primitives cannot occlude or change cache entries due to blending. While the shading domain might be aligned with the screen, the cache is independent of the framebuffer. In particular, exactly the same shaded values are returned for a given visibility sample irrespective of whether it was cached or recomputed, making the shading entirely deterministic and independent of computation order (Fig. 3). Furthermore, the freshly computed shading values are placed into the cache immediately for fine-grained reuse without having to wait for the next frame.

The mapping used for computing the shading index S for a given x, y, u, v, t tuple is a free parameter in our method, enabling us to super- or subsample shading spatially (Figs. 5, 6), or even easily add more dimensions like time-dependency when necessary.

Shading generally requires derivatives for antialiasing, e.g., mipmapping. We enable this by shading quadruplets of shading samples at a time, similar to current GPUs. We incur similar overshading at boundaries, and also need to "snap" shading locations of samples that fall out of the triangle to ensure valid interpolation.

Because memoized shading results stand in for actual shader execution, any secondary results of the shader invocation must also be retained. The transparency (alpha) values of shading samples are carried along with their color. In addition, pixel and alpha kill/discard results must be memoized and applied to all visibility samples that use the same shading sample. Similarly, all render target outputs must be memoized together.

4. GRAPHICS PIPELINE IMPLEMENTATIONS

Decoupled sampling method was conceived with modern real-time graphics pipelines in mind, meant to allow evolutionary implementations which retain existing optimizations where possible. We explore the implementation of decoupled sampling in two main graphics pipeline architectures: a sort-last fragment pipeline similar to current hardware GPUs, and Larrabee's tile-based sort-middle pipeline [Molnar et al. 1994; Eldridge 2001; Seiler et al. 2008].

Decoupled sampling is implemented by augmenting these pipelines with 1) a space-time-lens rasterizer that computes the barycentrics of all visibility samples, using stochastic sampling for lens and time, 2) decoupling logic that defines the mapping from barycentrics to shading space and the hash to the shading indices, 3) a shading cache for memoization of shading samples, generally stored on-chip and local to the shading system or fragment backend, and 4) cache management logic that manages reference counting and shader completion tracking, as well as the replacement policy.

The primary concerns in real-time graphics systems are parallelism and the memory hierarchy. Multithreading is used to hide long memory latencies and maintain high ALU utilization, but requires coherent behavior, as well as on-chip memory to maintain state (registers) for the many parallel items in flight. Processing order, driven by the synchronization points (sort-middle and sort-last), determines memory access patterns and cacheability. Finally, sorting and distributing parallel items at synchronization points requires memory—either on-chip or off—for buffering. These architectural themes underlie our design considerations, and distinguish the trade-offs and relative cost of different implementations in the two styles of pipeline we studied.

4.1 Efficient Implementation

Costs. We first consider the basic costs of adding decoupled sampling. An efficient real-time hardware implementation needs on-chip memory to cache memoized shading values for quick reuse. In many cases it is possible to reuse existing buffer points for caching, but larger reuse windows might create pressure to make them bigger. In practice, we find caching 1k shaded quad-fragments is highly effective (Sec. 5.3). Motion and defocus blur may also impact the efficiency of color and depth compression, though earlier investigations have shown that some existing compression schemes can still perform reasonably [Akenine-Möller et al. 2007]. In addition, there are modest arithmetic costs for tracking barycentrics for visibility samples, mapping and hashing them to shading samples, and performing cache lookups and management. These per-sample costs are small compared to shading per visibility sample, but can still grow significant as sampling rate increases.

Visibility & Shading Coherence. High performance graphics pipelines drive fragment processing using screen-space 2D block traversals optimized to maintain coherence simultaneously in visibility computation, shading and texture access, and framebuffer updates. By remaining visibility driven, decoupled sampling retains visibility coherence optimizations, but because it decouples shading from visibility sampling, shading coherence is no longer explicitly guaranteed—motion and defocus blur cause the same shading samples to touch many pixels. However, our experiments (Sec. 5.3)

indicate that a standard space-filling 2D traversal is sufficient to maintain coherence in the shading cache, and in particular to avoid pathological behavior. The reuse rate is influenced both by size of the cache, and somewhat by the amount of motion and defocus blur.

While the basic decoupled sampling algorithm logically shades individual shading samples, our implementations compute and cache shading in blocks in shading space to enable coherent SIMD execution, exactly as in a non-decoupled pipeline: the shading engine is not altered, and intra-block coherence is identical. In particular, the shading request batch size can be used for trading off potential overshading for greater shading coherence similar to existing GPUs.

Texture Coherence. While texture coherence is unaltered within shading blocks, the order in which these blocks are issued can be altered by decoupling. However, this effect is local: since triangles are rasterized in order and do not share shading samples, reordering due to blur and shading cache misses may only occur within the shading blocks generated by a single triangle. The modern dynamic thread schedulers that drive shaders (hardware schedulers in traditional GPUs and software fiber switching in Larrabee) are already designed to tolerate large fine-grained variation in texture latency due to unpredictable misses. Our experiments confirm that texture caching and bandwidth are largely unaffected by decoupled sampling (Sec. 5.3).

4.2 Cache Management & Synchronization

To implement our cache in a modern pipeline with multiple asynchronous and parallel stages and multiple triangles in-flight at once, the shading cache indices are augmented with triangle IDs. The resulting unique cache indices are used for tracking the cache entries throughout their lifetime. Shaded colors are not consumed until the back-end of the pipeline, and thus cache entries cannot be freed until all outstanding references to them are fulfilled. Outstanding references are tracked using simple reference counting. This is similar to a conventional sort-last GPU pipeline, except that buffer entries may be referenced by more than one fragment's visibility samples, so their reference count must be more than 1 bit. In addition, in pipelines where shaders can complete out of order and independently from framebuffer processing (like sort-last GPUs), an additional flag is needed to track whether a shading block has completed and its results populated in the cache before it is used by the back-end. Finally, when the output buffer is full, an old value is freed to make room for a new one. Our implementation frees the least recently used non-outstanding item. This is enforced conservatively: when the cache is full with entries that have not been consumed, new shading requests stall.

4.3 Sort-Last Fragment

Modern GPUs primarily use variants of the sort-last fragment architecture (Fig. 7) for synchronization and enforcing in-order execution: shaders execute in parallel, independent from pixel processing, and complete out of order. Strict ordering is enforced using reorder buffers of hundreds or thousands of entries before the raster operation (ROP) units that update the framebuffer. This allows fragments to be kept asynchronous and independent to as late as possible in the pipeline, maximizing available parallelism at the cost of fine-grained global synchronization.

We propose to reuse the existing reorder buffer between the shading and framebuffer stages as an explicit cache (Fig. 7, bottom left). In a standard pipeline, output slots are allocated for every block of shading samples as it is sent to the shader, and they are freed as soon as they are consumed by the framebuffer update. When the reorder buffer is full, earlier stages of the pipeline stall.

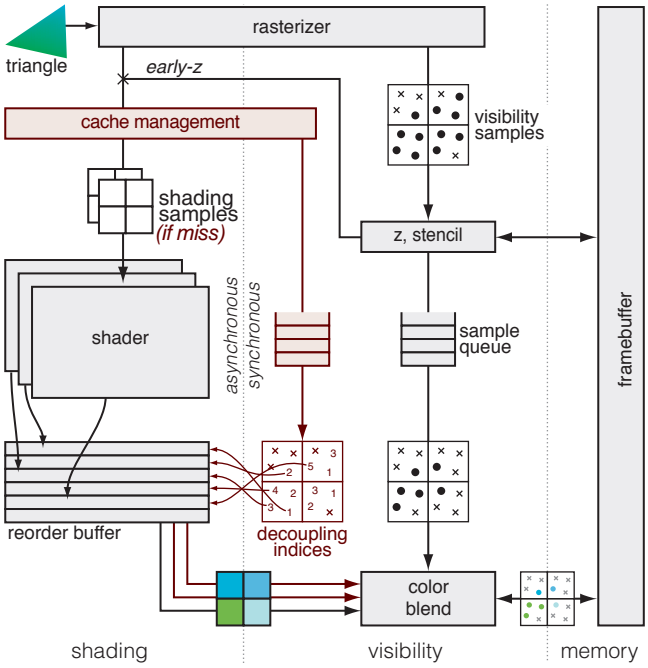

Fig. 7. A modern sort-last GPU fragment pipeline, augmented with decoupled sampling. Modifications required for decoupled sampling are highlighted in red. Decoupled sampling makes the generation of shading samples, and the tracking of their correspondence with visibility samples, explicit rather than implicit. This requires the addition of logic and communication to explicitly track decoupling. But, in a sort-last GPU, the actual storage of memoized shading values can be inexpensive, reusing existing buffers between shading and color blend.

For decoupled sampling, we similarly allocate storage for shaded colors when new shading samples are issued. To leverage reuse, we do *not* free values as soon as they are first consumed, but rather leave them in the buffer to be potentially reused by future visibility samples. When an already-present shading sample is requested, we only emit the corresponding cache indices into the output buffer, and reference count the corresponding buffer entries. This introduces two new architectural pressures. First, higher reuse rates call for extending the lifetime of items, implying an increase in the buffer size. (We study the effect of cache size on shading reuse in Sec. 5.) Second, the reorder buffer may require more fine-grained read ports to support efficient reads from more than one shading sample block per block of visibility samples. In general, though, decoupled sampling is a natural fit in a modern sort-last pipeline, since shading and visibility processing are *already* "decoupled" by asynchronous execution.

We validated our design for synchronization and resource management by building a transaction-level simulator of a modern, parallel GPU fragment pipeline modeled after ATI R6xx and extended with decoupled sampling. The greatest practical architectural pressure is the high framebuffer bandwidth incurred by high supersampling rates required by defocus and motion blur. This issue is orthogonal to decoupled sampling, but we studied it to understand its role in the overall cost of motion and defocus blur (Section 5.3.3).

4.4 Tile-Based Sort-Middle

In a tiling-based rendering architecture, a *front-end* pass transforms primitives, computes their coverage, and enqueues them at the screen tiles they overlap. A separate *back-end* pass processes all pixels in each tile, shading them and blending them into the framebuffer. By front-loading a screen-space sort of the geometry, they globally synchronize over triangles rather than pixels, and thereby dramatically simplify concurrency in fragment and pixel processing. Within a single pixel[2], all processing is synchronous and strictly in order over the lifetime of a rendering pass all the way into the framebuffer. This execution model fundamentally *couples* shading and visibility samples by tying together shading with framebuffer processing, so it does not require synchronization, reordering or explicit inter-stage buffering in the back-end. The trade-off for the simplicity of coarse-grained global synchronization is that triangles may cover multiple tiles, requiring redundant storage and processing. Any per-triangle back-end computation (raster, interpolant construction, etc.) is potentially duplicated, as is tile queue storage (termed *bin spread* by Seiler et al. [2008]), and visibility and shading coherence is lost across tile boundaries because of asynchrony between the tiles.

These design trade-offs are potentially challenging for decoupled sampling. Losing coherence across tiles reduces the effectiveness of shading reuse, and any storage used by the shading cache competes with the framebuffer in the on-chip memory. This is exacerbated by blur, which increases bin spread: higher visibility sampling rates require more storage per pixel, reducing the screen area covered by a tile in a fixed storage size, and motion and defocus blur increase the screen extents of a triangle, and therefore the number of tiles into which it falls. (We study these effects on bin spread in Sec. 5.3.3.) Finally, the coupled execution of shading and framebuffer (visibility) processing is most efficient with a predictable, fixed relationship between shading and visibility samples, where decoupled sampling creates (and exploits) a dynamically variable relationship to reduce shading work.

A first strategy to implement decoupled sampling is to simply extend the back-end so that each visibility sample references a unique shading sample, and prefix the back-end processing of each pixel with a cache lookup (and update in case of a miss). However, the irregular and variable rate relationship between visibility and shading samples causes a significant shading load imbalance because fixed-size chunks of visibility samples require radically different amounts of shading work. This variable rate relationship also causes poor SIMD utilization, because the whole SIMD batch of visibility samples needs to wait for shading completion in the case of even a single shading cache miss. In a SIMD architecture, this is equivalent to overshading due to shading an entire block whenever a single sample misses in the cache.

An alternative is to split the back-end into two separate asynchronous phases, much like the sort-last fragment pipeline: a shading phase that computes shading asynchronously from framebuffer updates, and a ROP phase that performs in-order framebuffer updates. This better mirrors the two natural computation domains of decoupled sampling. A basic sort-middle tiling is still performed first, and cores still perform back-end processing in a single pass. Within that pass, however, worker threads that previously only performed coupled shading-ROP tasks now dynamically schedule between shading and framebuffer updates, in effect introducing a per-tile sort-last stage to the pipeline. Just like in a sort-last implementation, buffering is used to queue ROP work which is blocked on shading results, and to reorder shading samples as they complete. This architecture is more amenable to decoupling, and avoids effective overshade due to load imbalance between pixels in a tile, but at the cost of scheduling and buffering for two separate back-end stages rather than one. In general, any implementation of decoupled sampling adds more relative memory overhead to a pipeline with synchronous fragment shading and pixel processing than one which already buffers between the two, like sort-last pipelines.

[2]"Pixels" here is a simplification: synchronization is actually over framebuffer *samples*—output locations—so neighboring triangles covering adjacent but non-overlapping samples within a pixel may proceed concurrently, since they do not affect the same parts of the framebuffer.

4.5 Discussion

A sort-last implementation of decoupled sampling naturally allows for a single, global shading cache. This leads to maximal shading reuse for a given raster traversal order and shading cache size, because coherence is not lost due to tiling. Such maximal reuse is achieved at the cost of having to support concurrent access from large numbers of threads through many fine-grained cache read ports. However, as shader outputs need to be buffered and globally reordered even without decoupling, other architectural changes are small. Because memoization is implemented by allowing items to remain buffered past their first use, the only effect of caching is extending the lifetime of entries past when they are first completed and consumed by the color buffer. In this context, the effective cache size overhead relative to a non-decoupled pipeline is those items that would have been retired after first reference but are kept for re-referencing. If the framebuffer processing is strictly in order, this overhead is limited by the size of the triangle currently being drained through the ROPs, even if many more are in-flight after it, because their shading samples could not have been freed from the buffer even without decoupling.

A sort-middle architecture presents a range of alternatives for implementing the shading cache. While local, per-tile caches are the simplest, it would also be possible to implement a global shading cache accessed by all tiles concurrently. This is at odds with the sort-middle design goal of reducing global synchronization, and the lack of coherence across tiles—the parts of the same triangle that fall into neighboring tiles may be processed far apart in time—makes this approach less attractive. However, this trade-off could be different for a hardware-scheduled sort-middle pipeline, which can more easily afford fine-grained global synchronization than a software pipeline like Larrabee. Still, the long-term trends towards ever-greater parallelism will always favor less synchronization.

In our evaluation (Sec. 5), we focus on comparing a sort-last architecture with a global cache to a sort-middle architecture in the limit, with purely local caches and no inter-tile communication.

5. IMPLEMENTATION AND RESULTS

To study decoupled sampling, including API integration and other details and interactions, we implemented it in a complete Direct3D 9 functional simulator, extended with APIs to control blur and shading rate[3]. We evaluate decoupled sampling on actual Direct3D ap-

[3]Decoupled sampling extends trivially to newer versions of Direct3D with one exception: unordered access views in pixel shaders in Direct3D 11 fundamentally break the pure-functional nature of shading. Side-effecting shaders cannot be evaluated a variable number of times without introducing nondeterminism. Unordered access views are already defined to provide virtually no determinism guarantees, however. Integrating them with decoupled sampling suggests two potential routes beyond simply leaving behavior

plications running real shaders, including Half-Life 2, Episode 2 and Team Fortress 2, extended to include motion and defocus blur. We study the architectural implications (cache and shading coherence, framebuffer bandwidth, etc.) on these scenes and describe results in Section 5.3.

5.1 Rasterizer Implementation

The rasterizer enumerates the visibility samples that a given input triangle hits, and computes barycentric coordinates, which are subsequently mapped into shading indices. While we here describe a particular implementation, the choice of rasterization algorithm is orthogonal to the rest of the decoupling pipeline.

Sampling Patterns. To enable motion and defocus blur, we rasterize triangles that vary over time and with samples that cover the lens aperture. The samples in our framebuffer have five dimensions: two for the subpixel x and y coordinates, and in addition, u, v for lens location and t for time. Like current GPUs, we employ the same x, y sampling pattern for all pixels, but employ stochastic rasterization and sample lens and time using 3D jittered grid sampling patterns that repeat every 32×32 pixels. (Accumulation buffering shares the lens and time pattern across all pixels, which results in highly disturbing aliasing artifacts—see Figure 8, left).

Hit Testing and Edge Functions. We dynamically classify each triangle into moving/non-moving and blurring/non-blurring due to defocus, and handle each of the four cases by a specialized algorithm. Stationary, non-defocus-blurred triangles are rasterized using an unchanged 2D hierarchical rasterizer. For triangles that move, but do not exhibit defocus blur, we employ time-continuous 3D homogeneous edge functions as proposed by Akenine-Möller et al. [2007]. For handling stationary triangles that undergo defocus blur, we introduce novel aperture-continuous 4D edge functions. Analogously to time-continuous edge functions, they simplify the per-sample visibility test by pushing common computations out into the triangle setup. See Appendix A for details. In the fully general case of both motion and defocus blurred triangles, we compute the warped triangle that corresponds to the u, v, t of the sample and test for a hit directly, because this turns out to be more efficient than using 5D time-aperture-continuous edge functions.

Scanning Order. We rasterize the triangles by looping over pixels in the 2D screen bounding box of the triangle, computed as the union of all possible time/lens projections. The screen is scanned in tiles of 32×32 pixels, which are processed in a Z curve order. Inside the tiles, individual quadruplets of pixels are also rasterized along a Z curve. To avoid the cost of testing each sample in the entire bounding rectangle, we utilize sub-rectangles defined by the strata of our sampling pattern in a way similar to Pixar's RenderMan implementation, and also analyzed by Fatahalian et al. [2009] for micropolygons.

Motion and defocus blur can alter the efficiency of different sample layouts and raster stamp traversal orders. We follow traditional multisampling GPUs and densely store all subpixel samples for a given pixel together in the same block. This corresponds to blocks which are narrow in x, y but span the full u, v, t range. Motion and defocus blur may also impact the efficiency of color and depth compression. While earlier investigations have shown that some existing compression schemes may still perform reasonably [Akenine-Möller et al. 2007], we view further evaluation of both memory layouts and compression as orthogonal to decoupled sampling and leave it as future work. All of our results assume no framebuffer compression.

undefined: only evaluating the unordered access portion of the shader to run once per unique shading sample, regardless of the number of times it may be recomputed under cache pressure, or only allowing unordered access writes at per-visibility sample frequency, which is not influenced by decoupling.

5.2 Qualitative Results

To compare image fidelity to accumulation buffering and full stochastic supersampling with varying sampling rates, we present results for an application featuring local deformable precomputed radiance transfer (Fig. 8) [Sloan et al. 2005]. Decoupled sampling (center) is compared to accumulation buffering (left) and stochastic supersampling—both of which shade every visibility sample—and observe shading quality similar to full stochastic supersampling at only a fraction of the shading cost. The supplementary material contains a video showing the scene in motion, including motion blur and a focus rack, rendered using different numbers of visibility samples and the comparison algorithms.

5.3 Evaluation

We studied the effects of the additional computational and memory requirements related to decoupled sampling, supersampled visibility, and motion and defocus blur by instrumenting our Direct3D 9 functional simulator. We gathered statistics on shading, texture, and framebuffer coherence as a function of sampling rate, amount of defocus/motion blur, and shading cache size for both tiled and non-tiled architectures. The amount of blur or *blur area* is reported as the average number of pixels a single shading sample touches. This is equivalent to the average circle of confusion for scenes with just defocus blur, and the length of the motion trails for pure motion blur (assuming a 1-pixel wide path). The measurements are shown in Figure 10 and analyzed below.

5.3.1 *Shading Coherence.* Figure 10 analyzes the effectiveness of the shading cache in capturing reuse for different sizes of cache and amounts of defocus and motion blur for two test scenes for both global (sort-last) and tiled (sort-middle) implementations of decoupled sampling. The Half-Life 2, Episode 2 scene (left column) exhibits defocus blur and the Team Fortress 2 scene (right column) exhibits motion blur due to camera ego-motion. Figure 9 similarly analyzes shading cache coherence over 50 frames of animation with simultaneous defocus and object motion blur for the scene in Figure 8.

With a global (sort-last) shading cache, shading rate stays nearly constant as a function of blur area and sampling rate, varying over at most a 3x range from 8 samples/pixel with no blur to the most incoherent blur at 64 samples/pixel in the same scene (Fig. 10, right). This is the goal of decoupled sampling, and it is extremely successful. The game frames do exhibit some variation in shading rate as a function of blur, likely because they both include large, heavily blurred background polygons which are much larger than any reasonable shading cache. The animated bats (Fig. 9 and accompanying video), meanwhile, contain only small triangles and shading rate exhibits virtually no variation (less than 20%) over an even wider variation in blur area.

In a sort-middle implementation, local shading caches per tile reduce potential shading reuse, because shading is not shared across tile boundaries. Our simulation assumes no sharing between screen-space tiles, which represents the worst-case behavior of the most extreme tiled renderer design. Indeed, the shading rate plateaus at a rather low reuse rate due to lost shading coherence. With tiling, shading rate grows much more as a function of blur, since blur increases bin spread. The slope of shading rate growth

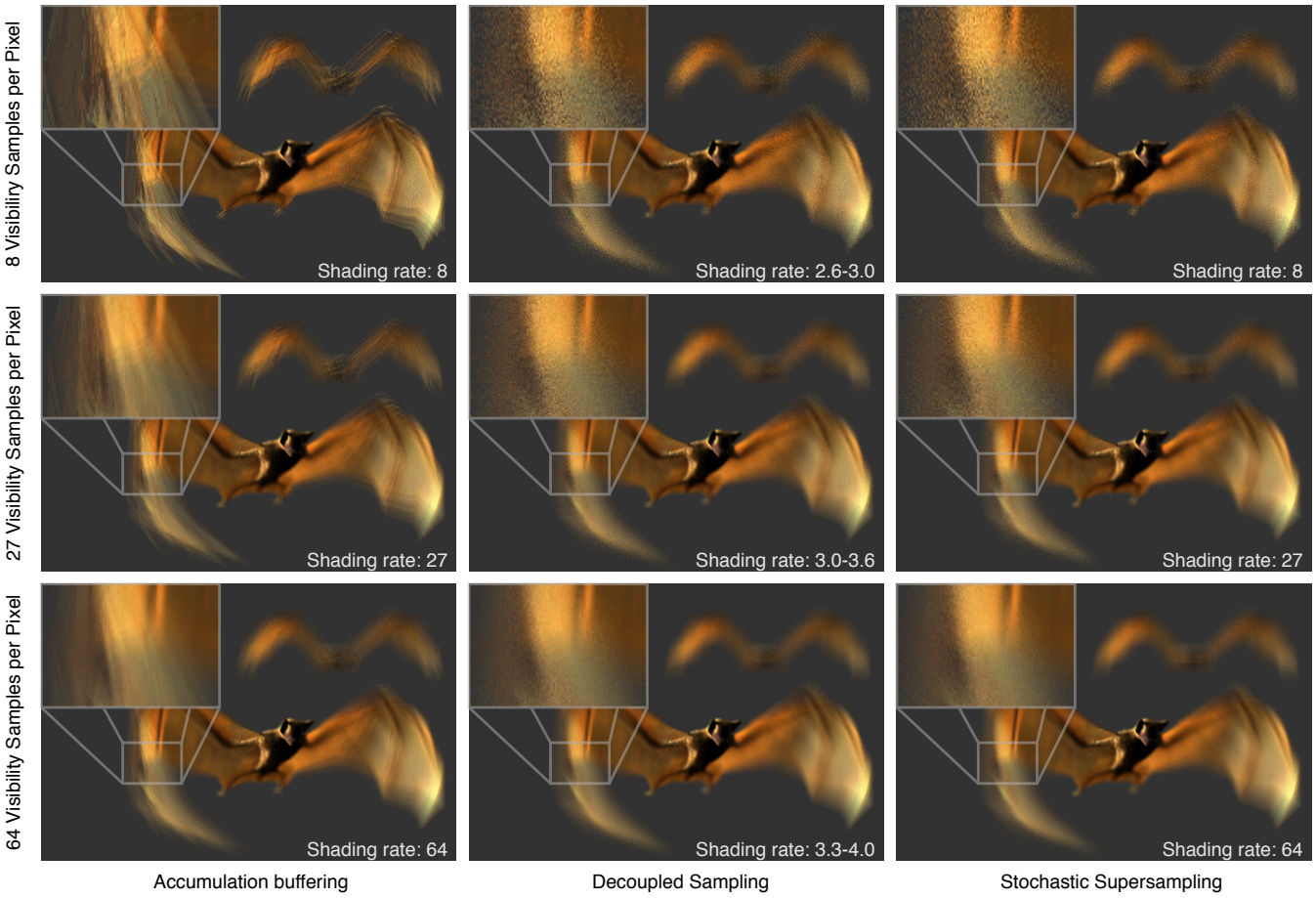

Fig. 8. A comparison of accumulation buffering, decoupled sampling, and stochastic supersampling. **Top to bottom:** 8/27/64 samples/pixel. Decoupled sampling **(center)** attains image quality similar to full stochastic supersampling **(right)** with only a small fraction of the shading cost. Accumulation buffering **(left)** shows significant banding and strobing artifacts, despite shading just as much as stochastic supersampling, but provides a useful comparison, as it is the only technique achievable directly on current hardware.

as a function of blur is steeper at higher sampling rates, since they reduce effective tile size. However, even losing substantial shading reuse due to tiling, decoupled sampling still shades significantly fewer samples than supersampling. To emphasize this, the second row of graphs shows the same data, but graphed as *savings* (higher is better) compared to an idealized supersampling implementation which shades exactly once per visibility sample. Even at high levels of blur, though less effective than a global cache, the decoupled tiled renderer still shades 2–12x less than supersampling in the game frames, with reasonably sized caches. The bats animation, with even higher blur and substantial overshade on small polygon boundaries (giving an ideal shading rate above 3), however, struggles to shade much less than an idealized supersampling engine with tiled caches, while a global cache still performs nearly perfectly.

Motion blur (Fig. 10, right) leads to greater shading incoherence than defocus blur (Fig. 10, left) per unit of *blur area*. This is caused by interaction with the 2D space-filling rasterization traversal order: defocus blur is confined to a compact area on this space-filling traversal, while motion blur causes longer streaks, spanning a wider range of the traversal for the same number of pixels touched. Dependence on the raster traversal order also implies that motion direction may affect shading coherence. We ran tests with synthetic (uniform) motion blur to study the effect of motion direction, and observed a $\pm 20\%$ variation over a 360° rotation, suggesting that a standard space-filling order effectively mitigates pathological behavior.

The last row of graphs in Figure 10 depicts the savings obtained as a function of cache size, at a fixed, moderate level of blur (corresponding to the images rendered elsewhere throughout the paper, and indicated by the vertical lines in the two graphs above), in comparison to an idealized supersampling renderer that shades exactly once per visibility sample and does not suffer from overshading at triangle boundaries. Decoupled sampling with quad-granularity shading blocks begins to outperform idealized supersampling (shading exactly once per visibility sample) when the cache size reaches 128 shading samples. Increasing cache size continues to improve shading performance, especially for scenes with blurry, large triangles (the game frames), up to thousands of samples for a global cache. A 4k-entry (16 kB, for 4 bytes/sample color) global cache provides large benefits (4–35x for the test scenes) for a cost that seems feasible at the time of writing. Conservative per-

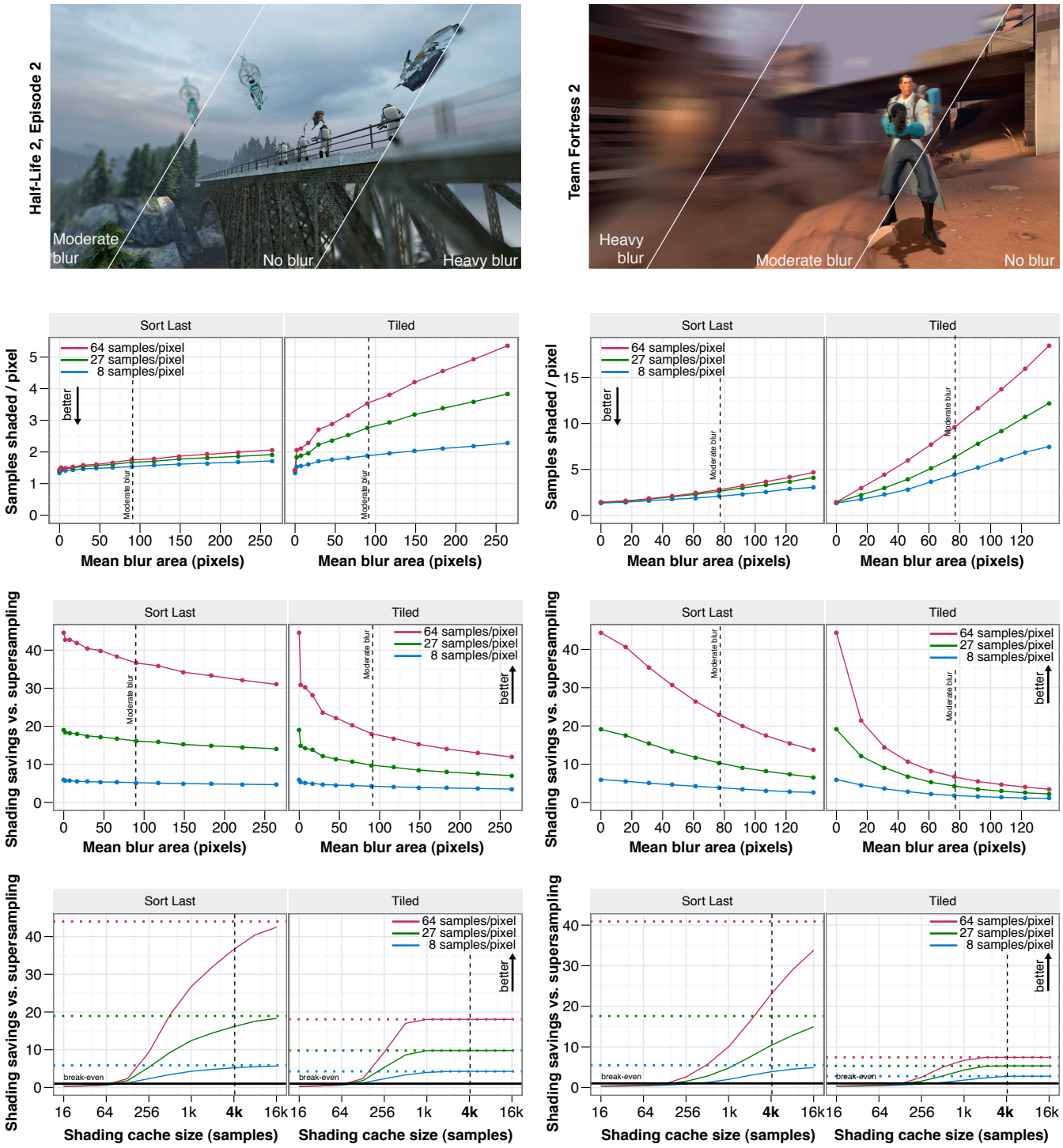

Fig. 10. Shading work for global sort-last and purely local tiled implementations of decoupled sampling, as a function of blur and shading cache size, at 8, 27 and 64 visibility samples per pixel. Shading rate (row 1) is the average number of shader invocations per pixel of coverage (lower is better). Shading savings gives the factor of reduction in shading work relative to an idealized supersampling implementation which shaded exactly one sample per visibility sample. The dotted lines in the shading cache size graphs (row 3) show the ideal shading savings using an infinite-sized cache. The top 2 rows use a 4k-sample shading buffer. The amount of blur is reported as the average area (in pixels) touched by a single shading sample. The horizontal axis ranges from no blur on the left to severe blur on the right (illustrated in the top row images). The moderate blur setting corresponds to all renderings seen elsewhere in the paper.

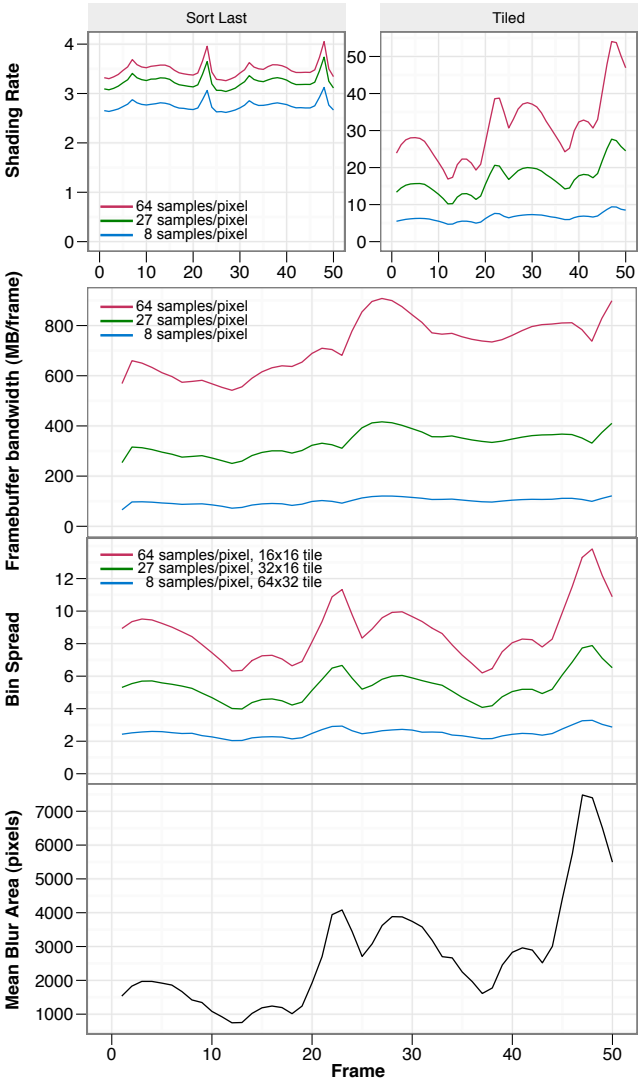

Fig. 9. Shading rate, framebuffer bandwidth, bin spread, and blur area over 50 frames of animation of the scene in Fig. 8, with motion and defocus blur, at 8, 27, and 64 samples per pixel. The animation cycle includes flapping wings and racks focus from the front to the rear bat with a large aperture. Sort-last shading rate is limited by overshading fragments on the edges of very small triangles, and is minimally influenced by blur. Blur area grows large enough that a tiled shading cache is insufficient to extract much reuse. Framebuffer bandwidth scales over a 50% range as a function of scene blur. Bin spread grows with large blur, especially with the small tiles required at high sampling rates. Blur peaks at the end of the sequence when the foreground bat is maximally out of focus. Blur area appears large, but dominated by defocus, which is an area quantity; it corresponds to a mean blur *diameter* of 30-90 pixels.

tile cache performance plateaus at a substantially lower cache size and shading savings than the global cache.

5.3.2 *Texture Coherence.* While we maintain SIMD ALU coherence in shaders by shading multiple quadruplets of samples at a time like a normal graphics pipeline, texture locality between blocks might be affected by the on-demand ordering of shader invocations and the introduction of blur. We studied this in our Direct3D 9 pipeline with a 32kB 16-way set associative texture cache with 64-byte lines, closely modeled after actual hardware. In the Half-Life and Team Fortress frames, we observed per-texel hit rates around 99.6% for non-blurred MSAA images at 8 samples/pixel, and hit rates of over 99.7% for the defocus and motion blurred images using decoupled sampling for all visibility sampling rates. The hit rate is slightly higher with decoupling because a modestly increased shading rate generates more references to roughly the same texels. Despite the higher hit rates, texture bandwidth requirements are slightly larger than for the pinhole image (an additional 5–15%, depending on the sampling rate) because more total samples are shaded. Hit rates only deteriorate for $\ll$ 8kB caches, and are no worse than for single sampling. We conclude that the texture cache effects of decoupled ordering are negligible, and that texture cache performance is unaltered by decoupled sampling.

5.3.3 *Visibility Coherence.* To build a more holistic picture of the broader architectural impact of motion and defocus blur, we investigated the changes in visibility bandwidth—framebuffer bandwidth and ROP cache coherence for sort-last, and binning data for sort-middle—caused by the blur. (Decoupled sampling is in itself orthogonal to the question, as it only supplies the *values* read from and written to the framebuffer.)

Bin Spread (Sort-Middle) As discussed in Section 4.4, blur and

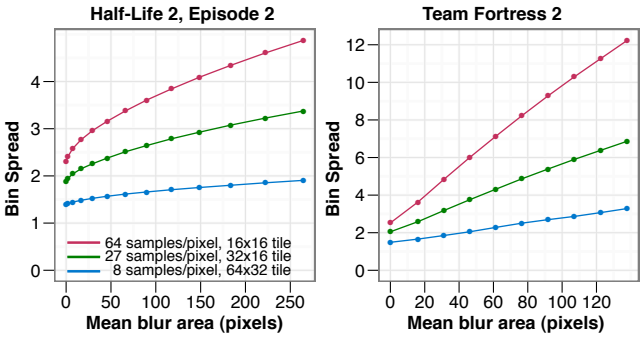

Fig. 11. Bin spread for a tiled architecture as a function of mean blur size and visibility sampling rate. Large blurs increase the number of tiles touched by a single primitive, while higher sampling rates reduce the x, y tile dimensions which fit in a given fixed-size memory. (This simulation assumes tiles are bounded to at most 128kB.)

higher visibility sampling rates affect "bin spread," the number of screen tiles a primitive touches, in several ways. We studied the effects on our game data assuming tile memories of at most 128 kB. This means tiles get smaller as sampling rates increase, and we expect steeper slopes for bin spread as a function of blur for higher sampling rates. Figures 11 & 9 show bin spread as a function of defocus and motion blur. Defocus blur (HL2, left) causes less dramatic bin spread increases than linear motion blur (TF2, right). This is again explained by the fact that linear motion paths of similar total screen area are much longer than the lens blur. We conclude that added memory pressure caused by high sampling rates, combined with the effective increase in triangle size caused by blur, make strong motion blur difficult to achieve without an increase in tile memory size. However, moderate amounts of defocus blur seem achievable with an acceptable performance penalty, especially at lower sampling rates.

Framebuffer Bandwidth (Sort-Last) Sort-last pipelines stream over the framebuffer in primitive order. In this context, the main function of a ROP cache is not to allow reuse between primitives

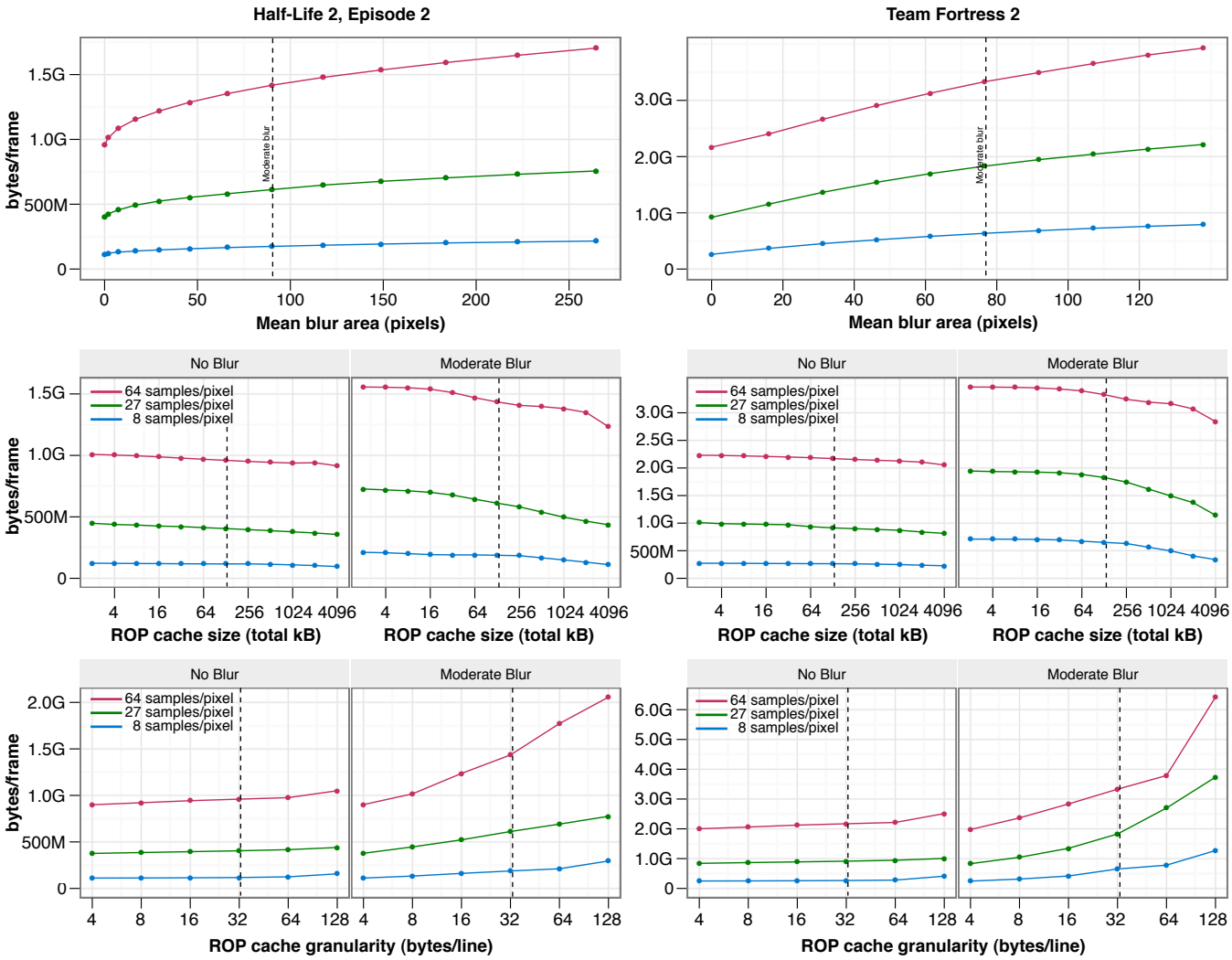

Fig. 12. Framebuffer bandwidth usage in a sort-last implementation of decoupled sampling for two game frames (Half-Life 2, Episode 2, left, and Team Fortress 2, right) as a function of the amount of blur and ROP cache size. Top row: total framebuffer bandwidth as a function of blur area, using 64kB each color and depth-stencil caches with 32-byte lines. Middle row: Framebuffer bandwidth in a moderately blurred frame as a function of ROP cache size, using a 32-byte line. Bottom row: Framebuffer bandwidth in a moderately blurred frame as a function of ROP cache granularity (line size), using 128kB total cache. Note that, at line at a line size of one sample (4 bytes), the blurred renders incur identical bandwidth to the corresponding non-blurred renders.

updating the same samples, but to coalesce reads and writes for a number of nearby covered samples into larger block transactions. The key question, then, is what impact blur has on the ability to coalesce streaming framebuffer accesses.

We studied ROP cache performance in our three test scenes by measuring the amount of off-chip framebuffer bandwidth in three scenarios: 1) as a function of blurriness with a fixed-size cache, 2) as a function of ROP cache size in a moderately blurred frame, and 3) as a function of cache line size with a moderately blurred image and a fixed-size cache. In our simulation, the framebuffer is laid out along the same Z curve used for rasterizer traversal (Sec. 5.1). The results are presented in Figure 12 & 9. As expected, for a fixed cache and line size (of 128kB and 32 bytes, respectively), we observe increases in bandwidth requirements with larger defocus and motion blurs. However, the increases remain within 50% over MSAA (no blur) even in the case of severe defocus and motion blurs: In short, coalescing still works with motion and defocus blur. Blur does not increase the total number of samples being tested and updated, it simply spreads them out in space. Neighboring regions of the same surface, however, are likely to sparsely fill in other samples within roughly the same space at roughly the same time, and traditional caches are sufficient to capture this aggregate coherence in framebuffer access, both with and without blur. Increasing cache size corresponds to enlarging the window over which samples may be coalesced, while reducing line size corresponds to allowing finer-grained transactions with memory. In the limit of a single-sample line size, blurred and non-blurred versions of the frame have effectively identical framebuffer bandwidth. In our results, reducing line size is the stronger lever than increasing cache size, but can be more difficult to apply in practice. We expect that a blur-oriented architecture would drive towards finer-grained lines, and use a 32-byte line in our simulations. This is at least as large

as the native atom of current memory technology, but smaller than non-blur-oriented ROP cache designs would likely choose.

We conclude from our results that, while the bandwidth cost of blurry rendering are nontrivial at high sampling rates due to the large data size of a highly sampled and uncompressed framebuffer, the increase in our tests of at most 50–60% due to blur is a modest additional cost for such effects, and within the capabilities of near-future memory systems. For instance, 27 visibility samples per pixel deliver good quality and require 750–1200 MB of bandwidth per frame in our example scenes.

5.3.4 *Decoupling (Mapping to shading grid).* To access the shading cache, the rasterizer computes shading coordinates for each visible sample by a 2D projective mapping. Its arithmetic cost is the same as evaluating one 2D perspective-correct interpolant—much less than supersampled shading—but this still must be done for each visible sample. As visibility sampling rate grows, this cost becomes nontrivial: in our simulations, it makes up 19–38% of the total frame cost because shading cost is reduced so much (Sec. 5.4). Like many features, decoupling should only be enabled when needed: when blur, adaptive shading, or other applications mean it would provide a significant savings in shading work.

5.3.5 *Bilinear Reconstruction.* All images/video, except Fig. 6, where shading is subsampled in parts of the screen, use nearest neighbor reconstruction from the shading cache. When not using adaptive shading rates, we find the visual difference between bilinear and nearest neighbor reconstruction negligible. We re-simulated the Half-Life 2 frame at moderate blur with bilinear reconstruction and the shading rate increased by at most 20% (due to overshade where bilinear samples required more than 1 quad), with slightly less overhead for higher sampling rates. Texture cache miss rate increased modestly but measurably (per-texel hit rate dropped from 99.6% to 99.5%).

5.4 End-to-End Performance Estimates

While it is challenging to directly predict the absolute performance of a hardware architecture that has not been built, we believe we can offer a reasonable estimate of relative cost and an intuition for overall performance of an actual implementation by using the data gathered by our instrumented functional simulator. We measure the *changes in arithmetic work and off-chip bandwidth* caused by motion blur, defocus blur, and decoupled sampling, in comparison to single-sampled rendering. We determine a baseline breakdown between the relative costs of pipeline stages for a single-sampled render using the simulation framework and methodology described by Seiler et al. [2008]. It simulates the full functionality of the pipeline, attributing cost to categories of operations based on separate cycle-accurate simulation, and adds delays based on detailed simulation of architecture components (e.g. caches) during functional execution of the specific frame. This gives a moderately precise cost of rendering each scene, per-sample-shaded and per-visibility sample rasterized and blended, at a 1x sampling rate, on a specific hardware architecture (Larrabee). This model is accurate for sort-middle architectures. We derive an approximate breakdown for sort-last architectures from the same data by ignoring tiling-related costs, and deriving framebuffer bandwidth from our own ROP cache simulation (Fig. 12). We then model the effect of decoupled sampling by extrapolating per-sample costs based on the exact numbers of samples shaded and rasterized, as well as bin spread and bandwidth, simulated in our Direct3D 9 pipeline for decoupled sampling with motion and defocus blur. While the absolute performance estimated may not be precise, this provides a first-order picture of the relative cost for different pipeline stages, and the pipeline as a whole, at different sampling rates.

To derive the results, we make the following assumptions and additions to the baseline single-sampled model. First, we assume that the vertex stage does not scale. Second, we add a decoupling mapping cost for the decoupled sampling pipelines based on the arithmetic cost on the same units used for shading. Third, we assume that rasterization cost scales linearly in visibility sampling rate. A stochastic rasterizer optimized comparably to a commercial 2D rasterizer is an open research problem and is orthogonal to decoupled sampling; our performance model corresponds roughly to an accumulation buffer rasterization strategy where a single aperture-time sampling pattern is used for all pixels, and a complete 2D setup is performed for each unique sample. This is more pessimistic than MSAA, which is generally sub-linear in total rasterization cost, but we expect blurry visibility to be less coherent. Fourth, we scale fragment shading based on the number of samples shaded, which we have precisely simulated for all configurations. For the supersampled renderer, this scales with the number of visibility samples. Finally, we use our simulated framebuffer bandwidth figures for framebuffer traffic on the sort-last architecture, and extrapolated bin bandwidth by the change in simulated bin spread on sort-middle.

We simulate the performance of our two game frames at 1280×720 resolution with tiled and global sort-last decoupled sampling at 8, 27, and 64 samples per pixel, and compare to an idealized supersampling renderer which shades exactly one sample per visibility sample. Figure 13 presents these results broken down by major pipeline stage, as well as simulated single-sampled and 4x MSAA for comparison.

As visibility sampling scales, vertex processing drops from a modest to a completely insignificant portion of total rendering cost. (It is small enough to be invisible in these graphs.) Computational costs for supersampling scale in direct proportion to the single-sampled render, with the exception of vertex processing. Rasterization (red) and framebuffer processing (orange), by contrast, grow from a modest share of single-sampled rendering cost to a much larger portion as sampling rates increase when decoupled sampling is enabled. (By comparison, 4x MSAA only doubles rasterization cost relative to single sampling in both frames.)

The key result is that decoupled sampling radically reduces shading cost (aqua) for these scenes, keeping it nearly constant with a global sort-last cache, and not constant but still substantially reduced relative to supersampling with a tiled cache. Consequently, decoupled sampling outperforms stochastic supersampling in total rendering cost by large multiples as visibility sampling rates increase. To achieve this win in shading cost, decoupled sampling pays a per-visibility sample cost to compute the mapping to shading samples (purple). While small per sample, this cost scales with sampling rate, and so becomes large as a fraction of overall cost at high sampling rates. At any sampling rate, however, it still remains far lower than the cost of shading per-visibility sample for any but the most trivial shaders. In summary, decoupled sampling reduces the overall computational cost of rendering these real game frames extended with 64-sample motion and defocus blur by approximately 2–4x compared to full supersampling, and shading cost by 3–6x (factoring the cost of decoupling into shading), with both sort-last and sort-middle rendering architectures.

Discussion. The shift in end-to-end computation balance from shading towards visibility-related costs as sampling rates increase suggests reevaluating the trend away from fixed-function visibility hardware: implementing rasterization and decoupling arithmetic as fixed-function units will likely significantly reduce their share of

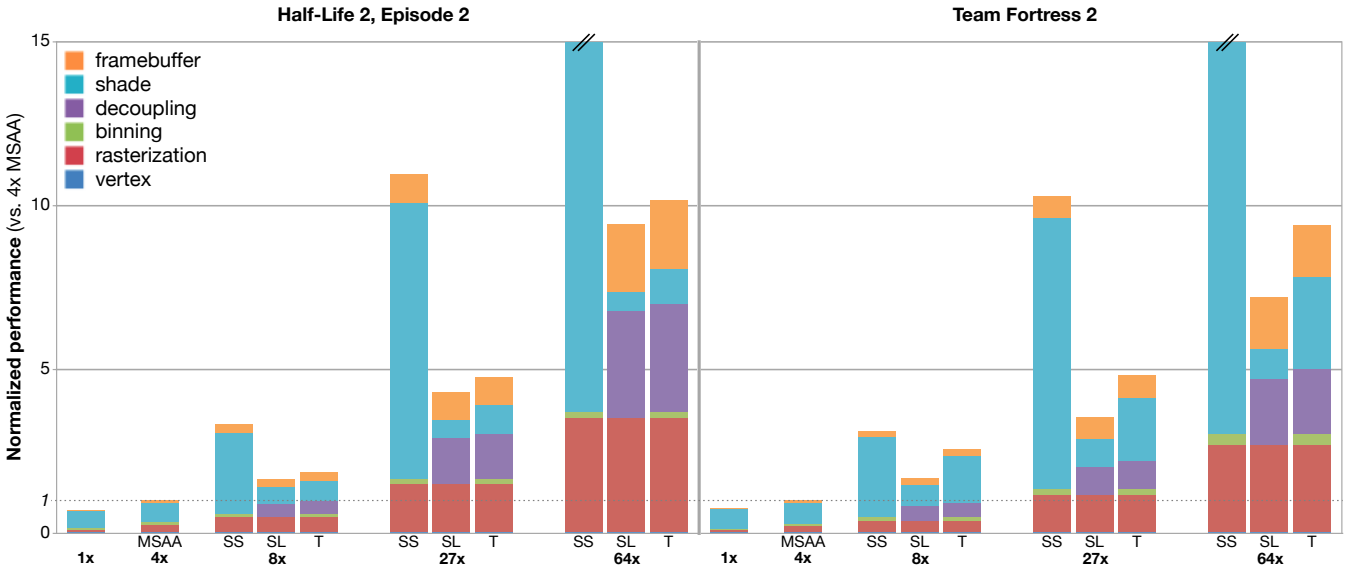

Fig. 13. Estimated relative end-to-end costs for the Half-Life 2 (left graph) and Team Fortress 2 (right graph) frames, measured in relation to a 4x MSAA rendering without motion or defocus blur. The bars show estimated relative costs of pipeline stages for sort-last decoupled sampling (SL), tiled decoupled sampling with per-tile caches (T) and supersampling (SS) for 8, 27 and 64 visibility samples per pixel. Also shown is a single-sampled rendering.

the total cost. Furthermore, these games have relatively inexpensive shading; the win of decoupling (and relative cost of shading in both sort-last and tiled implementations) would grow rapidly with the more expensive shaders of the most cutting-edge titles. The trend to more expensive shading is only expected to increase in the future.

Though we focus on relative end-to-end performance scaling, rather than absolute frames per second deliverable on a specific architecture, the total computational costs, in terms of the Larrabee model used as a baseline, are sufficient for all decoupled sampling tests to achieve 60Hz performance on these scenes within current-generation hardware resources. The required bandwidth is also achievable, although much higher than current compressed MSAA rendering, for all but 64x sampling, which is near the limits of current hardware. Hierarchical Z and compression would improve this, but such high sampling rates are fundamentally bandwidth-intensive. We conclude that a real-time implementation is many times faster than full supersampling, and likely feasible to implement.

6. CONCLUSIONS

Decoupled sampling enables the rendering of defocus and motion blur at reduced shading costs. This is achieved by decoupling shading from visibility sampling. Our caching approach can be seen as wholly deterministic function memoization, in contrast to schemes that opportunistically approximate using previous frames or views. Our results demonstrate that micropolygon shading, as in RenderMan's Reyes architecture and the Razor ray tracer [Stoll et al. 2006], is only one choice in the design space of architectures to decouple shading from visibility sampling. Decoupled sampling has the advantage over micropolygon shading that it only computes shading after precise visibility, reducing overshade in complex scenes with opaque surfaces. (Though micropolygon renderers perform conservative culling before shading, studies of real scenes have shown significant over-shading of non-visible points in RenderMan [Ragan-Kelley et al. 2007].) Decoupled sampling also allows decoupled shading and visibility sampling with larger-than-sub-pixel polygons, saving geometry processing overhead where possible. Finally, our approach imposes only limited alteration of current sort-last architectures, mostly the addition of a cache management unit. In all, it makes real-time rendering of motion and defocus blur feasible with modest extensions to current graphics pipelines.

Decoupled sampling is also feasible for sort-middle pipelines, but it requires a larger change to the basic sort-middle model to be efficient, and still struggles to achieve high shading efficiency in some situations due to loss of coherence across tile boundaries. More generally, blur and high sampling rates are fundamentally challenging for tiling architectures. While we have studied a local, per-tile cache, a global cache has potential for much higher wins, provided it is large enough to alleviate the effects of poor coherence between tiles. Spilling cache entries off-chip could keep local memory pressure low, while allowing a cache large enough to push a tiled renderer back towards the shading rates achievable by the global cache of a sort-last architecture. For expensive shaders, a per-sample gather even from off-chip may still be less expensive than full recomputation of shading. In addition, optimizing the scheduling order between tiles might regain some lost visibility coherence. One could potentially even choose to interleave tile processing to trade the framebuffer bandwidth of spilling and reloading tile memory for greater shading coherence. Studying the trade-off of global vs. local synchronization for caching is natural future work.

Because decoupled sampling fundamentally supersamples visibility while sampling shading at a dramatically lower rate, the load balance among stages changes with decoupling. This is fine for a software renderer, but in a hardware pipeline could require over-provisioning hard-wired visibility resources. Conversely, finer-grained synchronization allows better shading reuse, and fine-

grained synchronization is much cheaper in special-purpose hardware.

Stochastic motion and defocus blur make visibility more expensive both in terms of rasterization efficiency and framebuffer storage. We found that framebuffer caches are still effective, but blur creates pressure to consider finer-grained line sizes and perhaps larger caches. While we have pursued rasterization to some degree, this area is an interesting avenue of future work. In particular, a space-filling pattern is well suited to defocus, but is less ideal for strong motion, suggesting that traversing along motion lines should be investigated. This could improve shading reuse under large blur. In addition, framebuffer compression requires further study.

Like with MSAA, multi-pass deferred shading is challenging for decoupled shading and visibility rates since it traditionally uses visibility structures (the framebuffer) to encode shading information. The scattering of shaded values due to lens and motion blur compounds the difficulties. However, this is not a fundamental algorithmic problem with decoupling, but an artificial constraint of the multi-pass fashion in which deferred shading is currently implemented. We believe decoupling (like MSAA) would be compatible with a pipeline built to be natively aware of deferred shading.

Finally, the abstract idea of decoupled sampling is not tied to a rasterization-based hardware rendering architecture. It is easy to see that a whole continuum of applications, ranging from the standard hardware pipelines we have studied through micropolygon renderers to ray tracing can be easily formulated using our recipe; the main difference of such potential implementations to usual caching schemes is that decoupled sampling uses a consistent mapping between separate shading and visibility domains, and deterministically memoizes the shading results, rather than opportunistically reusing results which can yield effectively irregular shading sampling. In the future we intend to study decoupled sampling in a ray tracing environment. This presents several interesting possibilities, such as introducing shading LOD, as determined by ray differentials, as an additional dimension in the cache key.

ACKNOWLEDGMENTS

Jason Mitchell provided generous help in capturing real game scenes. All Half-Life and Team Fortress content is courtesy of Valve Software. This work was supported by Singapore-MIT Gambit and a grant from Intel Corp. Jonathan Ragan-Kelley was supported by NVIDIA and Intel PhD fellowships.

REFERENCES

AKELEY, K. 1993. RealityEngine graphics. In *Proceedings of SIGGRAPH 93*. Computer Graphics Proceedings, Annual Conference Series. 109–116.

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

BOULOS, S., LUONG, E., FATAHALIAN, K., MORETON, H., AND HANRAHAN, P. 2010. Space-time hierarchical occlusion culling for micropolygon rendering with motion blur. In *Proceedings of High Performance Graphics 2010*. 11–18.

BRUNHAVER, J., FATAHALIAN, K., AND HANRAHAN, P. 2010. Hardware implementation of micropolygon rasterization with motion and defocus blur. In *Proceedings of High Performance Graphics 2010*. 1–9.

BURNS, C. A., FATAHALIAN, K., AND MARK, W. R. 2010. A lazy object-space shading architecture with decoupled sampling. In *Proceedings of High Performance Graphics 2010*. 19–28.

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

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

COOK, R. L., PORTER, T., AND CARPENTER, L. 1984. Distributed ray tracing. In *Computer Graphics (Proceedings of SIGGRAPH 84)*. 137–145.

ELDRIDGE, M. 2001. Designing graphics architectures around scalability and communication. Ph.D. thesis, Stanford University.

FATAHALIAN, K., BOULOS, S., HEGARTY, J., AKELEY, K., MARK, W. R., MORETON, H., AND HANRAHAN, P. 2010. Reducing shading on gpus using quad-fragment merging. *ACM Transactions on Graphics 29,* 4 (July), 67:1–67:8.

FATAHALIAN, K., LUONG, E., BOULOS, S., AKELEY, K., MARK, W. R., AND HANRAHAN, P. 2009. Data-parallel rasterization of micropolygons with defocus and motion blur. In *Proceedings of High Performance Graphics 2009*. 59–68.

FISHER, M., FATAHALIAN, K., BOULOS, S., AKELEY, K., MARK, W. R., AND HANRAHAN, P. 2009. DiagSplit: Parallel, crack-free, adaptive tessellation for micropolygon rendering. *ACM Transactions on Graphics 28,* 5 (Dec.), 150:1–150:10.

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

HAMMON, E. 2007. Practical post-process depth of field. In *GPU Gems 3*, H. Nguyen, Ed. Addison Wesley, Chapter 28, 583–605.

HASSELGREN, J. AND AKENINE-MÖLLER, T. 2006. An efficient multi-view rasterization architecture. In *Rendering Techniques 2006: 17th Eurographics Workshop on Rendering*. 61–72.

JONES, T., PERRY, R., AND CALLAHAN, M. 2000. Shadermaps: a method for accelerating procedural shading. Tech. Rep. 2000-25, Mitsubishi Electric Research Labs.

LEE, S., EISEMANN, E., AND SEIDEL, H.-P. 2009. Depth-of-field rendering with multiview synthesis. *ACM Transactions on Graphics 28,* 5 (Dec.), 134:1–134:6.

MAX, N. L. AND LERNER, D. M. 1985. A two-and-a-half-D motion-blur algorithm. In *Computer Graphics (Proceedings of SIGGRAPH 85)*. 85–93.

MOLNAR, S., COX, M., ELLSWORTH, D., AND FUCHS, H. 1994. A sorting classification of parallel rendering. *IEEE Computer Graphics and Applications 14,* 4, 23–32.

NEHAB, D., SANDER, P. V., LAWRENCE, J., TATARCHUK, N., AND ISIDORO, J. R. 2007. Accelerating real-time shading with reverse reprojection caching. In *Graphics Hardware 2007*. 25–35.

OLANO, M. AND GREER, T. 1997. Triangle scan conversion using 2d homogeneous coordinates. In *1997 SIGGRAPH / Eurographics Workshop on Graphics Hardware*. 89–96.

OWENS, J. D., KHAILANY, B., TOWLES, B., AND DALLY, W. J. 2002. Comparing Reyes and OpenGL on a stream architecture. In *Graphics Hardware 2002*. 47–56.

PATNEY, A. AND OWENS, J. D. 2008. Real-time Reyes-style adaptive surface subdivision. *ACM Transactions on Graphics 27,* 5 (Dec.), 143:1–143:8.

RAGAN-KELLEY, J., KILPATRICK, C., SMITH, B. W., EPPS, D., GREEN, P., HERY, C., AND DURAND, F. 2007. The Lightspeed automatic interactive lighting preview system. *ACM Transactions on Graphics 26,* 3 (July), 25:1–25:11.

ROSADO, G. 2007. GPU gems 3. Addison Wesley, Chapter Motion Blur as a Post-Processing Effect, 575–581.

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. *ACM Transactions on Graphics 27,* 3 (Aug.), 18:1–18:15.

SITTHI-AMORN, P., LAWRENCE, J., YANG, L., SANDER, P. V., AND NEHAB, D. 2008. An improved shading cache for modern GPUs. In *Graphics Hardware 2008*. 95–101.

SLOAN, P.-P., LUNA, B., AND SNYDER, J. 2005. Local, deformable precomputed radiance transfer. *ACM Transactions on Graphics 24,* 3 (Aug.), 1216–1224.

STOLL, G., MARK, W. R., DJEU, P., WANG, R., AND ELHASSAN, I. 2006. Razor: An architecture for dynamic multiresolution ray tracing. Tech. Rep. 06-21, University of Texas at Austin.

TORBORG, J. AND KAJIYA, J. 1996. Talisman: Commodity real-time 3D graphics for the PC. In *Proceedings of SIGGRAPH 96*. Computer Graphics Proceedings, Annual Conference Series. 353–364.

YANG, L., SANDER, P. V., AND LAWRENCE, J. 2008. Geometry-aware framebuffer level of detail. *Computer Graphics Forum 27,* 4 (June), 1183–1188.

ZHOU, K., HOU, Q., REN, Z., GONG, M., SUN, X., AND GUO, B. 2009. Renderants: Interactive reyes rendering on gpus. *ACM Transactions on Graphics 28,* 5 (Dec.), 155:1–155:11.

APPENDIX

A. APERTURE-CONTINUOUS EDGE FUNCTIONS

Four-dimensional edge functions may be derived following Olano and Greer [1997]. Their ordinary 2D homogeneous edge equations are exactly the rows of the inverse of the 3×3 matrix $\mathbf{V}$ formed whose columns are the xw, yw, w coordinates of the three vertices of the triangle. Now, the effect of the lens parameters u and v are shears in camera space: x and y are translated parallel to the image plane by amounts that depend on the distance of the vertex to the focal plane. This shear can be represented in clip space by the matrix

$$\mathbf{S}(u,v) = \begin{pmatrix} 1 & 0 & -Hu/J & Hu \\ 0 & 1 & -Iv/J & Iv \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix},$$

where H, I, J are constants that depend on the world-space focusing distance, aperture size, field of view, and near and far clipping plane distances. The final 4D edge functions are given by the rows of $(\mathbf{S}(u,v)\mathbf{V})_3^{-1}$, where the subscript denotes the dropping of the third column and row (the z coordinates). Expanding the inverse using Cramer's rule reveals the form of each edge function:

$$\frac{1}{N(u,v)} \begin{pmatrix} Av + B \\ Cu + D \\ Eu + Fv + G \end{pmatrix}^T. \tag{1}$$

Here $N(u,v) = \det(\mathbf{S}(u,v)\mathbf{V})_3$ is a normalization term shared between all three edge functions and $A \ldots G$ are different for each of the three functions. $N(u,v)$ is a linear polynomial of u, v. Once the edge function has been evaluated for a given u, v pair, testing a pixel $(x, y, 1)$ against it is computed as a dot product. Note that (B, D, G) form the usual 2D edge function in the absence of defocus. Also note that the normalization is required only for computing barycentrics after a hit has already been determined using the unnormalized formula. Owing to the well-mannered nature of the depth-of-field warp, the 4D edge functions are, interestingly, cheaper to set up and to evaluate per visibility sample than time-continuous 3D edge functions.