High Performance Graphics (2012)
C. Dachsbacher, J. Munkberg, and J. Pantaleoni (Editors)

Design and Novel Uses of Higher-Dimensional Rasterization

J. Nilsson[1] P. Clarberg[1] B. Johnsson[1,2] J. Munkberg[1] J. Hasselgren[1] R. Toth[1] M. Salvi[1] T. Akenine-Möller[1,2]

[1]Intel Corporation [2]Lund University

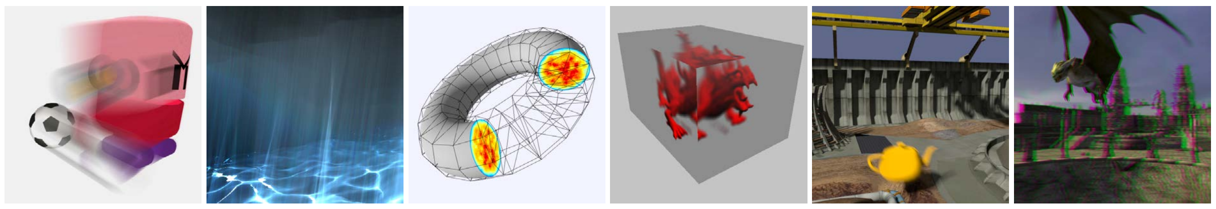

continuous collision detection
caustics
sampling
glossy reflections/refractions
motion blurred soft shadows
multi-view rendering

Figure 1: We demonstrate a number of novel use cases for (hypothetical) higher-dimensional rasterization hardware. In the first three, we exploit the volumetric extents of time-continuous and defocused triangles to perform geometric computations, i.e., occlusion and collision detection, caustic rendering, and use the rasterizer as a flexible tool for sampling. In the second set of applications, we render a scene from multiple viewpoints simultaneously, in order to achieve effects such as glossy reflections/refractions, soft shadows, and multi-view rendering.

Abstract

This paper assumes the availability of a very fast higher-dimensional rasterizer in future graphics processors. Working in up to five dimensions, i.e., adding time and lens parameters, it is well-known that this can be used to render scenes with both motion blur and depth of field. Our hypothesis is that such a rasterizer can also be used as a flexible tool for other, less conventional, usage areas, similar to how the two-dimensional rasterizer in contemporary graphics processors has been used for widely different purposes other than the original intent. We show six such examples, namely, continuous collision detection, caustics rendering, higher-dimensional sampling, glossy reflections and refractions, motion blurred soft shadows, and finally multi-view rendering. The insights gained from these examples are used to put together a coherent model for what a future graphics pipeline that supports these and other use cases should look like. Our work intends to provide inspiration and motivation for hardware and API design, as well as continued research in higher-dimensional rasterization and its uses.

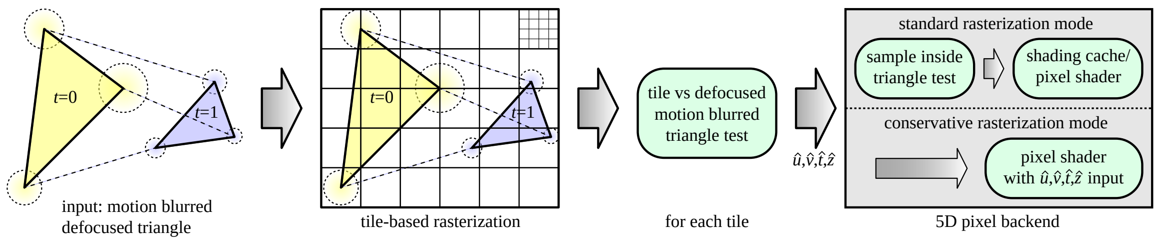
Categories and Subject Descriptors (according to ACM CCS): I.3.1 [Computer Graphics]: Hardware architecture—Graphics processors I.3.7 [Computer Graphics]: Computer Graphics—Three-Dimensional Graphics and Realism-Color, shading, shadowing, and texture

1. Introduction

The two-dimensional rasterizer in a graphics processor is a highly optimized fixed-function unit. Conceptually, it is just a loop over samples bounded by a geometric triangle, with the capability to execute a program for each covered sample/pixel. Besides using it for rendering three-dimensional geometry, the rasterizer has been employed as a tool for a wide variety of topics, including, e.g., shadow algorithms [Cro77, Wil78], constructive solid geometry [Wie96], Voronoi diagrams [HKL*99], collision detection [GRLM03], caustics [EAMJ05], ray tracing for global illumination [Hac05], curved surfaces [LB06], and more.

We note that *higher-dimensional rasterization* is currently a very active research topic, and many efficient algorithms are emerging [FLB*09, MCH*11, LAKL11, AMTMH12, MAM12], as well as hardware studies [BFH10]. The conventional usage area is correct visibility for motion blur and depth of field. The assumption in this paper is that there will be an efficient fixed-function 5D rasterizer, including efficient shading [RKLC*11], in future graphics processors. Our hypothesis is that this new machinery can be used for many other purposes beyond motion/defocus blur. We have identified six such unconventional uses, of which some are dependent on shading and some only utilize time, lens or depth bounds produced by the rasterizer.

DOI: 10.2312/EGGH/HPG12/001-011



t=0
t=1
input: motion blurred defocused triangle
tile-based rasterization
tile vs defocused motion blurred triangle test
for each tile
$\hat{u},\hat{v},\hat{t},\hat{z}$
standard rasterization mode
sample inside triangle test
shading cache/ pixel shader
conservative rasterization mode
pixel shader with $\hat{u},\hat{v},\hat{t},\hat{z}$ input
5D pixel backend

Figure 2: An overview illustration of our five-dimensional rasterization pipeline. The input is defocused, motion blurred triangles, rasterized in tile order. For each tile, a triangle-against-tile test is performed, and only tiles with non-empty intervals $(\hat{u},\hat{v},\hat{t})$ continue to the next stage. In the backend (top right), sample-inside-triangle testing is done, and surviving samples shaded. In conservative rasterization mode (bottom right), a tile size of 1×1 pixels is used, and the sample-inside-triangle test is bypassed. In this case, the pixel shader is executed once per tile and receives the tile intervals as inputs.

The applications can broadly be divided into two categories. The first set of algorithms exploit the fact that a moving and/or defocused triangle *sweeps out a volume* in three to five-dimensional space. This capability opens up for a number of less intuitive use cases. We show examples in *i*) continuous collision detection, *ii*) caustic rendering, and *iii*) higher-dimensional sampling. The stochastic rasterizer can also be used to simultaneously render a scene from *many different viewpoints*, through appropriate choice of lens, focus plane, samples, and use of the time dimension. The benefits are increased shader/texture coherence and fewer geometry passes. To illustrate this, we focus on *iv*) glossy reflections/refractions, *v*) motion blurred soft shadow mapping, and *vi*) stereoscopic and multi-view rendering with motion and defocus blur. Some screenshots are shown in Figure 1.

Based on the insights gained from these examples, we present a complete model for an efficient and flexible stochastic rasterization pipeline, including novel additions such as conservative depth computations. Our work is rather long term and speculative research in the sense that it assumes the existence of something that is not (yet) readily available. We provide a first set of algorithms for each topic, but we want to emphasize that we do not explore these fully. The core contributions are that our work:

1. Sketches out a space of new applications for higher-dimensional rasterization.
2. Presents the first coherent model for efficient stochastic rasterization, including some novel additions.
3. Highlights many practical design aspects, e.g., the importance of conservative rasterization and flexible sampling.
4. Provides motivation for further research in stochastic rasterization and its uses, as well as hardware and APIs.

We will start by describing our pipeline, and then present the specific novel use cases that have motivated our design.

2. Our Five-Dimensional Rasterization Pipeline

In this section, we describe the 5D stochastic rasterization pipeline that is used throughout this paper. An illustration of our pipeline can be found in Figure 2. The input is a triangle with defocused vertices at $t=0$ and at $t=1$ in order to handle both motion blur and depth of field. It is assumed that the vertices move linearly in world space, and we assume the thin lens model for depth of field rasterization, similar to other work [FLB*09, LAKL11, AMTMH12]. Furthermore, our pipeline can efficiently disable depth of field and only render motion blur, and vice versa. This is useful in several applications, e.g., continuous collision detection and caustics (3D), and for glossy effects (4D), see Sections 3, 4, 6.

Each pixel is assumed to have n samples, where each sample has a fixed screen space position, (x,y), fixed lens parameters, (u,v), and a time, t. Although most applications need well-distributed stochastic samples, which may be procedurally generated [JK08], we have found several cases where other sampling patterns are beneficial or even required. This includes glossy rendering, soft shadows, and stereoscopic or multi-view graphics (Sections 6, 7, 8). We thus propose the hardware uses a reasonably large programmable sample table, coupled with a scrambling method [KK02] for increased randomness when desired.

Our traversal order is tile-based since such a rasterization order has been shown to be beneficial in several different aspects [MCH*11, LAKL11, AMTMH12, MAM12]. This includes reduced memory bandwidth usage to buffers and textures, efficient handling of widely varying triangle sizes, and the possibility of efficient depth buffer compression [AHAM11]. In addition, primitive setup is only done once, and tile tests allow for arbitrary sample positions, which makes for higher quality samples. Consequently, as current graphics processors already employ a tile-based traversal order, the transition from two to five dimensions does not require a major pipeline re-design in this respect.

The output of a tile test is a set of intervals, $(\hat{u},\hat{v},\hat{t})$, where an interval is described as $\hat{t} = [\underline{t},\bar{t}]$, which is all t such that $\underline{t} \leq t \leq \bar{t}$. These intervals are conservative estimates of the samples that may overlap with the triangle being rendered. Details on how to compute them in 5D can be found in previous work [LAKL11, MAM12]. In some cases, e.g., caustics rendering (Section 4), we have found tighter bounds for the time interval [GDAM10] to be beneficial, so the exact implementation remains to be decided. In addition, we support optional, axis-aligned user clip planes, which can be specified to further limit the extents in *uvt* of a triangle. Each clip plane is given as a lower or upper boundary, e.g., $\bar{t}_{\text{clip}}$, and min/max operations are used to find the final set of sampling intervals. This is a minor extension, but it allows more flexible use of the rasterizer as a sampling engine (Section 5), and it is consistent with current APIs that support clip planes in x, y. We also introduce a way to compute a conservative interval, $\hat{z}$, of the depth over the tile in order to perform $z_{\max}$-culling [GKM93] and $z_{\min}$-culling [AMS03]. See Appendix A. The depth interval is also directly used in several applications, including 5D occlusion queries and continuous collision detection (Section 3).

As described in Figure 2, only tiles where all of the intervals, $\hat{u}, \hat{v}, \hat{t}$, are non-empty continue down the pipeline, which is common practice. In the *standard* mode (top right in the figure), each sample within those intervals is inside-tested against the triangle, and surviving samples shaded. Depending on the application, we use a shading memoization cache [RKLC*11], hereafter referred to as a shading cache, or execute the pixel shader per sample. The shading cache is generally preferable, but some use cases like caustics rendering and sampling (Sections 4, 5) require a 1:1 mapping between samples and shader executions.

By considering a large set of potential use cases, we have found that it is often extremely valuable to be able to *disable* sample testing altogether, and directly feed the intervals $(\hat{u}, \hat{v}, \hat{t}, \hat{z})$ output by the tile test to the pixel shader. In this case, the pipeline behaves as a *conservative* rasterizer at the granularity of the tile size [AMA05] in five dimensions, and the pixel shader is executed per tile. We show examples of using this for 5D occlusion queries, collision detection, caustics, and sampling (Sections 3, 4, 5).

In the following sections, we will present each of our example applications for five-dimensional rasterization in more detail, in order to motivate the above design choices.

3. Five-Dimensional Occlusion Queries

Here, we generalize standard occlusion queries to five dimensions, which can be used for motion blurred and defocused occlusion culling. In addition, we also show that a time-dependent occlusion query (no defocus) can be used for continuous collision detection. Most contemporary graphics processors and 3D APIs support a feature called *occlusion query* [SA11]. An occlusion query (OQ) can be used to find out how many fragments, n_f, generated by a set of polygons pass the depth test. For example, the faces of a bounding box around a complex character can be used as an occlusion query, and if n_f is zero then the character is occluded with respect to the contents of the depth buffer.

A *sample-based* way to extend occlusion queries to account for motion and defocus blur, is to render the object to the depth buffer using the samples of the 5D rasterizer and count the number of fragments that pass the test, i.e., in the

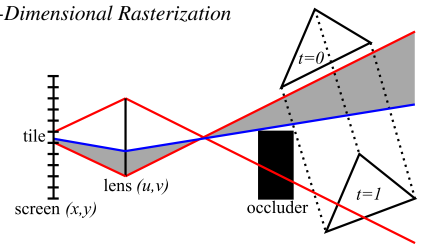


Figure 3: The gray area is the visible (i.e., not occluded) region in world space seen from a particular tile, where samples will pass the depth test and report the triangle as visible.

traditional sense. Figure 3 shows a moving triangle that is in part occluded by other geometry during the shutter interval. The gray area represents the time and lens intervals during which the moving triangle is visible for a particular tile. In a sampled approach, samples in the gray area will pass the depth test and report the triangle as visible.

For motion blur and defocus blur, a relatively large number of samples per pixel (e.g., $16-64$) may be needed for sufficient occlusion query precision. Each occlusion sample requires additional computation and bandwidth, but in many cases, the number of fragments that pass is not needed. We therefore propose to use a five-dimensional *interval-based* occlusion query, which reduces these requirements substantially. An interval-based occlusion query is similar to motion blur z-culling techniques [AMMH07, BLF*10], where $z_{\min}/z_{\max}$-values for tiles are stored in a fast memory. A tile may also have several $z_{\min}/z_{\max}$-values for different time intervals. In a 5D setting, this concept is extended to the u and v lens parameters, i.e., $z_{\min}/z_{\max}$-values are stored for disjoint boxes covering the entire *uvt*-space for the tile. Consequently, the occlusion rate is dependent on the number of $z_{\max}$-values per tile and also the tile size. With more subdivisions in time, efficiency for moving occluders increases. Thus, as a complement to sample-based OQs, the user can trade off occlusion rate for OQ speed.

When performing the query, the rasterizer provides intervals, $\hat{u}$, $\hat{v}$, $\hat{t}$, and $\hat{z}$, for each primitive and visited tile. The $\hat{u}$, $\hat{v}$, and $\hat{t}$ intervals are used to decide which boxes in *uvt*-space need to be considered when doing the depth comparisons, while the $\hat{z}$ intervals (conservative, not generating any false positives) are used in the actual comparisons. This means that individual samples for each pixel in a tile need not be touched, i.e., no expensive sample-inside-triangle tests or depth computation have to be performed. As soon as one comparison fails, the geometry is not fully occluded, and further rasterization and testing can be aborted.

Continuous Collision Detection Collision detection (CD) algorithms can be broadly divided into *discrete* and *continuous* methods [RAC02]. A drawback of discrete methods is that they may miss collisions involving fast-moving objects, e.g., a bullet shot through a thin paper. Continuous collision detection (CCD) algorithms avoid these problems,

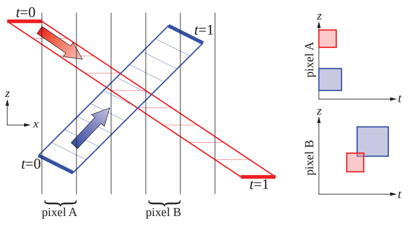


Figure 4: A two-dimensional example of continuous collision detection between two lines. The red (moving down) and blue (moving up) lines move from $t=0$ to $t=1$. For pixels A and B, the conservative bounds for time, t, and depth, z, are visualized to the right. The lines can never overlap in pixel A, since the red and blue boxes do not overlap in the tz-plane. For pixel B, the lines do overlap.

sometimes called “tunneling” effects, by computing the first time of contact between objects.

To accelerate CD, many image-based approaches running on the GPU have been proposed, starting with Shinya and Forgue’s work [SF91]. Govindaraju et al. [GRLM03, GLM06] presented an algorithm which computes the *potentially colliding set* (PCS) for a set of objects. Pairs of objects not in the PCS will definitely not collide. Below, we show how to compute the PCS for *CCD* using a three-dimensional rasterizer to determine whether two objects do not collide in a certain period of time. A generalization to many objects follow the lines of previous work [GRLM03].

We have chosen to describe our approach in relation to Govindaraju et al.’s work, which works as follows. To detect whether two objects do not (conservatively) collide at a certain instant of time, the first object is rendered to the depth buffer. In a second pass, the depth test is reversed, and the second object is rendered with an occlusion query. If no fragments pass, then there is no collision between the two objects. We call this an overlap test. This is done orthographically in the xy, xz, and yz planes, and also from opposite directions, which sums to six overlap tests. If at least one of these tests indicates that there is no collision, then the pair does not belong to the PCS. As motion is introduced, then the sample-based OQ, described above, can replace the “static” OQ, but a conservatively correct PCS is not computed when sampling at discrete times.

Instead, we propose a substantially different approach for CCD, based on interval-based occlusion queries, which use a conservative 3D rasterizer. We start with the case of two triangles. For each pixel and triangle, where a 1×1 tile test indicates overlap, we insert the time and depth intervals ($\hat{t}=[\underline{t},\bar{t}]$, and $\hat{z}=[\underline{z},\bar{z}]$), into a per-pixel buffer, e.g., let RGBA $=[\underline{t},\underline{z},\bar{t},\bar{z}]$ be an axis-aligned bounding box in tz. If the tz-boxes of two triangles overlap, they potentially collide in that pixel. This is illustrated in Figure 4.

To handle many triangles per object, we initialize each pixel to represent an empty bounding box. Then, for each

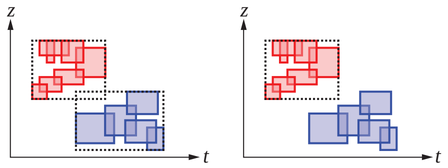


Figure 5: Collision detection in a single pixel between an object whose tz-fragments are red, and one object whose fragments are blue. Left: bounding boxes of the union of all tz-fragments for the red and blue objects are accumulated, and overlap tested. Right: alternatively, test blue tz-fragments individually for overlap against the bounding box of all the red fragments. This would be more efficient in this example, since it would not detect any overlap.

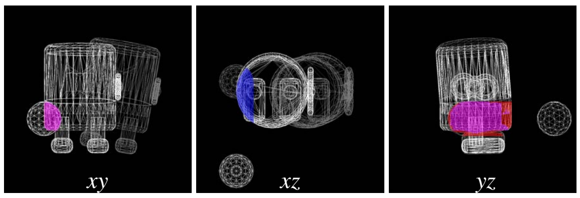


Figure 6: From left to right: orthographic xy, xz, and yz visualizations of pixels overlapping in time and in depth. For each pixel that overlaps in depth, the pixel’s blue channel is set, and if there is an overlap in time, the red channel is set. Hence, only purple pixels overlap in time and in depth. Current algorithms only use overlap in depth, while we use both time and depth. In this case, we detect that the objects do not overlap in the xz projection (middle image), while previous algorithms would put them in the potentially colliding set, since the two objects overlap in depth in all views.

triangle in the first object, the union of the current bounding box and the incoming fragment’s box is computed using min/max blending. For the second object, a similar buffer is generated. When both buffers have been created, a box vs box overlap test is performed for pixels with the same xy-coordinates (Figure 5, left). For a particular pixel, no collision can occur if the boxes do not overlap. Alternatively, when the second object is rasterized, each tz-interval can be directly tested for overlap with the first object’s bounding box (Figure 5, right).

We have implemented our algorithm in a simulated rasterization pipeline. An example is shown to the left in Figure 1, where a “toaster” is dodging an approaching soccer ball. For CCD, we use orthographic 3D rasterization in the xy, xz, and yz planes, i.e., we use only three passes compared to six passes using previous algorithms [GRLM03]. We visualize the pixels that overlap in depth and in time in Figure 6. As can be seen, our method detects that these two objects do *not* overlap, as opposed to traditional techniques using overlap in depth only.

To summarize, for many applications, continuous (as op-

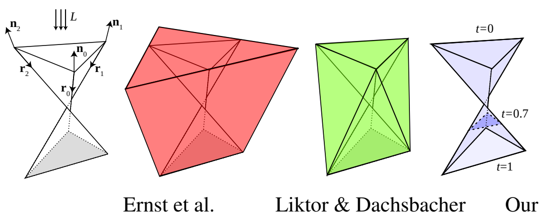


Figure 7: The caustic volume from a generator triangle, where the incoming light L is refracted based on the normals **n** along directions **r** is shown to the left. Then follows three methods for bounding the volume. Note that the rasterized areas differ substantially.

posed to discrete) CD is indeed required. To that end, we argue that the method of Govindaraju et al. does not take time into account and adds all pairs with overlapping trajectories to the PCS (and incur higher cost). Our contributions are (i) an improved z-interval estimate (appendix), (ii) a first description of continuous CD for triangle soups on the GPU, and (iii) an inexpensive detection test (2D AABB).

We believe that occlusion queries for higher dimensional rasterization will prove to be very useful, and we foresee a wide range of uses. This includes, for example, modifications to hierarchical occlusion culling [BWPP04] with occlusion queries so that motion blur, depth of field, and their combination can be even more efficiently rendered. For time-continuous collision detection, there are many avenues for further research. This includes self-collision, finding the time of contact, and different hierarchical strategies for improved performance.

4. Caustics Rendering

Caustics are the formation of focused patterns from light refracted and reflected in curved surfaces or volumes. These patterns add realism to a scene, e.g., the beautiful, shimmering light on the bottom of a swimming pool. The effect can be created by simulating the traversal of photons, calculating the local light density at the receiving surface. Physically correct methods [JC98], which inherently handle caustics as well as other phenomena, are still too costly for real-time use. Using graphics hardware, several approaches that trace light beams have been proposed [Wat90, IDN02, EAMJ05, LD11], as well as splatting-based methods [SKP07].

In beam tracing techniques, *generator* triangles represent the origin of caustic beams, e.g., an ocean surface. The beams are created by letting incoming light be refracted or reflected in the generator triangles. Ernst et al.'s algorithm [EAMJ05], later refined by Liktor and Dachsbacher [LD11], generates surface or volume caustics by intersecting the caustic beams with receiving surfaces or eye rays, respectively. In dynamic scenes, the reflected/refracted rays and the set of caustic beams are generated in each frame. The generated caustics volumes are then rasterized, similar to shadow volume rendering, and for each covered pixel, the light contribution of each volume is computed. A tight enclosure of each beam is essential to decrease its screen space area and reduce the fill rate when accumulating the light intensity. As the sides of the caustic beams describe bilinear patches, such fitting is not clear-cut. Ernst et al. use a bounding prism approach with touching planes, while Liktor and Dachsbacher refine this for heavily warped volumes.

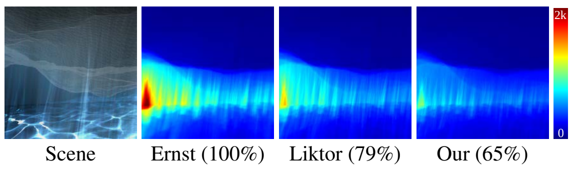


Figure 8: A visualization of the number of pixel shader executions relative to the method by Ernst et al. for a volume caustics example. Dark red represents 2,200 pixel shader executions. We use a motion blur rasterizer to render the caustic volumes as moving triangles, which results in less fill-rate than the two bounding volume methods.

A motion blur rasterizer can in this setting be used to model the caustic volume. Instead of bounding the bilinear patch sides of the caustic beam with a volume (see Figure 7), the moving triangle is directly rasterized with the generator triangle at $t = 0$, and the end triangle at $t = 1$ (with backface culling disabled). The pixel shader is invoked for pixels covered by the moving triangle at *any* time, $t \in [0, 1]$. To find this overlap, and to reduce the fill-rate as much as possible, we compute the exact time interval when the pixel is inside all three moving triangle edges [GDAM10].

With our approach, a bounding volume is not needed and the resulting fill rate is, in general, lower. Figure 7 shows a warped triangle, and compares the screen space coverage of the bounding volume methods with the area of the motion blurred triangle. In Figure 8, the three approaches are compared in an ocean scene with 130,000 caustic volumes. Here, our approach has lower fill-rate than the other two, and avoids bounding volumes altogether.

The modeling of beams as moving triangles may enable the use of rasterizer output in various other algorithms, e.g., to simplify pixel shaders. It would be interesting to explore this concept in a broader sense. For instance, for volumetric caustics, modeling of light attenuation through scattering and absorption could possibly be approximated by the rasterizer's depth interval; $\hat{z} = [\underline{z}, \overline{z}]$, i.e., $\overline{z} - \underline{z}$ can be used as an approximation to the distance the eye ray travels in the beam. For surface caustics, $\hat{z}$ may be used to cull parts of the beams that are completely occluded, and parts of the beams that are definitely in front of the geometry in the depth buffer for the current tile. By doing so, it is expected that the majority of expensive pixel-in-volume tests can be avoided.

5. Sampling

Many applications in computer graphics, scientific and medical visualization, and engineering need to draw random samples over an arbitrary domain. Examples in graphics include rendering, texture synthesis, and object placement [PH10]. A standard approach is to divide the sampling domain into simpler shapes, pick a shape at random with the correct probability, and place a sample in it. Although not trivial to parallelize, several papers have shown it can be done in general GPGPU code [Wei08, GM09, EDP*11].

Our core insight is that a higher-dimensional rasterizer performs many similar operations, including scheduling/dispatching of the work, random selection, and rejection sampling against complex shapes in 3D–5D. Rasterization also naturally decouples the sample placement from the choice of tessellation, i.e., a sample that misses one primitive falls into an adjacent, which is an added benefit compared to methods that sample shapes independently. All this ends up being a fair amount of code otherwise. The underlying assumption is that a hardware rasterizer would be more power-efficient, even if the algorithms have to be tweaked a bit. Naturally, this is currently hard to prove, so we will focus on sketching the idea and a few applications.

The 5D Rasterizer as a Volumetric Sampler The stochastic rasterization process is usually illustrated in clip space by moving/shearing a triangle's vertices, but it can equivalently be seen as the triangle carving out a volumetric shape, $\mathcal{S}$, in 5D *xyuvt*-space. This domain is filled with uniformly distributed samples, and the rasterizer quickly finds which ones are inside $\mathcal{S}$ through tile tests as described in Section 2. The analogy in 2D is a triangle in screen space, which cuts out a set of uniform samples in *xy*. The volume of $\mathcal{S}$ directly controls the *expected* number of samples, N, placed in it, i.e., $E[N] = \rho V(\mathcal{S})$, where ρ is the sampling density.

In 3D *xyt*-space, $\mathcal{S}$ is a generalized triangular prism with the triangular end caps at $t = 0$ and $t = 1$ (or a tighter range if user clip planes in t are used). Note that due to varying per-vertex motion and perspective foreshortening, the edges connecting the end caps may be *curved* and the sides are usually non-planar in *xyt*. This is non-intuitive, as the edges are always straight lines in clip space, although the sides may be bilinear patches [AMMH07]. When one extra dimension, u, is added, each vertex is sheared in x as u varies. The shear may be non-linear as it is a function of depth, which is time-dependent. The carved out hypervolume has 12 vertices, with end caps being generalized triangular prisms in 3D. Finally, in 5D, $\mathcal{S}$ is a complex shape with 24 vertices.

To sample an arbitrary domain, $\mathcal{D}$, we first construct a conservative bounding volume, $\mathcal{B}$, so that $\mathcal{D} \subseteq \mathcal{B}$. The bounding volume is then tessellated into a number of non-overlapping, adjacent primitives, $\mathcal{S}$, which are individually rasterized. In 2D, this corresponds to tessellating the interior of an arbitrary bounding polygon into triangles. Due to rasterization tie-breaking rules, any sample is guaranteed to be placed in at most one primitive. Finally, the pixel shader performs an analytical test per sample (in $\mathcal{B}$), to reject any remaining samples outside of $\mathcal{D}$. Figure 9 shows an illustrative example in 3D. The exact sample placement is given by the built-in low-discrepancy sample generator (Section 2). Alternatively, the conservative mode can be used and any number of random samples generated over the bounds $\hat{u}\hat{v}\hat{t}$. In this case, the samples have to be manually tested against $\mathcal{S} \cap \mathcal{D}$, not just $\mathcal{D}$, since the bounds may overlap. Collectively, the result is a uniform random sampling of $\mathcal{D}$; the hardware rasterizer performs an initial fast, but coarse sample culling, and the pixel shader performs a final fine-grained test (which can be skipped if $\mathcal{B} = \mathcal{D}$).

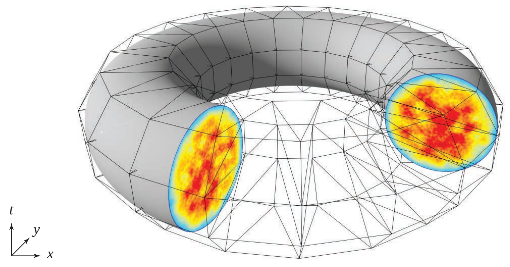


Figure 9: The stochastic rasterizer can be used as a flexible sample generator in, e.g., numerical integration. In this example, generalized triangular prisms conservatively bound the integration domain (here a torus). The prisms are rasterized as motion blurred triangles in *xyt*-space (note the t-axis pointing upwards). At each sample, the integrand is evaluated and accumulated in the pixel shader. Similar strategies apply to other sampling problems.

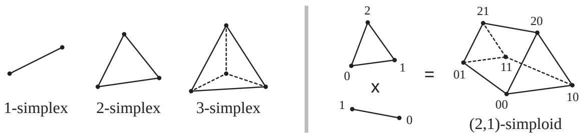


Figure 10: The n-simplex is the simplest possible n-dimensional polytope, consisting of $n+1$ vertices, with all pairs connected by edges. The product of m simplices makes an $(n_1, \ldots, n_m)$-simploid in $\sum n_i$ dimensions. The figure shows simplices in up to three dimensions, and an $(2,1)$-simploid, i.e., triangular prism.

To make $\mathcal{S}$ easier to work with, we can restrict the vertex locations in such a way that all sides of the primitive are planar. For example, in the 3D case, if we restrict each vertex to not move in depth, and each pair of edges in the triangular end caps of the prism to be parallel, all edges will be straight and all sides planar. The result is a tapered triangular prism. In general, we can place the vertices so that $\mathcal{S}$ is an n-polytope (a geometric object with flat sides in n dimensions), where $n \leq 5$. The class of polytopes generated by the stochastic rasterizer in 3D, 4D, and 5D, will formally be $(2,1)$, $(2,1,1)$, and $(2,1,1,1)$-simploids [Moo92], respectively. See Figure 10 for examples.

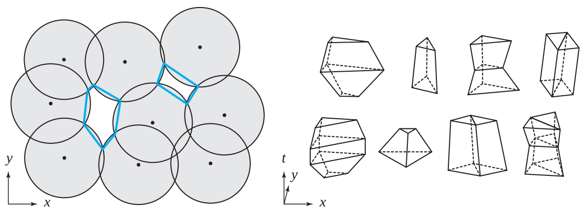


Figure 11: Optimized dart throwing keeps track of the voids, i.e., unsampled regions, in the sampling domain to guide the insertion of new points. We represent voids as polygons/polytopes, as shown in blue on the left. Candidate points are generated in parallel by stochastically rasterizing all voids, after random displacement and appropriate scaling. Some examples in 3D are shown on the right.

Accelerated Dart Throwing Poisson-disk sampling using dart throwing [Coo86] is one of the intriguing use cases for our framework. The resulting blue noise samples are ideal for many applications, e.g., stippling and texture synthesis. In its basic form, dart throwing generates a large number of random candidate points over $\mathcal{D}$, but only keeps the ones that are separated by a certain minimum distance. Modern algorithms accelerate the process by tracking the voids, V, between samples using, e.g., octree cells or general polytopes [Wei08,EDP*11]. Conceptually, the sampling domain, $\mathcal{D}$, is first subdivided into voids, which are put in an "active" list, and the following operations are performed:

1. Select a void, V, from the active list with probability according to its volume.
2. Choose a random candidate point, p, in the void.
3. Check if p meets the minimum distance criteria for the neighboring points, and if so, add it to the point set.
4. Check if V is completely covered, and if not, split it into smaller voids that are added to the active list.

With a stochastic rasterizer, we can perform steps (1) and (2) on a large number of voids in parallel. Each void is represented as a single (or union of) simploids compatible with the rasterizer. Figure 11 shows some examples. The expected number of candidate points in each void is controlled by uniformly scaling it, and the voids are independently and randomly displaced to avoid bias. All voids in the active list are then rasterized with the depth test disabled, and the generated candidate points stored to a linear array (e.g., using an append buffer). A compute pass finally processes the points to eliminate conflicts, and updates the active list. When the active list is empty, a *maximal* distribution has been achieved, i.e., no more points can be inserted.

Adaptive White Noise Non-uniformly distributed samples in n dimensions can be achieved by sampling uniformly over an appropriate domain in n+1 dimensions, and orthographically projecting the samples back to n dimensions. Intuitively, the shape of the sampling domain in $\mathbb{R}^{n+1}$ is defined by viewing the density function, $\rho(\mathbf{x})$, where $\mathbf{x} \in \mathbb{R}^n$, as a height field. By generating samples under the height field $(\mathbf{x}, \rho(\mathbf{x})) \in \mathbb{R}^{n+1}$, and projecting on $\mathbf{x}$, we effectively get samples distributed according to $\rho(\mathbf{x})$.

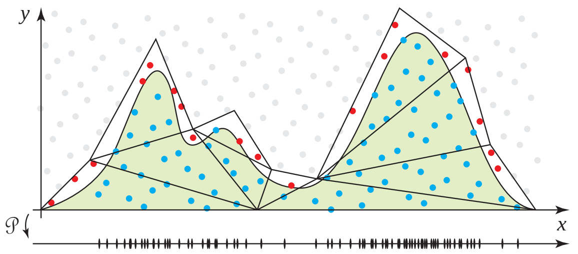


Figure 12: Non-uniform samples can be created by sampling the region under the density function (shown in green), $\rho(\mathbf{x})$, uniformly and projecting the generated samples on $\mathbf{x}$. The rasterizer quickly rejects all samples (gray) outside the rasterized triangles, and in the pixel shader, the remaining samples are tested against ρ to remove outliers (red). The same technique applies in higher dimensions.

This is not fundamentally new, but the stochastic rasterizer gives us an efficient way of sampling the height field $(\mathbf{x}, \rho(\mathbf{x}))$ in up to five dimensions. This allows non-uniform sampling in up to 4D. Figure 12 shows a simple example of a 1D density function, which is lifted to 2D, bounded and tessellated, and sampled by rasterizing the resulting triangles. Note that samples generated using this method have white noise characteristics, as due to the projection, it is difficult to ensure a minimum point distance (i.e., blue noise).

6. Planar Glossy Reflections and Refractions

To render glossy effects, it is necessary to sample the scene in multiple directions from the surface point being shaded. For planar surfaces, this can be implemented as a two-dimensional shear of the reflected/refracted geometry. This has been exploited in previous multi-pass approaches based on accumulation buffering [DB97], or stochastic motion blur rasterization [AMMH07]. Our solution is instead to use the 4D rasterizer to perform the entire 2D shear in a single pass.

For each frame, our method first generates a reflection/refraction map. The map is stochastically rendered with correct occlusion into a multisampled render target and then pre-filtered into a 2D texture, which is used as texture when the scene is rendered from the original viewpoint. For reflections, we set the camera at the reflected position of the primary camera, as illustrated in Figure 13 (left). Alternatively, to compute a refraction map, we choose a single ray from the primary camera and refract it correctly. We then translate the camera, along the refraction plane normal, to intersect this ray. In our implementation, the central ray of the primary camera is used (right in Figure 13).

Our way of translating the camera results in an approximate index of refraction that is correct at the chosen central ray. However, it is progressively less accurate, but still

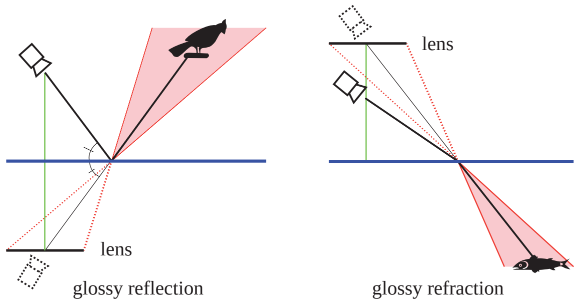


Figure 13: The dotted cameras in the figure are used for rendering the reflection map (left) and the refraction map (right), respectively. The green lines indicate the direction in which the primary camera is translated to find these positions. The frustums that our method sample using 4D rasterization are shown in red for a single texel. Note that the reflecting/refracting plane (blue line) becomes both the focus plane and the near plane.

plausible, toward the edges of the image. For both reflection and refraction, the viewport is chosen to tightly enclose the glossy surface, which is also set as the focus plane. The center of the lens is located at the reflection or refraction camera position as described above, and the lens plane is parallel to the reflection or refraction plane. This gives us a camera setup with an off-center viewport, which can be used with four-dimensional rasterization.

In our implementation, we use a uniform, stratified sampling of a circular camera lens. This results in a visually plausible BRDF, and although it is not physically based, it is possible to control the roughness of the surface by setting the size of the lens. For future work, it would be interesting to examine different BRDFs, either by adjusting the sample distribution or by calculating a weight for each sample. The multi-sampled render target is finally resolved into a texture and no specialized filter is required when sampling. Our reflection and refraction maps are generated in such a way that they map directly to the glossy surface and can, if desired, be weighted together according to the Fresnel term when rendering the surface. Examples can be seen in Figure 14.

7. Motion Blurred Soft Shadow Mapping

We show that a five-dimensional stochastic rasterizer can be set up to generate motion blurred soft shadows through a variation of the shadow mapping algorithm [Wil78]. We note that the (u,v) sample parameters may be used to stochastically vary the sample position on an area light source, as shown in Figure 15. Furthermore, the time parameter may be used to interpolate object motion similar to what is described by Akenine-Möller et al. [AMMH07]. Using the stochastic rasterizer, we can thus create a 5D visibility field as seen from an area light source, i.e., a 5D shadow map. This gives us visibility data similar to what is used by Lehtinen et al. [LAC*11], which needs to be filtered to give a low-noise estimate of visibility for every screen pixel.

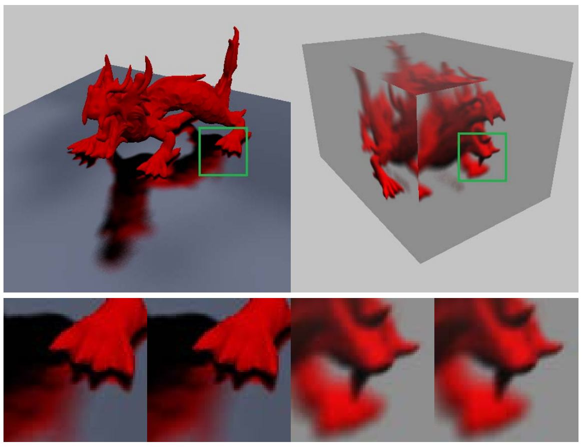


Figure 14: Two renderings with 16 samples per texel for the reflection/refraction maps, using a simple box resolve filter. Left: glossy reflection. Right: glossy refraction. The bottom row compares images with 16 and 64 samples per texel.

As lookups in a 5D data set are inherently difficult (offline algorithms often use 5D kD-trees, for example), we generate a 5D shadow map restricted to a small set of fixed (u,v,t)-samples. This gives results equivalent to accumulation buffering for soft shadows [HA90], but our 5D shadow map is generated in a *single* render pass. The shadow map is queried in the subsequent shading pass, which uses motion blur rasterization with the same set of t parameters. With this setup, the shadow map lookups can be made very efficient. The data structure is an array of 2D shadow maps, which is indexed by t. Thus, in constant time, each sample's t parameter determines which shadow map and corresponding projection matrix to use, and we perform a traditional shadow map lookup using these. This implies that the 5D shadow map needs to be queried per sample for correct motion blurred shadows, as the shadow map lookup depends on the parameters of the samples seen from the camera. An example of soft shadows with motion blur is shown in Figure 1, generated with a set of 64 unique (u,v,t) values.

The most interesting venue for future work lies finding hardware-friendly data structures that allow efficient lookups in higher dimensions. This would enable more well-distributed samples and produce noise instead of the banding artifacts associated with accumulation buffering. We note similarities to the work by Lehtinen et al. [LAC*11], but much work remains before such filtering approaches can be used in the real-time domain. Another interesting area of future work is lossy compression of visibility data to make 5D shadow lookups more efficient at reasonable loss of quality; for instance transforming the visibility data for more efficient lookup by extending the work of Agrawala et al. [ARHM00].

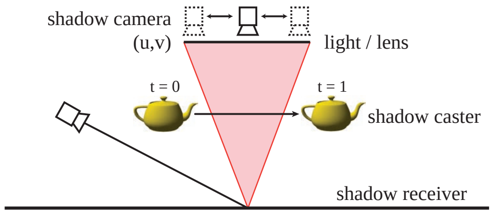


Figure 15: The camera setup used to create a 5D shadow map. The (u,v) parameters position the shadow camera over the area of the light source, and t is used for object motion. For simplicity, we set the focus plane to infinity.

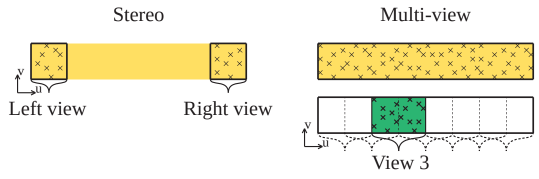


Figure 16: Sampling for multi-view and stereoscopic rendering. The yellow areas show the full extent of the lens. The green area shows a single view's part of the lens.

8. Stochastic Multi-View Rasterization

Stereoscopic and multi-view rendering are seeing widespread use due to the increased availability of 3D display technology. Popular usage areas include films, games, and medical applications. Stereo involves rendering separate images for the left/right eyes, while multi-view displays may need between five and 64 different images [Dod05].

We use a stochastic 5D rasterizer to generate multi-view images with motion blur and depth of field in a *single* pass. This extends Hasselgren and Akenine-Möller's [HAM06] work for static multi-view rendering. Their shading cache is directly applicable, but in our pipeline originally targeting motion blur and depth of field, we use a memoization shading cache [RKLC*11], allowing us to get shader reuse not only between views, but also over time and the entire lens. This is similar to what was suggested by Liktor and Dachsbacher [LD12], and we believe this will be a very important use case for higher-dimensional rasterization.

The viewpoints are located on a *camera line* with small distances in between. To avoid inter-perspective aliasing [ZMDP06], it is important to filter over a region of the camera line when generating a view. With a standard 2D rasterizer, multi-view images with motion blur and depth of field can be generated by rendering a large set of images at different positions on the camera line, lens and time [HA90] and filter them together. This is impractical for many views, has low shading and texture cache coherence, and requires many geometry passes. In contrast, we render all views stochastically in one pass with very high shading reuse.

If we add depth of field to a multi-view rendering setup, the camera line becomes an oblong region/lens. We sample over this region in a single four-dimensional rasterization pass, as seen in Figure 16. The fifth dimension of the stochastic rasterizer is used for motion blur, as usual. For stereoscopic rendering, the samples are divided into two disjoint regions on the lens, which effectively creates two smaller lenses, as can be seen to the left in Figure 16. The images in Figure 17 were rendered using this approach. For more efficient rasterization, the separate lens regions could share the tile test for the v-axis, but perform separate tests for the u-axis.

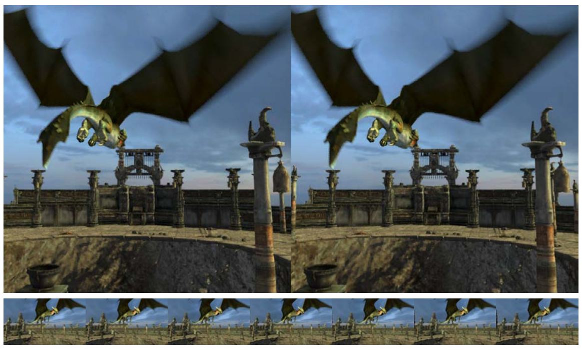


Figure 17: Top: a pair of stereoscopic images with motion/defocus blur rendered using our 5D rasterizer with eight samples per view. These images were constructed for parallel-eye viewing, and hence the stereo effect can be seen when the human viewer focuses at infinity. Bottom: multi-view rendering with seven views, each filtered from a subregion of the lens, in this case using overlapping box filters.

We have implemented stereoscopic and multi-view rendering with motion and defocus blur in our pipeline equipped with a shading cache (see Section 2). Rendering the two stereo views in Figure 17 results in just a 10% increase in shader invocations, compared to rendering a single view only, using a shading cache with 1024 entries. For multi-view rendering with seven views, the total increase in pixel shader invocations is just 15% in this example.

9. Conclusion

We have presented a number of future-looking use cases for higher-dimensional rasterization, as well as several improvements to the pipeline itself, which we hope will help guide future hardware and API design. Our inspiration came from the analogy with the traditional rasterization pipeline, which has been used for a wide range of tasks that were not immediately obvious. We have shown that the same will likely hold true for a stochastic rasterization pipeline.

One of our core insights is that the time and lens bounds provided by recent tile-based traversal algorithms have a much wider applicability than just sample culling. Seeing the moving/defocused triangle as a volumetric primitive and utilizing its extents in the different dimensions, opens up for many interesting use cases. Our second set of applications are more straightforward, but they highlight many practical aspects of the design and present a useful model for thinking about stochastic rasterization as a tool for efficiently rendering a scene from many different viewpoints. Besides the presented applications, it may also be possible to use higher-dimensional rasterization in other related topics, such as computer vision and computational photography.

The next step is to further analyze and improve the most promising applications. We have focused on looking broadly at what *can* be done with a higher-dimensional rasterizer, but detailed simulations are necessary to reveal the true winners. Other than that, the most obvious future work would be to implement the full pipeline in hardware. We believe and hope the graphics research community will continue to explore topics along these lines, in which case our research will have truly long term value.

Acknowledgements The authors thank Aaron Coday, Tom Piazza, Charles Lingle, and Aaron Lefohn at Intel for supporting this research. We also thank Nir Benty and Uzi Sarel for supplying simulation infrastructure, and Manfred Ernst for the caustics scenes, all at Intel. The powerplant scene is from the Microsoft DX SDK. Tomas Akenine-Möller is a Royal Academy of Research Fellow supported by a grant from the Knut and Alice Wallenberg Foundation.

References

[AHAM11] ANDERSSON M., HASSELGREN J., AKENINE-MÖLLER T.: Depth Buffer Compression for Stochastic Motion Blur Rasterization. In *High Performance Graphics* (2011), pp. 127–134. 2

[AMA05] AKENINE-MÖLLER T., AILA T.: Conservative and Tiled Rasterization Using a Modified Triangle Set-Up. *Journal of Graphics Tools, 10*, 3 (2005), 1–8. 3

[AMMH07] AKENINE-MÖLLER T., MUNKBERG J., HASSELGREN J.: Stochastic Rasterization using Time-Continuous Triangles. In *Graphics Hardware* (2007), pp. 7–16. 3, 6, 7, 8, 11

[AMS03] AKENINE-MÖLLER T., STRÖM J.: Graphics for the Masses: A Hardware Rasterization Architecture for Mobile Phones. *ACM Transactions on Graphics, 22*, 3 (2003), 801–808. 3

[AMTMH12] AKENINE-MÖLLER T., TOTH R., MUNKBERG J., HASSELGREN J.: Efficient Depth of Field Rasterization using a Tile Test based on Half-Space Culling. *to appear in Computer Graphics Forum* (2012). 1, 2

[ARHM00] AGRAWALA M., RAMAMOORTHI R., HEIRICH A., MOLL L.: Efficient Image-Based Methods for Rendering Soft Shadows. In *Proceedings of SIGGRAPH* (2000), pp. 375–384. 8

[BFH10] BRUNHAVER J., FATAHALIAN K., HANRAHAN P.: Hardware Implementation of Micropolygon Rasterization with Motion and Defocus Blur. In *High Performance Graphics* (2010), pp. 1–9. 1

[BLF*10] BOULOS S., LUONG E., FATAHALIAN K., MORETON H., HANRAHAN P.: Space-Time Hierarchical Occlusion Culling for Micropolygon Rendering with Motion Blur. In *High Performance Graphics* (2010), pp. 11–18. 3

[BWPP04] BITTNER J., WIMMER M., PIRINGER H., PURGATHOFER W.: Coherent Hierarchical Culling: Hardware Occlusion Queries Made Useful. *Computer Graphics Forum, 23*, 3 (2004), 615–624. 5

[Coo86] COOK R. L.: Stochastic Sampling in Computer Graphics. *ACM Transactions on Graphics, 5*, 1 (1986), 51–72. 7

[Cro77] CROW F. C.: Shadow Algorithms for Computer Graphics. In *Computer Graphics (Proceedings of SIGGRAPH 77)* (1977), vol. 11, pp. 242–248. 1

[DB97] DIEFENBACH P. J., BADLER N. I.: Multi-Pass Pipeline Rendering: Realism For Dynamic Environments. In *Symposium on Interactive 3D Graphics* (1997), pp. 59–70. 7

[Dod05] DODGSON N. A.: Autostereoscopic 3D Displays. *IEEE Computer, 38*, 8 (2005), 31–36. 9

[EAMJ05] ERNST M., AKENINE-MÖLLER T., JENSEN H. W.: Interactive Rendering of Caustics using Interpolated Warped Volumes. In *Graphics Interface* (2005), pp. 87–96. 1, 5

[EDP*11] EBEIDA M. S., DAVIDSON A. A., PATNEY A., KNUPP P. M., MITCHELL S. A., OWENS J. D.: Efficient Maximal Poisson-Disk Sampling. *ACM Transactions on Graphics, 30*, 4 (2011), 49:1–49:12. 6, 7

[FLB*09] FATAHALIAN K., LUONG E., BOULOS S., AKELEY K., MARK W. R., HANRAHAN P.: Data-Parallel Rasterization of Micropolygons with Defocus and Motion Blur. In *High Performance Graphics* (2009), pp. 59–68. 1, 2

[GDAM10] GRIBEL C. J., DOGGETT M., AKENINE-MÖLLER T.: Analytical Motion Blur Rasterization with Compression. In *High Performance Graphics* (2010), pp. 163–172. 2, 5

[GKM93] GREENE N., KASS M., MILLER G.: Hierarchical Z-Buffer Visibility. In *Proceedings of SIGGRAPH 1993* (1993), pp. 231–238. 3

[GLM06] GOVINDARAJU N. K., LIN M. C., MANOCHA D.: Fast and Reliable Collision Culling Using Graphics Hardware. *IEEE Transactions on Visualization and Computer Graphics, 12*, 2 (2006), 143–154. 4

[GM09] GAMITO M. N., MADDOCK S. C.: Accurate Multidimensional Poisson-Disk Sampling. *ACM Transactions on Graphics, 29*, 1 (2009), 8:1–8:19. 6

[GRLM03] GOVINDARAJU N. K., REDON S., LIN M. C., MANOCHA D.: CULLIDE: Interactive Collision Detection between Complex Models in Large Environments using Graphics Hardware. In *Graphics Hardware* (2003), pp. 25–32. 1, 4

[HA90] HAEBERLI P., AKELEY K.: The Accumulation Buffer: Hardware Support for High-Quality Rendering. In *Computer Graphics (Proceedings of SIGGRAPH)* (1990), pp. 309–318. 8, 9

[Hac05] HACHISUKA T.: *GPU Gems 2*. 2005, ch. High-Quality Global Illumination Rendering Using Rasterization, pp. 615–634. 1

[HAM06] HASSELGREN J., AKENINE-MÖLLER T.: An Efficient Multi-View Rasterization Architecture. In *Eurographics Symposium on Rendering* (2006), pp. 61–72. 9

[HKL*99] HOFF III K. E., KEYSER J., LIN M., MANOCHA D., CULVER T.: Fast Computation of Generalized Voronoi Diagrams using Graphics Hardware. In *Proceedings of SIGGRAPH 1999* (1999), pp. 277–286. 1

[IDN02] IWASAKI K., DOBASHI Y., NISHITA T.: An Efficient Method for Rendering Underwater Optical Effects using Graphics Hardware. *Computer Graphics Forum, 21*, 4 (2002), 701–712. 5

[JC98] JENSEN H. W., CHRISTENSEN P.: Efficient Simulation of Light Transport in Scenes with Participating Media using Photon Maps. In *Proceedings of SIGGRAPH* (1998), pp. 311–320. 5

[JK08] JOE S., KUO F. Y.: Constructing Sobol' Sequences with Better Two-Dimensional Projections. *SIAM Journal on Scientific Computing, 30*, 5 (2008), 2635–2654. 2

[KK02] Kollig T., Keller A.: Efficient Multidimensional Sampling. *Computer Graphics Forum, 21*, 3 (2002). 2

[LAC*11] Lehtinen J., Aila T., Chen J., Laine S., Durand F.: Temporal Light Field Reconstruction for Rendering Distribution Effects. *ACM Transactions on Graphics, 30*, 4 (2011), 55:1–55:12. 8

[LAKL11] Laine S., Aila T., Karras T., Lehtinen J.: Clipless Dual-Space Bounds for Faster Stochastic Rasterization. *ACM Transaction on Graphics, 30*, 4 (2011), 106:1–106:6. 1, 2

[LB06] Loop C., Blinn J.: Real-Time GPU Rendering of Piecewise Algebraic Surfaces. *ACM Transactions on Graphics, 25*, 3 (2006), 664–670. 1

[LD11] Liktor G., Dachsbacher C.: Real-Time Volume Caustics with Adaptive Beam Tracing. In *Symposium on Interactive 3D Graphics and Games* (2011), pp. 47–54. 5

[LD12] Liktor G., Dachsbacher C.: Decoupled Deferred Shading for Hardware Rasterization. In *Symposium on Interactive 3D Graphics and Games (to appear)* (2012). 9

[MAM12] Munkberg J., Akenine-Möller T.: Hyperplane Culling for Stochastic Rasterization. In *Eurographics Short Papers Proceedings (to appear)* (2012). 1, 2

[MCH*11] Munkberg J., Clarberg P., Hasselgren J., Toth R., Sugihara M., Akenine-Möller T.: Hierarchical Stochastic Motion Blur Rasterization. In *High Performance Graphics* (2011), pp. 107–118. 1, 2, 11

[Moo92] Moore D.: *Understanding Simploids*. Graphics Gems III. 1992, pp. 250–255. 6

[PH10] Pharr M., Humphreys G.: *Physically Based Rendering: From Theory to Implementation*, 2nd ed. Morgan Kaufmann, 2010. 6

[RAC02] Redon S., Abderrahmane, Coquillart S.: Fast Continuous Collision Detection between Rigid Bodies. *Computer Graphics Forum, 21*, 3 (2002), 279–287. 3

[RKLC*11] Ragan-Kelley J., Lehtinen J., Chen J., Doggett M., Durand F.: Decoupled Sampling for Graphics Pipelines. *ACM Transactions on Graphics, 30*, 3 (2011), 17:1–17:17. 1, 3, 9

[SA11] Segal M., Akeley K.: The OpenGL Graphics System: A Specification, v 4.2, August 2011. 3

[SF91] Shinya M., Forgue M.-C.: Interference Detection through Rasterization. *The Journal of Visualization and Computer Animation, 2*, 4 (1991), 132–134. 4

[SKP07] Shah M., Konttinen J., Pattanaik S.: Caustics Mapping: An Image-space Technique for Real-time Caustics. *IEEE Transactions on Visualization and Computer Graphics, 13*, 2 (2007), 272–280. 5

[Wat90] Watt M.: Light-Water Interaction using Backward Beam Tracing. In *Proceedings of SIGGRAPH 1990* (1990), pp. 377–385. 5

[Wei08] Wei L.-Y.: Parallel Poisson Disk Sampling. *ACM Transactions on Graphics, 27*, 3 (2008), 20:1–20:9. 6, 7

[Wie96] Wiegand T. F.: Interactive Rendering of CSG Models. *Computer Graphics Forum, 15*, 4 (1996), 249–261. 1

[Wil78] Williams L.: Casting Curved Shadows on Curved Surfaces. In *Computer Graphics (Proceedings of SIGGRAPH 78)* (1978), pp. 270–274. 1, 8

[ZMDP06] Zwicker M., Matusik W., Durand F., Pfister H.: Antialiasing for Automultiscopic 3D Displays. In *Eurographics Symposium on Rendering* (2006), pp. 73–82. 9

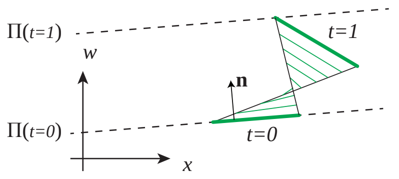


Figure 18: A conservative maximum depth bound from our second test is expressed as a plane equation $\Pi(t)$ that moves linearly in time, which has larger (or equal) depth than the triangle (as seen from the camera) at all times, $t \in [0,1]$.

Appendix A: Conservative Depth Computations

For efficient $z_{\min}/z_{\max}$-culling, it is important to conservatively estimate the min/max depth of a triangle inside a tile. This is straightforward for static triangles, but considerably harder for motion blurred triangles. The only known method [AMMH07] reverts to using min/max of vertices when the triangle changes facing during $t \in [0,1]$, for example. This leads to poor bounds. In this discussion, the vertices at $t=0$ are denoted $\mathbf{q}_0\mathbf{q}_1\mathbf{q}_2$, and $\mathbf{r}_0\mathbf{r}_1\mathbf{r}_2$ at $t=1$. Our approach is based on two types of tests, which gives tight estimates of minimum and maximum depth over a motion blurred triangle inside a tile. The first test is similar to the moving bounding box around the triangle [MCH*11], except that for the max depth computation, we only use the farthest rectangle of the box, and vice versa.

The idea of the second test is to compute a plane equation through the triangle at time $t=0$ ($t=1$), and move that linearly in t along the plane's normal direction, such that it always has the moving triangle on one side of the plane. This is illustrated in Figure 18. For example, the moving plane equation for maximum depth (based on the triangle at $t=0$), can be constructed as follows. The normal, $\mathbf{n}$, of the triangle at $t=0$ is computed, and negated if $n_w < 0$ in order to ensure that it moves in the right direction. The plane equation is then $\mathbf{n}\cdot\mathbf{x}+d=0$, where $d=-\mathbf{n}\cdot\mathbf{q}_0$, and $\mathbf{x}$ is any point on the plane. Next, we compute the maximum signed distance from the plane over the vertices of the moving triangle at $t=1$:

$$v = -\max_i(\mathbf{n}\cdot\mathbf{r}_i+d), \quad i \in \{0,1,2\}. \tag{1}$$

At this point, we have created a moving plane equation, which moves in the direction of the plane normal: $\Pi(t): \mathbf{n}\cdot\mathbf{x}+d+vt=0$. As can be seen, we have added the term vt, which ensures (by construction) that the triangle is "below" the moving plane, $\Pi(t)$. If $\Pi(t)$ does not sweep through the camera origin, e.g., $d+vt \neq 0, \forall t \in [0,1]$, then $\Pi(t)$ has greater depth than the moving triangle at all times, $t \in [0,1]$. Similarly, a moving plane equation can be derived from the triangle normal at $t=1$.

For each moving plane equation, the maximum depth is computed over a tile by evaluating the plane equation at the appropriate tile corner using the time interval, $\hat{t}$, from the tile test. The minimum of these two evaluations and the maximum depth from the first test is used as a conservative maximum depth over the tile. Similar computations are done for evaluating the minimum depth.

This approach can also be extended to handle depth of field. Briefly, the depth estimations need to be done at the four corners of the two-dimensional bounding box of the lens shape, and similar to before, the second test has to be disabled if the plane sweeps over any part of the lens (and not only a single camera point).