To appear in ACM TOG 30(4).

Clipless Dual-Space Bounds for Faster Stochastic Rasterization

Samuli Laine Timo Aila Tero Karras Jaakko Lehtinen

NVIDIA Research*

Abstract

We present a novel method for increasing the efficiency of stochastic rasterization of motion and defocus blur. Contrary to earlier approaches, our method is efficient even with the low sampling densities commonly encountered in realtime rendering, while allowing the use of arbitrary sampling patterns for maximal image quality. Our clipless dual-space formulation avoids problems with triangles that cross the camera plane during the shutter interval. The method is also simple to plug into existing rendering systems.

CR Categories: I.3.3 [Computer Graphics]: Picture/Image Generation—Display algorithms;

Keywords: rasterization, stochastic, temporal bounds, dual space

1 Introduction

Traditional rasterization assumes a pinhole camera and an infinitely fast shutter. These assumptions cause the rendered image to lack two effects that are encountered in real-world images: motion blur and defocus blur. Motion blur is caused by visible objects moving relative to the camera during the time when the shutter is open, whereas defocus blur is caused by different points of the camera's lens seeing different views of the scene.

Most realtime rendering algorithms use point sampling on the screen and average the resulting colors to get pixel colors. In stochastic rasterization, we associate time (t) and lens position (u, v) with each sample. The resulting average implicitly accounts for motion and defocus blur.

In a traditional (non-hierarchical, non-stochastic) rasterizer, one would find the 2D bounding box of the triangle on the screen and test all samples within it. With this approach, the number of inside/outside tests is typically a few times larger than the number of samples actually covered by the triangle. This brings us to an important predictor for rasterization performance: sample test efficiency (STE), which is defined as the number of samples covered divided by the number of samples tested [Fatahalian et al. 2009].

In a stochastic rasterizer, achieving good STE has been one of the most elusive goals. If we just find the bounding box of the triangle in screen space, that does not guarantee high STE. Consider the example of a triangle that covers only one sample. If we add motion blur and allow the triangle to move from one image corner to another, it will still cover only approximately one sample, but its 2D bounding box is the whole screen. The sample test of a stochastic rasterizer is also several times more expensive than in traditional rasterization, further amplifying the performance impact of STE. Interestingly, stochastic rasterization does not appreciably increase the cost of shading [Cook et al. 1987; Ragan-Kelley et al. 2011].

Our goal is to make STE less dependent on the amount of motion or defocus blur, especially in the context of relatively low sampling densities of realtime rendering, where the current offline approaches [Cook et al. 1990] are inefficient. Contrarily to some recent approaches that improve STE by restricting to specific (lower quality) sampling patterns [Fatalian et al. 2009], our work focuses on achieving high STE without sacrificing quality, allowing the use of arbitrary sample sets. We achieve this by computing u, v and t bounds for a pixel or a tile of pixels during rasterization, and then avoiding the processing of samples that fall outside the computed bounds.

We assume that the motion of the vertices is linear in world space, and that the depth of field effect is physically based. We perform the computation of (u, v) bounds after projecting the vertices to screen space. This is feasible because the apparent screen-space motion of a vertex is always affine with respect to lens coordinates. For t bounds, motion along z makes the situation much more complicated because linear motion in 3D is mapped to non-linear motion on the screen, and camera plane crossings cause singularities. To avoid these problems, we instead work in a linear dual space where the operations required for computing the bounds are linear and straightforward, and the lack of perspective division ensures that there are no singularities.

Our algorithm has the following desirable properties:

- Clipless dual-space formulation avoids problems with triangles that cross the camera plane during the shutter interval.
- High STE is achieved for arbitrary sampling patterns.
- Unlike in previous methods, each primitive is set up exactly once.

2 Previous Methods

Akenine-Möller et al. [2007] describe a practical stochastic rasterization algorithm for rendering triangles with motion blur. They fit an oriented bounding box that contains the triangle during the exposure time, i.e., $t \in [0, 1]$, enumerate the pixels that the OBB overlaps using hardware rasterization, and finally test each sample against the moving triangle. This basic structure can be found in other stochastic rasterization methods as well. McGuire et al. [2010] enumerate the pixels in a 2D convex hull formed by the six vertices of a triangle at $t = 0$ and $t = 1$, which covers the projection of the triangle at all t given that the triangle stays in front of camera plane. Near clipping is handled as a complicated special case.

Pixar Let us consider a simple example: a pixel-sized quad moves X pixels during the exposure, and our frame buffer has S stratified samples per pixel. A brute force algorithm (or indeed [Akenine-Möller et al. 2007] and [McGuire et al. 2010]) would test all of the $S \times (X + 1)$ samples. Cook et al. [1990] improve this by processing the exposure time in multiple parts. For example, if we split it into two equally long parts, each covers $\frac{X}{2} + 1$ pixels and we need $\frac{S}{2}$ sample tests per pixel, thus the total amount of work reduces to $S \times (\frac{X}{2} + 1)$. Further optimization is achieved by splitting the exposure more densely, up to S parts.

In the general case, Cook et al. subdivide the (u, v) and t domains into N strata, pair these to form (u, v, t) strata, compute a screen-space bounding box for each stratum, and test all samples that are covered by the bounding box and whose (u, v, t) coordinates are within the stratum. At low sampling densities few splits can be

*e-mail: {slaine,taila,tkarras,jlehtinen}@nvidia.com

InterleaveUVT 4 spp | Sobol sampling 4 spp | InterleaveUVT 16 spp | Sobol sampling 16 spp

Figure 1: *Quality comparison between InterleaveUVT using 2×2 pixel tiles and a good sampling pattern generated using Sobol sequences. InterleaveUVT suffers from a limited number of unique (u, v, t) triplets, especially with low sample counts common in realtime rendering. The test scene has both motion and defocus blur.*

afforded and STE remains low, but efficiency improves when sampling density increases.

As Hou et al. [2010] point out, the pairing between lens and time strata imposes constraints on the sampling pattern, and therefore creates correlation between defocus and motion blur. This can lead to aliasing when both effects are combined. To make the method better suited as a comparison method, we modify it slightly by removing the pairing restriction. Instead, our *modified Pixar* algorithm subdivides the (u, v, t) space into $N_1 \times N_2 \times N_3$ strata, allowing the entire space to be sampled using arbitrary sampling patterns.

InterleaveUVT Fatahalian et al. [2009] discretize the (u, v, t) domain into N unique triplets, and perform traditional 2D rasterization for each. The major improvement over accumulation buffer [Haeberli and Akeley 1990] is the use of high-quality interleaved sampling patterns. Instead of having all of the N triplets in every pixel, as in the accumulation buffer, Fatahalian et al. organize the frame buffer so that each tile (e.g. 2×2 pixels) contains all N triplets. These are carefully assigned to (x, y) sampling positions so that each tile can use a different permutation. This converts most "ghosting" artifacts of the accumulation buffer into high-frequency noise and the perceived image quality is vastly improved.

Because each (u, v, t) triplet is processed separately using 2D rasterization, the STE of InterleaveUVT is fairly high. However, the bounding boxes of triangles need to be snapped to full tiles (due to permutations), and this reduces the efficiency with small triangles. Also, the fixed number of unique (u, v, t) triplets can reduce image quality especially for concurrent motion and defocus, as illustrated in Figure 1. Neither the modified Pixar method nor our new algorithm have this limitation. In a realtime rendering context each unique triplet implies a separate rasterization pass.

3 Our Algorithm for Bounding (*u*,*v*,*t*)

To illustrate our basic idea, let us consider a small triangle moving horizontally across 10 pixels during a frame with no defocus blur. As an example, the triangle covers the 7th pixel for approximately $t \in [0.6, 0.7]$ and all of the ∼90% of the pixel's samples outside that interval can be trivially rejected. This general idea works for less trivial motion and for any screen rectangle. It also applies to (u, v) intervals on the lens. In this section we develop the details of making the basic idea practical.

To render a triangle, we process the tiles inside its screen-space bounds, determine the t and (u, v) intervals for each tile, and then test all samples within the tile that fall within those intervals. If any of the intervals is empty, the tile can be discarded. Highest STE is obtained with tile size of 1×1 pixels, but tiles can also be made larger to decrease the number of per-tile calculations. These trade-offs are discussed in Section 4.1. For hardware implementation, we envision that the interval tests would be carried out by the hardware rasterizer that already produces a sparse set of covered samples that are compacted afterwards.

In the following, we start by deriving the intervals for an infinitesimally small screen point, followed by a generalization to finite tile size. We first explain our approach in the context of defocus blur, and then consider motion blur. Our input consists of clip-space vertex positions[1] (x, y, w) for $t = 0$ and $t = 1$, as well as the signed radii of circles of confusion (CoC), also expressed in clip space. These determine how the positions of the vertices change from their lens-center locations as we change the lens coordinates. In a physically-based setting, the signed CoC has an affine dependence on depth w in clip space, i.e., $\mathrm{CoC}(w) = aw + b$.

3.1 Bounds for (*u*,*v*)

The computation of u and v bounds is relatively simple because, for vertices that are in front of the camera plane, we can apply the perspective division and work with screen-space coordinates which we denote $\hat{x}$ and $\hat{y}$. The apparent motion of a screen-space vertex $(\hat{x}, \hat{y})$ is always affine with respect to lens coordinates, because the depth of the vertex, i.e., its distance from the camera plane, is constant over the lens.

We parameterize the lens by $(u, v) \in [0, 1]^2$. As the first operation we calculate two screen-space bounding boxes that cover all six input vertices (three for $t = 0$ and three for $t = 1$), one for the minimum lens corner with coordinates $(0, 0)$ and one for the maximum corner at $(1, 1)$. The bounding box of the union of these two boxes is the set of pixels that need to be processed during rasterization.

Now, because the apparent motion of vertex is linear with respect to lens coordinates, we can synthesize a conservatively correct bounding box for any given lens coordinates by linearly interpolating between these two bounding boxes. The $\hat{x}$ coordinates of the bounding boxes depend only on u, and similarly for $\hat{y}$ and v, and therefore all calculations can be performed separately for the two axes.

Let us first consider the calculation of u bounds for an infinitesimal screen-space point at $\hat{x}$. We will denote the horizontal bounding box extents for $u = 0$ as $[\hat{x}^{u=0}_{\min}, \hat{x}^{u=0}_{\max}]$ and similarly for $u = 1$. The interpolated bounding box's minimum $\hat{x}$ extent at a given u is

$$\hat{x}^{u}_{\min} = \hat{x}^{u=0}_{\min} + u(\hat{x}^{u=1}_{\min} - \hat{x}^{u=0}_{\min}), \tag{1}$$

from which we can easily solve the u' for which $\hat{x}^{u}_{\min} = \hat{x}$:

$$u' = (\hat{x} - \hat{x}^{u=0}_{\min})/(\hat{x}^{u=1}_{\min} - \hat{x}^{u=0}_{\min}). \tag{2}$$

The sign of the denominator is the direction of movement with respect to u. If positive, the range of u is limited to $[-\infty, u']$, and otherwise to $[u', \infty]$. We can repeat Eq. 2 for the maximum $\hat{x}$ extent to derive another range. The intersection of the two ranges tells for which u the bounding box overlaps $\hat{x}$. If the resulting range does not overlap the lens domain $[0, 1]$, we can skip the entire tile, and otherwise we can limit the testing to samples within the range.

To extend from an infinitesimal $\hat{x}$ to a finite tile size, we simply enlarge the two initial bounding boxes by the half-length of the tile, and let $\hat{x}$ denote the tile's center.

Since we typically compute bounds for many tiles, it is beneficial to rewrite Eq. 2 as $u' = a\hat{x} - b$, where a is the reciprocal of the denominator and $b = a\hat{x}^{u=0}_{\min}$. The *per-tile* cost is then 4×1 fused

[1] After the viewing and perspective transforms have been applied but before the perspective division.

multiply-adds (FMA) for (u, v) bounds plus a few MIN/MAX operations, which is cheap compared to the $\sim$25 FMAs required by each sample test.

Vertices behind camera plane If some of the vertices are behind the camera plane ($w < 0$), we need to detect on which sides of the camera the moving triangle may cross the plane, and extend the bounding boxes to the respective screen edges.

We examine (x, w) and (y, w) separately in clip space, and the following explanation focuses on (x, w), cf. Figure 2. If all of the six vertices extended with their respective CoCs are on one side of the camera (left or right), the solution is trivial. In more complicated cases we use a separating line algorithm: If all of the extended vertices are on the same side of some line that goes through the camera point, the moving triangle cannot intersect the camera plane on the opposite side.

In practice, to determine whether a moving triangle intersects the camera plane on the right, we consider the six vertices extended rightward according to their CoCs. We start with a separating line candidate that goes through one of the vertices and the camera point. Then we loop over the remaining points once, and for each point we test if it is on the same side of the candidate as $(\infty, 0)$. If so, the candidate is updated, and we test if the point that defined the previous candidate is now on the same side as $(\infty, 0)$. If this happens, there is no separating line. Otherwise, if all points have been processed once, we have found a separating line and consequently there is no intersection on the right.

3.2 Bounds for *t*

Based on extensive attempts to derive sensible formulae for t bounds based on screen-space projections of vertices, we believe that the perspective division should be avoided for two reasons. First, linear motion of points in 3D translates to non-linear motion on the screen, making it necessary to use rational expressions for tight $\hat{x}\hat{y}$ bounds. This would increase the cost and complexity of determining the t bounds. Second, the division produces meaningful results only for points that are in front of camera plane, and cases where $w < 0$ need to be handled separately, including the cases where the sign of w changes during the shutter interval. We avoid these problems by performing the computations in a linear dual space (slope-intercept space).

In the following, we consider only the x axis, and hence our input vertices are in the two-dimensional xw clip space. The y axis is handled separately, in the same fashion. We will initially ignore the CoCs, and as with (u, v) bounds, we begin by calculating bounds for infinitesimal screen-space points.

Dual space Our key insight is replacing the perspective projection by an orthogonal projection parameterized by view direction $\gamma \in [-1, 1]$. Figure 3a illustrates the operation in clip space for a single γ. A sightline from the camera towards direction γ can intersect a triangle only if the corresponding orthogonal projection of the triangle onto $w = 0$ includes the camera point. The projected coordinate δ of a point (x, w) is given by

$$\delta = x - w\gamma \tag{3}$$

and for overlap to occur we need $\delta_{\min} < 0 < \delta_{\max}$, where $\delta_{\min}$ and $\delta_{\max}$ are the minimum and maximum δ obtained for the vertices of the triangle.

Figure 3b shows the same situation in dual space spanned by γ and δ. From Eq. 3 it is apparent that points in clip space are mapped to linear functions in dual space. Conversely, points in dual space are mapped to lines in clip space, and in particular, the clip-space lines

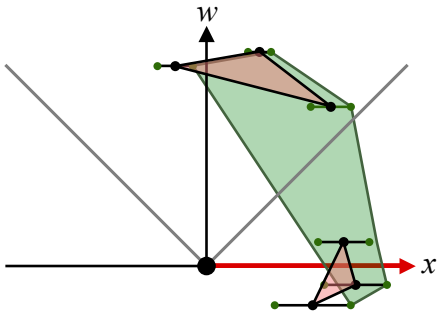

Figure 2: *Handling vertices behind the camera plane when calculating (u, v) bounds. Camera plane is the horizontal line at the bottom, and frustum side planes are indicated by the diagonal lines. In this example, the 2D convex hull of the six input vertices, extended with their respective CoCs to the right, intersects the half-line from camera towards $x = +\infty$, which is detected using a separating line algorithm.*

that go through the camera are mapped to points on the γ axis in dual space. Therefore we can figure out which directions, e.g., the $t = 0$ triangle covers by determining which points on the γ axis are covered (lower shaded region in Figure 3b).

Let us now consider the intermediate t values. First, let us fix some γ and consider the δ coordinate of a vertex that moves linearly in clip space as t changes from 0 to 1. From Eq. 3 we can see that linear motion in x and w produces linear change in δ; the linearity is also apparent from Figure 3a. The δ range spanned by the three vertices of a triangle can thus be conservatively estimated at any t by linearly interpolating $\delta_{\min}$ and $\delta_{\max}$ according to t, i.e., $\delta^t_{\min} = \delta^{t=0}_{\min} + t(\delta^{t=1}_{\min} - \delta^{t=0}_{\min})$.

There is still the problem that $\delta_{\min}$ and $\delta_{\max}$ are rather complex piecewise-linear functions of γ. To avoid storing them as-is, we find dual-space lines that bound the actual region, indicated as dashed lines in Figure 3b, by computing $\delta_{\min}$ and $\delta_{\max}$ at $\gamma = -1$ and $\gamma = 1$. If we mapped these lines to points in (x, w), they would correspond to points that stay on the left or right side of the triangle at all t, when viewed using any $\gamma \in [-1, 1]$.

We now have everything we need for computing the t bounds for a given screen-space point at $\hat{x}$. First we obtain γ by scaling and translating $\hat{x}$ to range $[-1, 1]$. Based on this γ, we evaluate the $\delta_{\min}$ and $\delta_{\max}$ for $t = 0$ and $t = 1$ from the four bounding lines. Because $\delta_{\min}$ and $\delta_{\max}$ can be conservatively estimated for any t using linear interpolation, we can transform the requirement that $\delta^t_{\min} < 0 < \delta^t_{\max}$ to

$$\delta^{t=0}_{\min} + t(\delta^{t=1}_{\min} - \delta^{t=0}_{\min}) < 0 < \delta^{t=0}_{\max} + t(\delta^{t=1}_{\max} - \delta^{t=0}_{\max}), \tag{4}$$

from which the t bounds can be solved analogously to how the (u, v) bounds were calculated. However, this time we cannot precalculate the reciprocal as its value depends on γ.

Extensions and optimizations We take non-zero CoCs into account by biasing the x coordinates of vertices into the direction that is appropriate for maintaining conservativeness, e.g., when calculating $\delta_{\min}$ we bias x towards the negative direction regardless of the sign of the CoC radius. Tile extents are handled similarly, by further increasing the bias by half of the tile extents in clip space (obtained by multiplication by w). This way, during rasterization we can evaluate the t bounds at the tile center and obtain a range that is valid for all points in the tile.

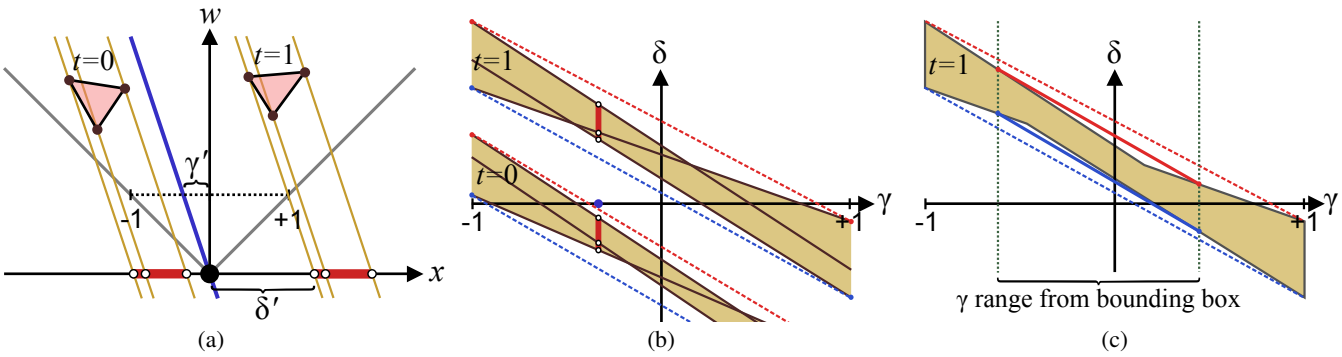

Figure 3: *(a) Relationship between (x, w) clip space and (γ, δ) dual space. Triangle at $t = 0$ and $t = 1$ is projected onto $w = 0$ line along direction γ'. The positions of the projected points are denoted δ'. Extents $[\delta_{\min}, \delta_{\max}]$ for $t = 0$ and $t = 1$ are shown with bold red. (b) In dual space, the δ of each vertex traces a linear function of γ. Projection direction γ' is shown as a blue dot and the corresponding δ ranges are highlighted. The regions covered by the triangle at $t = 0$ and $t = 1$ are shaded, and the dashed lines bound these regions. (c) By restricting our interest to a limited γ range, a tighter fit can be obtained (solid lines). Only $t = 1$ is shown for clarity.*

Since we will need the t bounds only for pixels that the rasterizer processes, we can safely limit the γ range to the screen-space bounding box computed in Section 3.1, thus allowing a tighter fit (Figure 3c). Furthermore, we can neglect the γ where $\delta_{\min}$ is positive for both $t = 0$ and $t = 1$, because the dual space point cannot be covered for any $t \in [0, 1]$, and similarly for γ where $\delta_{\max}$ is guaranteed to remain negative. The sign of δ still needs to be conservatively correct, so we employ this latter optimization only when all vertices have $w > 0$, in which case the correctness of the sign automatically follows from $\delta_{\min}$ and $\delta_{\max}$ being monotonic.

As vertices in front of and behind the camera are not distinguished in the calculations, it is possible that a triangle that moves behind the camera could have nonzero t bounds for superfluous pixels, thus reducing efficiency. As an optimization, we reduce the t range from $[0, 1]$ to the time span the triangle actually spends, even partially, inside the frustum. This range is calculated by computing the triangle extents at $t = 0$ and $t = 1$ with respect to the frustum side planes and the camera plane and solving for the t range where the triangle is inside all planes.

We have only considered t bounds computed for the x axis so far. For the y axis we obtain similar bounds, and the final t bounds are found by intersecting these two along with the above t range. If motion is diagonal on screen, the intersection of the t bounds obtained for the x and y axes is empty outside the area traced by the triangle. Therefore diagonal motion is handled as efficiently as axis-aligned motion.

4 Results

We compare against the modified Pixar method and a brute-force approach that tests all samples within a bounding box that covers the triangle for all (u, v, t). We call the latter "Bbox scan". We also run the tests using InterleaveUVT, although it does not produce the same images. Two of our test scenes were constructed from DirectX captures from Age of Conan PC MMO (Funcom) and Assassin's Creed (Ubisoft), and have an average triangle area of about 40 pixels. The third test scene contains several highly tessellated car models, and with an average triangle area of 0.4 pixels it has the kind of statistics one might see in today's film content. To introduce motion blur, we added either camera or object motion, and for defocus blur we increased camera aperture size and chose a suitable focus distance.

Figure 4 shows our test scenes under motion and defocus. The sam-

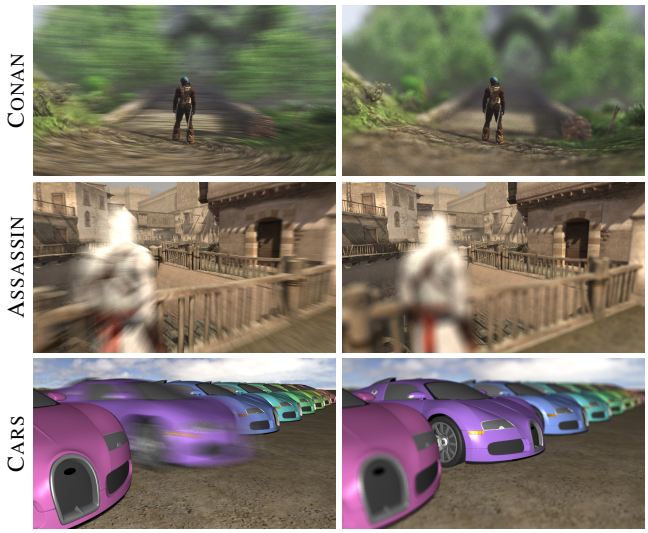

Figure 4: *Test scenes rendered with motion blur (left) and defocus blur (right). Output resolution was 960×540 with 16 spp.*

ple coordinates were generated using a 5D Sobol sequence [Joe and Kuo 2008], and the tests were done using 16 samples per pixel (spp) for brute force and our method. We used a tile size of one pixel. The number of samples has no effect on STE of brute force and our method, but it does affect the STE of the modified Pixar method that was therefore evaluated using three sample counts. The strata subdivision was performed as follows. when only motion or depth of field present, we subdivided only in t or (u, v), respectively. In the combined case we subdivided (u, v, t) into 2×2×4 strata for 16 samples per pixel, and 4×4×4 for 64 samples. For 4 samples, we used 2×2×1 subdivision, as it resulted in higher STE than 1×1×4.

The results are summarized in Table 1. The STE of our method was practically immune to motion or defocus, when only one effect was present, and the benefit over prior art increased with scene complexity, peaking in the highly defocused CARS at 160× brute force and 15× modified Pixar at 16 spp. When both effects are strong in the same parts of the scene, the efficiency of all three methods is reduced; CONAN shows an extreme example of strong camera motion and defocus. Doubling the magnitude of motion or defocus decreased the STE of comparison methods significantly, but had

Scene	Bbox scan	modified Pixar 4	modified Pixar 16	modified Pixar 64	New method
CONAN	23.6	23.6	23.6	23.6	23.6
motion	2.7	9.5	17.7	21.9	23.7
motion ×2	1.3	6.0	14.6	20.9	24.0
defocus	1.7	4.4	8.8	13.9	23.1
defocus ×2	0.7	1.8	4.4	8.8	21.9
both	0.7	1.2	2.6	4.3	5.6
both ×2	0.4	0.6	1.1	2.0	2.9
ASSASSIN	23.2	23.2	23.2	23.2	23.2
motion	10.8	19.5	22.4	23.1	23.4
motion ×2	4.3	14.8	21.2	23.0	23.6
defocus	9.5	15.1	19.0	21.1	23.2
defocus ×2	4.3	9.5	15.1	19.0	23.0
both	4.5	6.8	7.2	10.1	14.1
both ×2	1.3	2.1	3.9	6.7	6.9
CARS	8.59	8.60	8.59	8.60	8.59
motion	0.50	2.97	6.43	8.02	8.63
motion ×2	0.14	1.27	4.70	7.40	8.69
defocus	0.19	0.59	1.53	3.13	8.57
defocus ×2	0.05	0.19	0.59	1.53	8.51
both	0.12	0.25	0.49	1.08	4.51
both ×2	0.03	0.07	0.16	0.39	2.42

Table 1: *STE results as percentages (higher is better). Rows with scene name refer to static case without motion or depth of field. ×2 means the shutter time was doubled, and in case of defocus the aperture radius was doubled.*

virtually no effect to our STE except in the combined case. As expected, the STE of InterleaveUVT was approximately constant regardless of motion or defocus, around 21% in the game scenes, but only 3.8% in CARS. We emphasize again that it creates lower quality images than the other methods, and a higher sample budget would be needed for comparable quality.

4.1 Further Analysis

We have restricted our analysis to STE, which is an important factor in stochastic rasterization efficiency. However, when the screen footprint of a triangle is heavily enlarged by the stochastic effects, the per-tile computations will eventually start to dominate the overall performance. In practice, it may therefore be beneficial to employ techniques from previous work [Akenine-Möller et al. 2007; McGuire et al. 2010] to construct a tighter 2D footprint and hence reduce the number of tiles that need to be touched.

Assuming that the per-sample bound tests are carried out by the hardware rasterizer with negligible cost, the remaining workload can be divided into per-primitive setup, per-tile bound computation, and per-sample visibility test. In the Pixar method, the setup cost is roughly 100 operations (multiply-add, compare, etc.), but it has to be repeated for each (u, v, t) stratum. Similarly, InterleaveUVT needs about 50 operations to setup the triangle for one (u, v, t) tuple. With 16 samples per pixel and a 2×2 pixel tile, this amounts to a total of 1600 and 3200 operations per primitive for the two methods, respectively. Our method performs a setup of roughly 500 operations exactly once per primitive regardless of spp or tile size. With 16 spp, the cost is significantly lower than with the comparison methods. With 4 spp, all three methods are roughly equal in setup cost.

In our method, the total cost of computing (u, v, t) bounds for one tile is about 40 operations. In cases where we have to perform multiple visibility tests per tile, the cost is comparably small. However, in the CARS scene with defocus, the primitives are tiny enough compared to the amount of blur to make the two costs roughly equal. This bottleneck can be countered to a large extent by increasing tile size to 2×2 pixels, which offered the best tradeoff in terms of operation counts in all of our test scenes. Increasing the tile size decreased the STE by 5–10% in the game scenes and by 16–55% in CARS. However, the decreased STE was more than compensated by the reduced amount of per-tile work, and the overall operation count was decreased by 23–54% in CONAN, by 5–19% in ASSASSIN, and by 26–57% in CARS.

It should be kept in mind that in a real hardware implementation the actual area and power cost of each of the stages is not perfectly correlated with the number of arithmetic operations, and hence the above estimates based on operation counts should not be taken too literally. For example, if the calculation of per-tile bounds were performed in fixed-function hardware, its comparative cost might be diminished enough to make increasing the tile size unnecessary for all practical situations.

5 Discussion

One could consider using screen-affine motion as a simpler approximation to perspective motion. We believe that this would provide acceptable quality in most cases, and in fact started out by trying to formulate our algorithm for screen-linear motion. Unfortunately, clipping is a major problem, as the post projective vertex position is undefined if the vertex lies on or behind the camera plane. We initially implemented a time-continuous clipper that is executed before projecting the vertices, but this approach has the problem—in addition to its inherent complexity—that the clip vertices of moving edges follow quadratic rational instead of linear rational trajectories. This requires either significantly complicating the bound calculations, or approximating the clip vertex motion by linear motion, which in turn can produce arbitrarily large errors in the image. Despite our efforts, we were unable to find a robust solution for screen-linear motion approximation.

Our STE is practically immune to the amount of defocus or motion blur, but the combined case is more difficult. The root cause is that in this case both time and lens coordinates affect whether a primitive covers a given screen-space point. The covered samples in such pixels lie inside an oblique region in (u, t) and (v, t) spaces, whereas with just one effect they fall approximately inside axis-aligned slabs. The oblique region cannot be perfectly bounded by lines oriented along u, v, or t axes, which explains the reduced STE in our method. Interestingly, the image quality issues in InterleaveUVT and the original Pixar method are related to this phenomenon. Both employ sparse sampling of the (u, v, t) space, but still guarantee that projections to t and (u, v) are individually well stratified. When only one effect is present and the covered region is an axis-aligned slab, the well-stratified projections ensure good results, but when both effects are present and the covered region is oblique, the sparseness of the joint distribution yields banding artifacts.

Our method combines the high STE of InterleaveUVT with the free positioning of samples. Unlike the comparison methods, we set up and process every triangle exactly once regardless of the number of samples. This makes our method comparatively easy to integrate into existing rendering systems, and also makes it a viable candidate for hardware acceleration of stochastic rasterization. For future work, we hypothesize that the efficiency of simultaneous motion and defocus blur could be further improved by constructing non-axis-aligned bounds in (u, v, t) space.

Acknowledgments

We thank Funcom for the permission to use a test scene from Age of Conan PC MMO and Ubisoft for the permission to use a test scene from Assassin's Creed. Kayvon Fatahalian and Solomon Boulos provided helpful insights into previous work. We also thank Peter Shirley, Eric Enderton and Jacopo Pantaleoni for discussions and feedback, and the anonymous reviewers for corrections and suggestions for improvement.

References

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

COOK, R. L., CARPENTER, L., AND CATMULL, E. 1987. The Reyes image rendering architecture. In *Computer Graphics (Proc. ACM SIGGRAPH 87)*, vol. 21, 95–102.

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

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 *Proc. High Performance Graphics*, 59–68.

HAEBERLI, P., AND AKELEY, K. 1990. The accumulation buffer: hardware support for high-quality rendering. In *Proc. ACM SIGGRAPH*, 309–318.

HOU, Q., QIN, H., LI, W., GUO, B., AND ZHOU, K. 2010. Micropolygon ray tracing with defocus and motion blur. *ACM Transactions on Graphics 29*, 4, 64:1–64:10.

JOE, S., AND KUO, F. Y. 2008. Constructing Sobol sequences with better two-dimensional projections. *SIAM J. Sci. Comput. 30*, 2635–2654.

MCGUIRE, M., ENDERTON, E., SHIRLEY, P., AND LUEBKE, D. 2010. Real-time stochastic rasterization on conventional GPU architectures. In *Proc. High Performance Graphics*, 173–182.

RAGAN-KELLEY, J., LEHTINEN, J., CHEN, J., DOGGETT, M., AND DURAND, F. 2011. Decoupled sampling for graphics pipelines. *ACM Transactions on Graphics 30*, 3.