Theory and Analysis of Higher-Order Motion Blur Rasterization

Carl Johan Gribel[1] Jacob Munkberg[2] Jon Hasselgren[2] Tomas Akenine-Möller[1,2]

[1]Lund University and [2]Intel Corporation

Abstract

A common assumption in motion blur rendering is that the triangle vertices move in straight lines. In this paper, we focus on scenarios where this assumption is no longer valid, such as motion due to fast rotation and other non-linear characteristics. To that end, we present a higher-order representation of vertex motion based on Bézier curves, which allows for more complex motion paths, and we derive the necessary mathematics for these. In addition, we extend previous work to handle higher-order motion by developing a new tile vs. triangle overlap test. We find that our tile-based rasterizer outperforms all other methods in terms of sample test efficiency, and that our generalization of an interval-based rasterizer is often fastest in terms of wall clock rendering time. In addition, we use our tile test to improve rasterization performance by up to a factor $5\times$ for semi-analytical motion blur rendering

CR Categories: I.3.3 [Computer Graphics]: Three-Dimensional Graphics and Realism—Display Algorithms I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Radiosity;

Keywords: motion blur, rasterization

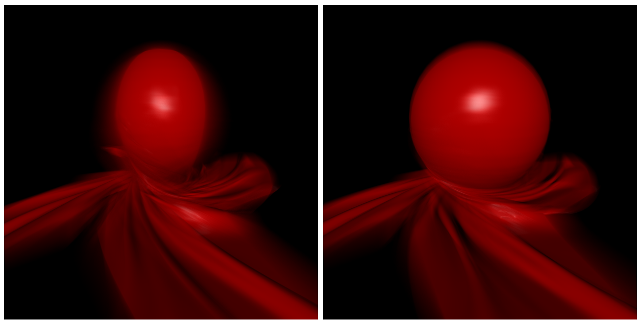


Figure 1: *The* CLOTH BALL *animation from the UNC Dynamic Scene Benchmark, with small triangles moving in circular arcs around the center of the ball. Left: When linearly interpolating the vertex positions between frame 73 and 93, there are severe shrinking artifacts in the motion blurred rendering. Right: We interpolate quadratically between three frames (73, 83 & 93) to create a curved motion trail for each vertex.*

1 Introduction

Motion blur rendering is used both in feature film graphics and for real-time rendering. Usually, the vertices move in linear paths, and higher-order motion is not used at all, used infrequently, or may be approximated by a set of linear vertex motion paths. The setting for our study is cases where higher-order motion blur matters, and this includes, for example, high-quality, long exposure renderings, i.e., most likely for feature films or still images. To correctly capture long, curved motion trails, the temporal visibility integral must be carefully integrated. Each visibility evaluation is expensive, and many evaluations per pixel are needed to get down to acceptable noise levels. We face a computationally expensive problem, where high quality is of utmost concern.

In animated real-time rendering, it is unlikely that the quality difference between, e.g., per-vertex linear motion trails and quadratic motion trails matters. Furthermore, the added cost of evaluating complex motion may be too expensive for an interactive application. Some care must be taken, however. Fast *rotations* are poorly approximated by linear per-vertex motion, and the artifacts can be quite surprising. Figure 1 shows an example of this.

The straightforward way of handling fast rotations (and other non-linear motion effects) is to add more linear segments within the frame and perform per-vertex linear motion blur rasterization within each segment. However, many linear segments may be needed to get acceptable quality, and the number of segments is another parameter to tweak. The purpose of our research is therefore to investigate the consequences and cost of directly using per-vertex motion trails specified as higher-order Bézier curves, i.e., quadratic curves and up.

Many previous techniques for efficient culling of a moving primitive against a screen space region break down if the per-vertex motion trails are no longer linear [Laine et al. 2011; Munkberg et al. 2011; Munkberg and Akenine-Möller 2012]. A naïve approach of simply testing all samples within the bounding box of the moving primitive is extremely costly. To avoid this, we present a new tile versus triangle test where each vertex is specified using a higher-order Bézier curve. In typical scenes, this reduces the render time with an order of a magnitude, since the majority of samples outside the moving primitive are culled early on. In addition, we also present a generalization of the interval-based rasterization technique to higher-order motion. Our generalization is surprisingly competitive in terms of wall clock time, while our tile-based rasterizer is better in terms of sample test efficiency.

Finally, we also analyze how previous semi-analytic motion blur rasterization (where the coverage test is solved analytically, but the depth function is approximated) generalizes to higher-order motion, and we propose a method to approximate the depth function in a way to avoid rendering errors.

2 Previous Work

There is a wealth of previous work on motion blur and stochastic rasterization. Sung et al. [2002] have a comprehensive survey of work on motion blur algorithms up to 2002, and we refer the interested reader to that survey for more information about the older algorithms.

Stochastic rasterization has become a popular research topic over the last few years. Cook et al. [1987] were first to use stochastic rasterization in their REYES renderer, where primitives are split and diced until sufficiently small in screen space. Shading is computed once, and then spread out over samples covering a moving primitive. Akenine-Möller et al. [2007] introduced time-dependent edge functions and rasterized an oriented box around the moving triangle. After that, a lot of focus has been spent on making the traversal more efficient. Fatahalian et al. [2009] presented details about Pixar's rasterization method, called INTERVAL, and also introduced another method called INTERLEAVE, which is based on interleaved sampling [Keller and Heidrich 2001]. A thorough analysis of the efficiency in a data-parallel setting was given. They also introduced the notion of sample test efficiency (STE), which is an efficiency measure of a rasterizer.

To make rasterization more efficient, researchers started developing tile tests. Laine et al. [2011] introduced a tile test in dual space for motion blur (3D), a screen space test for depth of field (4D), and the combination (5D). Munkberg et al. [2011] presented a similar test for motion blur based on the moving triangle's clip space bounding box, and also added an edge versus tile test for higher culling rates. Half-space culling has also been used for efficient tile tests for depth of field rasterization [Akenine-Möller et al. 2012]. Laine & Karras [2011] improved the culling efficiency of the dual space tile test in cases with simultaneous defocus and motion blur (5D) by taking correlations between time and lens parameters into account. Hyper-plane culling was then introduced for efficient tile tests for 5D rasterization [Munkberg and Akenine-Möller 2012], where linear bounds in ut- and vt- space were combined with a hyper-plane test for higher STE and lower total arithmetic cost.

Gribel et al. [2010] used time-dependent edge equations to analytically compute the overlap in time for a single sample point. This allowed them to render temporally alias-free images by solving time-continuous edge equations. A visibility engine based on depth intervals was also presented together with an oracle-based compression method. Line sampling [Jones and Perry 2000] has been used to extend motion blur and depth of field rasterization algorithms. Gribel et al. [2011] extended their previous method, which computed an analytical overlap in time, with lines samples in the spatial domain. Tzeng et al. [2012] used analytical line sampling over the lens, similar to how a motion blur rasterization was used to render depth of field [Akenine-Möller et al. 2007], for rapid rendering on the GPU.

Other methods have been proposed to speed up the pipeline, such as conservative backface culling for 3D, 4D, and 5D rasterization [Munkberg and Akenine-Möller 2011] and a hierarchical TZ-pyramid was introduced for space-time occlusion culling [Boulos et al. 2010]. Decoupled sampling techniques [Ragan-Kelley et al. 2011; Burns et al. 2010] decouple shading from visibility, with the goal of shading as little as possible. The shaded samples are stored in a cache for later reuse.

It should be noted that, as far as we know, all methods above use linear vertex motion, with the exception of the backface culling work [Munkberg and Akenine-Möller 2011], which is generalized to higher-order motion. The topic of our paper is to explore what happens when the order of the vertex motion is increased.

3 Theory

Linear per-vertex motion in three dimensions may look surprisingly complex when rendered on screen. This is mostly due to that after projection, the screen space vertex movement is a rational function in time. This makes time-dependent clipping very hard, for example. The projected motion trails in screen space are still straight lines, but their acceleration differ. With higher-order per-vertex motion, the vertex motion is determined by a curve in space. Note that higher-order vertex motion makes the depth function higher

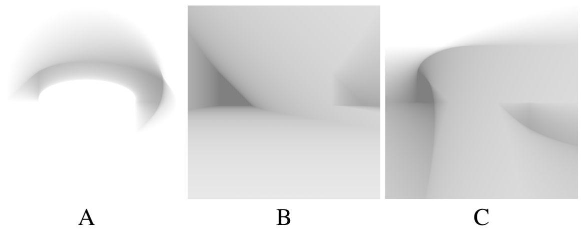


Figure 2: *Due to the quadratic motion in depth in these examples, the screen space projection of the moving primitive can be very complex. We let a black triangle move in x from left to right and approach the camera at $t = 0.5$. A: Quadratic motion trails in xz. B: Here, the triangle circles around the camera, and is behind the camera at $t = 0.5$. C: Same as B, but the triangle also rotates 180 degrees locally around the z-axis.*

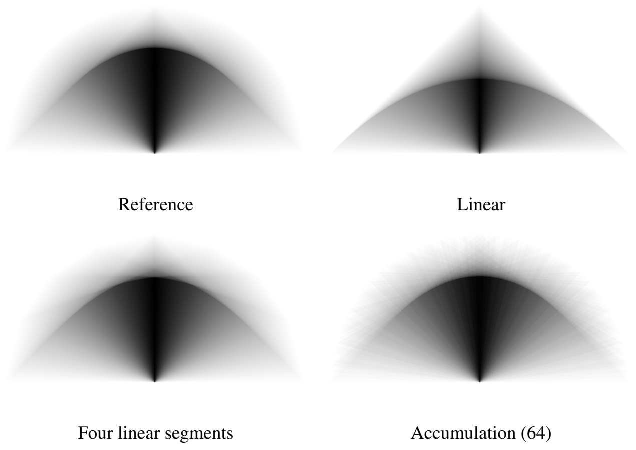


Figure 3: *A triangle with 90° rotation in the xy-plane approximated by quadratic Bézier curves. The upper left image shows a reference image with 256 spp. With linear motion trails (upper right), the swept area is reduced. If the motion is approximated with four piecewise linear segments, we approach the correct results, but still have artifacts (lower left). Finally, by using accumulation buffering, we can avoid higher-order coverage tests, but get ghosting artifacts, even at 64 accumulation passes. Zoom in on the electronic version of the paper to see these effects more clearly.*

order as well, which means that a triangle vertex can intersect the $w = 0$ plane more than once. In Figure 2, we show a few, somewhat unintuitive, examples of a single triangle with different types of quadratic motion.

Furthermore, the visibility integral is more complex and must be evaluated with high resolution, e.g., many temporal samples. In Figure 3, we show that coarse piecewise linear approximations of the movement, or a too small set of unique sample times, show up as noticeable artifacts. In this paper, we will discuss techniques to render these images efficiently with high quality.

3.1 Higher-Order Edge Equations

In this section, we will generalize the theory of edge equations for the case where the vertex motion paths are Bézier curves.

We use a notation similar to previous work [Gribel et al. 2010], where $\mathbf{p}(t)$ is a vertex in homogeneous clip space, i.e., the projection matrix has been applied but not perspective division. A vertex moving along a Bézier curve is then described by [Farin 2002]:

$$\mathbf{p}(t) = \sum_{i=0}^{n} B_i^n(t)\mathbf{p}_i, \tag{1}$$

where t is the time, which spans the unit interval $t \in [0, 1]$, n is the degree of the curve ($n = 1$ gives linear interpolation), $\mathbf{p}_i$, $i \in [0, \ldots, n]$, are the control points, and $B_i^n(t)$ are the Bernstein polynomials.

Recall that a homogeneous edge function [Akenine-Möller et al. 2007] for an edge between two vertices $\mathbf{p}$ and $\mathbf{q}$ is defined as $e(x, y) = \mathbf{n}_{\text{pq}} \cdot (x, y, 1)$, where $\mathbf{n}_{\text{pq}} = \mathbf{p} \times \mathbf{q}$ can be said to be the "normal" of the edge. Substituting $\mathbf{p}$ and $\mathbf{q}$ for two vertices moving along Bézier curves, $\mathbf{p}(t)$ and $\mathbf{q}(t)$, we get:

$$\begin{aligned}\mathbf{n}_{\text{pq}}(t) = \mathbf{p}(t) \times \mathbf{q}(t) &= \sum_{i,j}^{n} \frac{\binom{n}{i}\binom{n}{j}}{\binom{2n}{i+j}} B_{i+j}^{2n}(t)\big(\mathbf{p}_i \times \mathbf{q}_j\big) \\ &= \sum_{k=0}^{2n} B_k^{2n}(t)\mathbf{c}_k,\end{aligned} \tag{2}$$

where $\mathbf{c}_k$ are described by:

$$\mathbf{c}_k = \sum_{i+j=k} \frac{\binom{n}{i}\binom{n}{j}}{\binom{2n}{i+j}} \mathbf{p}_i \times \mathbf{q}_j. \tag{3}$$

Note that $\mathbf{c}_k$ can be thought of as the control points of the normal $\mathbf{n}_{pq}(t)$. As can be seen, Equation 2 is a Bézier curve of degree $2n$, and the edge equation is given by:

$$e(x, y, t) = \mathbf{n}_{\text{pq}}(t) \cdot (x, y, 1). \tag{4}$$

This is the generalized scalar triple product of the edge equation for higher-order Bézier per-vertex motion. A sample point is inside a triangle if all three edge equations evaluate to $e_i(x, y, t) \leq 0$.

In this paper, we will use this expression for the edge equation both when designing an analytical coverage test and for an efficient triangle edge versus screen space tile culling test. For stochastic sample testing, however, it is more efficient to interpolate the triangle vertices for a given time, and use a variant of the sample test suggested by Laine et. al [2011].

4 Traversal

In this section, we describe different alternatives for efficient traversal when rasterizing triangles undergoing higher-order motion. The core idea is to reduce the number of coverage tests by quickly discarding samples that cannot hit the moving triangle.

4.1 Higher-Order Interval Rasterization

A simple approach to reduce the cost of rasterizing a moving object is to subdivide the motion into many smaller temporal intervals, thereby solving a simpler problem in each segment. This is the idea of the INTERVAL rasterization algorithm (first proposed by Pixar and described by Fatahalian et al. [2009]). We extend it to support higher-order motion as follows. For each interval $[t_a, t_b]$, the Bézier curves for each vertex are reparameterized to obtain a new set of control points, representing the same curve segment within $[t_a, t_b]$. Using the *blossom* notation [Farin 2002], the new control points are given by $\mathbf{b}[t_s, t_s]$, $\mathbf{b}[t_s, t_e]$ and $\mathbf{b}[t_e, t_e]$ for the quadratic case. The same technique generalizes easily to higher-order motion.

To compute a screen-space bounding box for the moving triangle within $[t_s, t_e]$, we feed the nine new control points (three per vertex) into Laine & Karras' algorithm for efficient screen-space bounds [2011]. Note that higher-order motion trails often result in larger screen-space bounding boxes. Please refer back to Figure 1 for an example.

Finally, each coverage test is modified such that the triangle is positioned for the unique sample time. In the quadratic case, the added cost of positioning a moving triangle compared to linear motion makes the cost of the coverage test increase by about 50% (31 vs 21 fused multiply-add operations).

If many temporal subdivisions are used, the motion within each segment may be linearized with minor impact on quality, and in that case, a standard linear motion coverage test can be used. However, with only a few segments, piecewise linear motion may result in objectionable artifacts (see Figure 3).

Rasterization algorithms that sample the triangle temporally with a fixed set of sample times, such as accumulation buffering and interleaved sampling [Keller and Heidrich 2001] trivially extend to higher-order motion. The triangle (moving with curved motion trails) is only positioned once for each sample time, and a standard 2D rasterizer coverage test can be used. However, note that in cases where higher-order motion matters, the ghosting artifacts from these algorithms can be highly visible, and many unique sample times are needed.

4.2 Tile-based Traversal

In some rendering algorithms, it is beneficial to traverse hierarchically in screen space, and only visit every pixel or tile overlapping the moving triangle once per primitive. For example, in semi-analytical motion blur rendering [Gribel et al. 2010], an analytical coverage test computes the visibility integral valid over the entire shutter time for each pixel, and one want to avoid computing this redundantly. Tile-based rasterizers also have the benefit of supporting many optimizations, such as hierarchical traversal, occlusion culling, and increased cache coherence [Munkberg et al. 2011]. Furthermore, some shading approaches benefit of having a local visibility estimate for a triangle before shading.

In this section, we extend the work by Laine et al. [2011], Laine & Karras [2011], and Munkberg et al. [2011] by presenting new generalized tile tests for higher-order motion. Similar to previous work, we use both the triangle vertices and the triangle edges to design efficient culling tests against each screen-space tile.

Tile – Vertex Overlap Test Figure 4 shows a flatland illustration of the problem we need to solve in the tile vs. vertex overlap test. We derive the temporal overlap along the tile's extents in x and y, and compute the final result as the intersection of the resulting time intervals. Below, we describe the x-overlap test. For every tile on the screen, we set up two tile frustum planes aligned with the sides of the tile. Each frustum plane is defined by its plane equation $\mathbf{n}_i \cdot \mathbf{p} = 0$, where $\mathbf{n}_i$ is the frustum plane's outward normal (note that these planes go through the origin). This setup is shown in Figure 4.

We now need to find the intersection of the frustum planes against a vertex, $\mathbf{p}(t)$, moving along a Bézier curve in three dimensions. If *all* of the triangle's vertices are outside a frustum plane in an interval $t \in [a, b]$, we can safely ignore testing any samples within that time interval.

The intersection of a Bézier curve, $\mathbf{p}(t) = \sum B_j(t)\mathbf{c}_j$, and a plane, $\mathbf{n}_i \cdot \mathbf{p} = 0$, is given by:

$$\sum_j B_j(t)\mathbf{n}_i \cdot \mathbf{c}_j = 0, \tag{5}$$

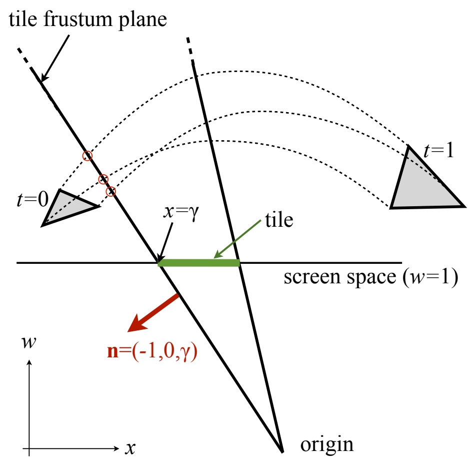


Figure 4: *A tile (green) in screen space and its two frustum planes bounding the x-extents of the tile. The planes are defined in xyw-space. The left frustum plane, passing through the point $(\gamma, 0, 1)$ has a normal $\mathbf{n} = (-1, 0, \gamma)$. We search for the intersections of the three moving triangle vertices with this plane (marked as red circles).*

which is a scalar polynomial equation of the same degree as the Bézier curve, $\mathbf{p}(t)$. Note that if the vertex motion is, for example, quadratic, a triangle vertex can enter a frustum plane twice within the frame.

Now, let us study the intersection with the plane passing through the left side of the tile. The normal of the left tile frustum plane has y-coordinate zero. As shown in Figure 4, we let the screen-space x-coordinate be parameterized by $\gamma \in [-1, 1]$. The normal vector then becomes $\mathbf{n}_i(\gamma) = (-1, 0, \gamma)$.

If we insert this normal into Equation 5, we get:

$$\sum_j B_j(t)\mathbf{n}_i(\gamma) \cdot \mathbf{c}_j = 0 \Longleftrightarrow \sum_j B_j(t) d_j(\gamma) = 0, \qquad (6)$$

where $d_j(\gamma) = -c_{j_x} + \gamma c_{j_w}$ is the orthogonal distance to the plane of each control point. Note that the following (see Equation 6) is a tensor product surface:

$$s(t, \gamma) = \sum_j B_j(t) d_j(\gamma), \qquad (7)$$

which is linear in γ and the degree of the Bézier curve in t.

Solving $s(t, \gamma') = 0$ for a given γ' gives us the times when a vertex passes through the frustum plane through $x = \gamma'$. The naïve approach is to interpolate each scalar control point to $d_j(\gamma') = -c_{j_x} + \gamma' c_{j_w}$, and find the solutions to the polynomial equation $\sum_j B_j(t) d_j(\gamma') = 0$.

Our key insight is that because the surface is linear in γ, we can bound $s(t, \gamma)$ at $\gamma = -1$ and $\gamma = 1$ with *subdividable linear efficient function enclosures* (slefes) [Peters 2003], and then linearly interpolate these bounds in γ. Let $\{\mathbf{a}_i\}$, $i \in [0, n]$ denote the breakpoints of the lower slefe at $\gamma = -1$ (i.e., a piecewise linear lower bound of $s(t, -1)$) and $\{\mathbf{b}_i\}$, the corresponding breakpoints of the lower slefe at $\gamma = 1$, bounding $s(t, 1)$.

As can be seen in the example with quadratic motion in Figure 5, the slefes sweep out two bilinear patches defined by the vertices

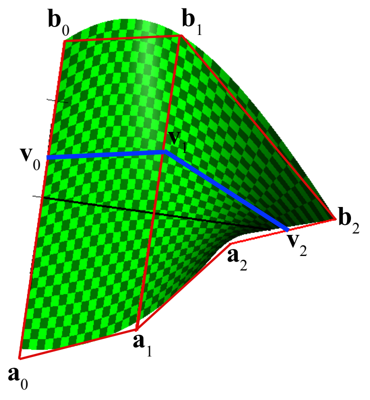


Figure 5: *A surface with the lower bound outlined in red. The lower bound consists of two bilinear patches for quadratic per-vertex motion. The $\mathbf{a}_i$ coefficients represent breakpoints of the lower slefe at $\gamma = -1$ and $\mathbf{b}_i$ at $\gamma = 1$. The $\mathbf{v}_i$ coefficients represent a piecewise linear bound (blue) valid for a specific gamma value. The $\mathbf{v}_i$'s are obtained by linear interpolation of $\mathbf{a}_i$ and $\mathbf{b}_i$.*

$(\mathbf{a}_0, \mathbf{a}_1, \mathbf{b}_1, \mathbf{b}_0)$ and $(\mathbf{a}_1, \mathbf{a}_2, \mathbf{b}_2, \mathbf{b}_1)$. In our implementation, we compute the $\mathbf{a}_i$ and $\mathbf{b}_i$ slefe breakpoints in the triangle setup.

For a given tile coordinate, γ', we linearly interpolate between $\mathbf{a}_i$ and $\mathbf{b}_i$ to obtain the vertices, $\mathbf{v}_i(\gamma')$, as shown in Figure 5. For quadratic motion, this gives us two linear segments:

$$\begin{aligned} \mathbf{l}_0(t) &= (1-t)\mathbf{v}_0(\gamma') + t\mathbf{v}_1(\gamma'), \\ \mathbf{l}_1(t) &= (1-t)\mathbf{v}_1(\gamma') + t\mathbf{v}_2(\gamma'), \end{aligned} \qquad (8)$$

where we solve for the (up to two) t-values using $\mathbf{l}_i(t) = 0$.

This technique generalizes to motion along Bézier curves of degree n, where there will be n piecewise linear segments, and up to n t-values where $\mathbf{l}_i(t) = 0$.

Triangle Test To generalize the test from a single vertex to a triangle, we compute slefes for all three vertices and take the min/max of the slefe breakpoints, $\mathbf{a}_i$ and $\mathbf{b}_i$. Here, we use minimum for the lower and maximum for the upper bound slefe respectively. This gives us a conservative lower and upper bound slefe for the union of the surfaces $s_i(t, \gamma)$ for all three vertices. Again, this can be performed once in the triangle setup. For each tile, we then interpolate *one* set of $\mathbf{v}_i$ coefficients for the triangle. If all $\mathbf{v}_i$ are positive for a frustum plane, the triangle can be culled for that tile.

To compute a t-interval of potential overlap, we need to determine the time t_{in} when the triangle first enters the tile frustum, and a time t_{out} when the triangle leaves the tile frustum (please refer to Figure 4 for an example). It is for this reason that we keep both the lower and upper bound slefes. We can derive t_{in} from the solutions to $\mathbf{l}_i(t) = 0$ using the lower bounds. To obtain t_{out}, we proceed as above, but instead use the upper bound slefe to compute the linear $\mathbf{l}_i(t)$ functions. With this, we obtain an interval $[t_{in}, t_{out}]$ for when the triangle overlaps with the tile's range in x. The same procedure is then executed for the two tile frustum planes bounding the tile's range in y to obtain another interval in t. Finally, the intersection of these two intervals is a conservative interval of potential overlap. In practice, instead of bounding using $\gamma = -1$ and $\gamma = 1$, we use the extents of the triangle's screen-space bounding box for tighter bounds. The total cost per tile for the culling test is roughly double that of Laine et al.'s tile culling test for linear motion blur [2011].

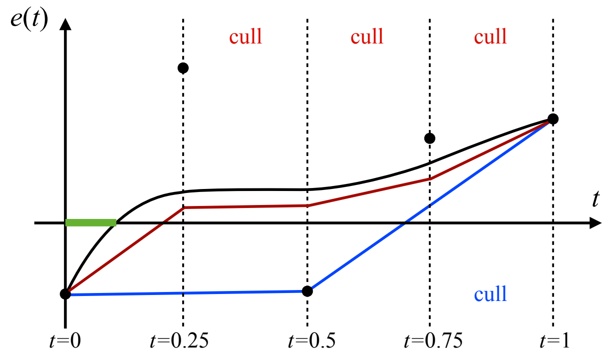


Figure 6: *Solving for $e(t) \leq 0$ for quadratic edge functions ($n = 2$). By finding a lower slefe bound (red) for the edge equation's projection on the tile center, we can cull more regions than the convex hull (blue) derived from the curve's control points. The analytical region where $e(t) < 0$ is marked in green. The black circles are the control points of the Bézier curve.*

Tile – Edge Overlap Test Next, we sketch a test for a moving edge against a screen-space tile, when the edge's vertices move along Bézier curves.

As shown in Section 3, the edge equation is a Bézier curve of degree $2n$. At triangle setup, we compute the $2n + 1$ control points, $\mathbf{c}_i$, according to Equation 4 for this edge. Then, for each tile:

1. Compute the $2n + 1$ scalar control points, $d_i = \mathbf{c}_i \cdot (x, y, 1)$. The overlap with the tile and the edge is determined by when the Bézier curve $e(t) = B_i(t)d_j$ is less than zero.
2. If all $d_i > 0$ the tile can be culled.
3. Compute a lower slefe from d_i to obtain $2n + 1$ breakpoints d'_i.
4. If all $d'_i > 0$ the tile can be culled.
5. if $d'_i > 0$ **and** $d'_{i+1} > 0$ the moving edge is outside the current tile within the interval $t_i = \frac{1}{2n}[i, i + 1]$. This last step is illustrated in Figure 6 for $n = 2$, i.e., for quadratic vertex paths.
6. Conservative times of intersection are obtained by solving for when the line segments $(\frac{i}{2n}, d_i) : (\frac{i+1}{2n}, d_{i+1})$ intersect the x-axis.

Note that the unit time interval ($t \in [0, 1]$) is split into $2n$ sub-intervals above. For quadratic motion, this means that the unit time interval is split into four sub-intervals. The total cost per tile to evaluate the test for all three triangle edges, including solving for the times of intersection, is roughly 200 operations.

Note that we do not have to evaluate at all four tile corners if we pad the control points at triangle setup. Let half the tile width (height) be denoted Δ_w (Δ_h). With $\Delta = (\Delta_w, \Delta_h, 0)$ we have:

$$\mathbf{c}_i \cdot (\mathbf{p} \pm \Delta) \geq \mathbf{c}_i \cdot \mathbf{p} - (|c_{i_x}|\Delta_w + |c_{i_y}|\Delta_h). \tag{9}$$

The second term is independent of $\mathbf{p}$, so we subtract it from the w-component of $\mathbf{c}_i$ at the triangle setup, and only need to evaluate $\mathbf{c}_i \cdot \mathbf{p}$ per tile to get a conservative lower bound over the tile.

Finally, the time intervals of overlap obtained from the moving edges can be intersected with the intervals from the moving vertices, and only in the remaining interval do we evaluate coverage tests. In our results, we study the efficiency of these two tests combined.

5 Semi-Analytical Rendering

Inside Test and Barycentric Interpolation Equation 4 can be interpreted geometrically as a *volume* in homogeneous space, i.e., a polyhedra bounded by the view-point, the edge surface, and the sample point. The sign of this volume depends on the sample point being outside (positive) or inside (negative) the edge. Equation 4 can thus be utilized as an inside function during rasterization. When the order of motion is n, the order of the edge equation becomes $2n$, as can be seen in Equation 2. For quadratic per-vertex motion ($n = 2$), Equation 4 has order 4, meaning that a sample point can go from being inside to outside, or the opposite, up to four times as the edge moves. A sample point is inside a triangle if all three edge equations evaluate to $e_i(x, y, t) \leq 0$.

The perspective-correct barycentric coordinates are defined as:

$$b_i(x, y, t) = \frac{e_i(x, y, t)}{\sum_j e_j(x, y, t)}. \tag{10}$$

The normalized depth ($\frac{z}{w}$) at a sample point for a triangle $\{\mathbf{p}_0(t), \mathbf{p}_1(t), \mathbf{p}_2(t)\}$ can be interpolated as:

$$\frac{z}{w}(x, y, t) = \frac{\sum_i b_i(x, y, t)p_{i_z}(t)}{\sum_i b_i(x, y, t)p_{i_w}(t)}. \tag{11}$$

Note that each term in the sum is a polynomial of degree $3n$ in t, e.g., for quadratic motion, the normalized depth is a rational function of degree 6. Alternatively, one can use w directly, which is a rational function of degree 6 in the numerator and degree four in the denominator. See the work by Andersson et al. [2013] for more information about the depth function for a moving triangle.

Semi-Analytical Rasterization In a semi-analytical rasterizer, the edge equations (Equation 4) of each triangle are *solved* instead of sampled. The acquired roots are used to generate inside-intervals over continuous intervals of time, Δt. Intervals generated during rasterization are stored per sample in lists and resolved for final visibility at the end of the rendering pass. We use an algorithm that overall is similar to that of Gribel et al. [2010], including an oracle-based, lossy interval compression algorithm. In addition, the fast opaque-geometry resolve algorithm by Barringer et al. [2012] is employed. Two different quartic solvers are used, namely, a direct analytical solver [Glassner 1990; Paeth 1995] and an iterative solver based on Bézier clipping [Sederberg and Nishita 1990].

One aspect of the algorithm that need renewed consideration for higher-order motion is the way depth is handled over each interval. In the linear motion case, Gribel et al. [2010] approximate depth as a linear function over Δt, and even though they show that the error introduced this way is limited, this is an assumption we cannot make since vertices are now moving quadratically instead of linearly. To this end, we use a refinement scheme similar to Gribel et al. [2011], where intervals are split if the approximation error exceeds a tolerance. Each interval evaluates the true depth in its endpoints, and splitting an interval effectively means that the depth function is sampled accurately in one additional point in t.

Consider Figure 7 for a simplified failure case. During this frame, a red triangle moves to the front and back again while a blue triangle passes by behind it. If depth is linearised, the red triangle will be erroneously occluded by the blue one. By measuring and limiting the approximation error, e.g., to 1% of the depth range, and

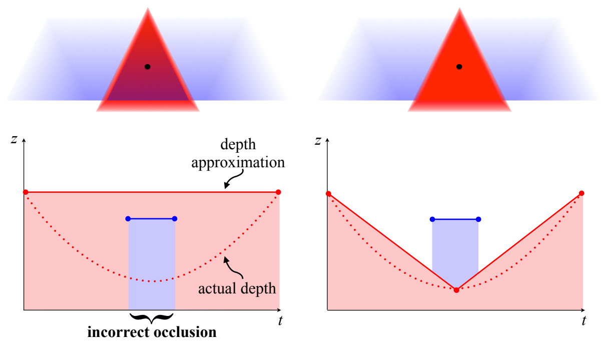


Figure 7: *Simplification of how overly crude depth approximations can lead to visibility errors. A red triangle jo-jo's toward the camera while a blue triangle passes by behind it. When the depth of the red triangle is linearized, it wrongly becomes occluded by the blue one. By splitting the interval, adaptively if needed, the effects of this error can be reduced or, as here, even eliminated.*

	TILE			INTERVAL		
	Tile size			Num intervals		
Scene	4×4	2×2	1×1	4	16	64
CLOTHBALL	11	18	**24**	1.2	6.4	15
SPONZA	47	65	**79**	10	18	21
BIGGUY	17	27	**36**	3.1	11	18
BEN	12	21	**34**	0.68	5.2	14
HAND	30	46	**62**	7.8	19	24
HAIRBALL	3.5	7.5	**14**	0.75	2.8	4.8

Table 1: *Sample test efficiency (higher is better) results using our set of test scenes. The tile tests with* 1×1 *tiles have consistently the highest STE.*

splitting the interval, this error can be reduced. We estimate this error by sampling the true depth function at the midpoint of Δt, and compare it to the linearized depth function. Note that this scheme does not completely eliminate the problem, but it limits the extents of intervals that are erroneously occluded. In Section 6, we present measurements of this error in practical scenes.

6 Results

6.1 Culling Test Efficiency

In this section, we evaluate different traversal strategies for higher-order motion. Unless noted otherwise, all results in this section are generated with 64 samples per pixel with a simple shader reinterpreting the normal as a color. We use a standard set of test scenes as shown in Figure 8. All scenes contain large, non-linear motion.

We denote the tile-based traversal using the tests from Section 4.2 TILE, and compare against our generalizations of INTERVAL from Section 4.1. For reference, we also include an INTERLEAVE rasterizer with a fixed set of sample times. In our test scenes, we used 256 sample times in INTERLEAVE (using 2×2 tiles in xy) to match the visual quality of 64 pseudo-random stochastic samples per pixel.

In Table 1, we report sample test efficiency (STE), i.e., how many of the tested samples that hits the primitive. As can be seen, the tile tests from Section 4.2 have consistently the highest STE scores. As expected, STE increases with smaller tile sizes and more intervals, respectively.

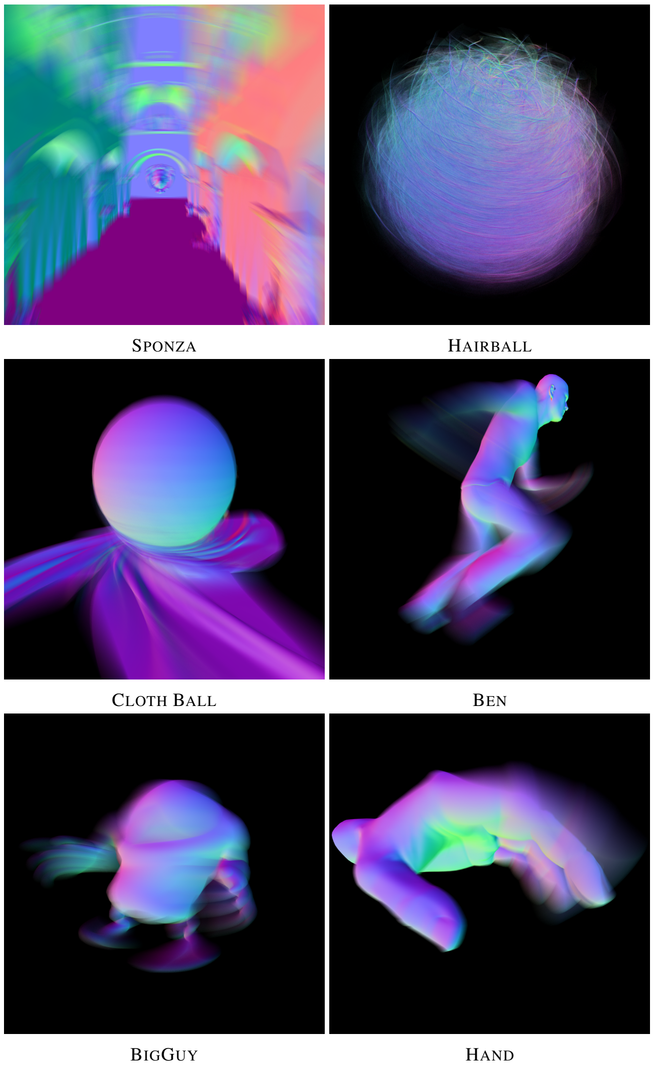


Figure 8: *The test scenes used in the evaluation. All rendered at* 1024×1024 *pixels, using 64 samples per pixel.*

However, by only looking at STE, the cost of executing culling tests is not taken into account, and in practice, depending on the architecture, they may actually account for a large fraction of the total execution time. In practice, this is an intricate problem. For example, when using INTERVAL with N intervals for a static scene, every triangle position is unnecessarily interpolated $N - 1$ times. For large motion, on the other hand, the temporal subdivision is critical to reduce the number of coverage test. The tile tests are executed for each tile covered by the moving primitive's bounding box, and for large motion trails, this cost may be significant.

Our approach to discuss the costs is to report execution times for reasonably optimized scalar, single-threaded implementations of the algorithms. In Table 2, we use the same set of algorithm as in the STE evaluation above (Table 1), and we encourage the reader to compare the two tables. Our generalized INTERVAL (Section 4.1) with many temporal subdivisions is the faster option in most scenes, especially for scenes with many small triangles. For scenes with larger triangles (Sponza and Hand) the tile tests are more efficient. This is due to culling against triangle edges. Note that there is no clear relationship between STE and execution time. In practice, the cost of culling versus STE must be balanced for the target platform.

Scene	TILE Tile size 4 × 4	 2 × 2	 1 × 1	INTERVAL Num intervals 4	 16	 64
CLOTH BALL	1	1.38	3.5	4.3	0.93	**0.61**
SPONZA	**1**	1.0	1.7	2.8	1.8	2.0
BIGGUY	1	1.2	2.4	2.9	1.1	**0.77**
BEN	1	1.6	4.3	7.1	1.1	**0.52**
HAND	**1**	1.1	1.7	1.1	1.2	1.1
HAIRBALL	1	1.0	2.0	2.8	0.87	**0.65**

Table 2: *Performance results using our set of test scenes (lower is better). For each scene, the fastest version is marked with bold font. All results are normalized relative to* TILE *with* 4×4 *tiles.*

Performance Scaling From Table 2, we note that the tile-based traversal strategy is less efficient for highly tessellated scenes, such as the HAIRBALL scene, while it performs very well for lower and moderate tessellation. However, recent work has shown that level of detail algorithms can be aggressively applied to stochastic rendering for both geometry [Tzeng et al. 2012] (for defocus blur) and shading [Vaidyanathan et al. 2012]. Figure 9 shows the visual impact of a drastic reduction in tessellation rate for one of our test scenes. A general observation is that a lower level of detail can be used when the amount of blur is high.

In Figure 10, we vary the amount of motion in the Crytek SPONZA scene, and compare the performance scaling of TILE, INTERVAL and INTERLEAVE. For small motion, TILE is faster, and has substantially higher STE. However, as the motion increases, the STE scores and performance converges. For highly tessellated scenes, INTERVAL scales better.

Our results are interesting from many perspectives. For example, our generalized INTERVAL method is the best recommendation for a software-based implementation, since it is faster in terms of wall clock rendering time. However, note that these performance numbers may not directly translate to an efficient parallel implementation nor an efficient hardware implementation of the algorithms. In fact, it is likely that a hardware implementation would favor the tile-based approach due to the many advantages, e.g., occlusion culling, shading, and cache performance, that come with such a traversal order. However, a thorough evaluation of this is left for future work. In addition, INTERVAL and INTERLEAVE do not provide any benefit for semi-analytical rendering, while TILE does.

Importance of Slefes Instead of using slefes when bounding the moving triangle, we could directly use the convex hull around the triangle's Bézier control points for coarser but slightly cheaper bounds. For highly non-linear motion, however, the convex hull around the control points is often overly conservative. This is illustrated in Figure 6. In addition, the difference in cost is modest. For the tile-vertex overlap text, the slefes are computed once in the triangle setup. For the tile-edge test, the difference in cost of computing a lower slefe and a lower convex hull of a fourth order Bézier curve is only a few instructions. For completeness, we include the STE scores for TILE with and without slefes below, measured using 1×1 pixel tiles.

	CLOTH	SPONZA	BIGGUY	BEN	HAND	HAIRB
Slefes	24	79	36	34	62	14
C hull	2.4	71	3.6	7.4	36	5.5

Note that the tighter bounds from slefes result in significantly higher STE scores for many scenes, particularly in the two scenes with large rotations (CLOTH BALL and BIGGUY).

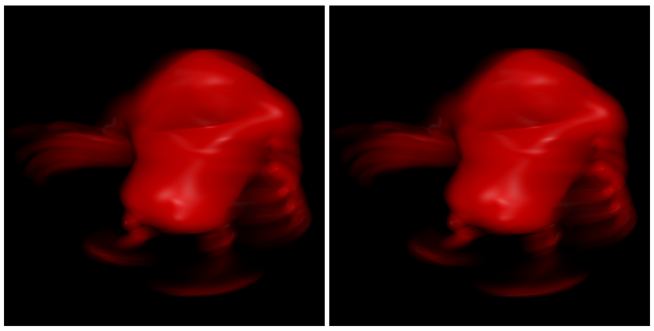
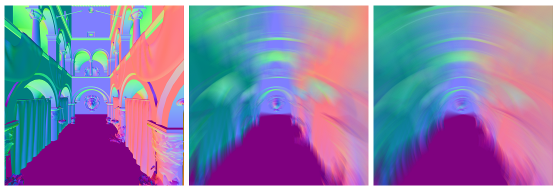


Figure 9: *Two versions of the BigGuy character with Phong shading, one with 5700 triangles, the other with 0.4M triangles. Note that there is hardly any visual difference (PSNR: 47dB) but the render time for the left image is* $5\times$ *longer.*



Rotation:	0°	20°	40°
STE (%):	**62**|23|22	**47**|19|20	**37**|16|19
Rel. time:	**1.0**|2.9|1.7	2.1|3.4|**2.0**	3.4|4.0|**2.1**

Figure 10: *Test with increasing motion on the Crytek Sponza scene. The STE (higher is better) and timings (lower is better) are on the form (*TILE | INTERVAL|INTERLEAVE*) and are normalized relative to the leftmost timing for* TILE. *We use* 4×4 *tiles and 64 intervals for this benchmark (fastest settings for large blurs).*

Piecewise Linear Motion For reference, we also implemented a version of INTERVAL that uses piecewise linear segments in each interval to reduce the cost of each coverage test. Interestingly, this did not give a large performance benefit. The reason for this is that for good quality in our test scenes, 16 linear segments or more are needed. As the number of segments grow, the cost of traversal and control flow grows relatively to the cost of the inside test. With 64 intervals, we noticed less than 20% performance increase for piecewise linear motion for our scalar implementation. The STE scores are very similar to those of INTERVAL.

6.2 Semi-Analytical Rendering

Figure 11 and 12 show a series of images rendered with our experimental semi-analytical rasterizer. We use two quality metrics: the *peak signal-to-noise ratio* (PSNR) for direct image quality compared to a reference image, and *Visibility Accuracy*. The latter is the rate, with respect to a stochastically sampled reference image, by which the *correct primitive* is visible. This metric takes accuracy in both t and z into account. The scenes are intended to stress the rasterizer in various ways: SPONZA contains varying degrees of camera rotation while BEN and HAND display motion with high depth complexity.

In our evaluation of the semi-analytical rasterizer, all images were rendered in 800×800 resolution. For reference images 512 samples per pixel were used. The semi-analytical rasterizer samples once at pixel centres. To isolate measurements to temporal visibility, the spatial samples of the reference images were fixed to the pixel centres as well, while being distributed along t as usual. Constant shading was used per primitive to remove illumination frequencies. Depth approximation errors were limited by splitting intervals with errors exceeding 1% of the depth range.

Both BEN and HAND display very high quality, with PSNR in the $49 - 50$dB range and $97 - 99\%$ visibility accuracy. Since both scenes contain significant motion over the z-axis, this seems to indicate that the suggested depth approximation is reasonable. SPONZA contains highly tessellated objects mixed with large floor and wall polygons. Since motion is induced by camera rotation, the motion applies to all parts of the scene, which make errors present at more places in the image. Hence, the PSNR is lower, around $41 - 42$dB, which may be acceptable.

To find roots of the quartic edge equations, we evaluated two solvers, namely, a power-based analytical quartic solver [Glassner 1990; Paeth 1995] and Bézier clipping [Sederberg and Nishita 1990]. Bézier clipping operates on Bernstein form polynomials and finds roots iteratively in the $[0, 1]$-range in ascending order. As a root is found, it is deflated [Press et al. 2007], and the process continues with a lower order polynomial. Both solvers produce practically identical visual results, as can be seen in Figure 12. Our experience is that the power-based solver is fast and, despite requiring a conversion to power form, still quite robust. It comes with a hit-or-miss approach however. Customization possibilities are limited and once roots start to drift, they are difficult to redeem. Bézier clipping offer the opposite: highly customizable, perpetual convergence even in hard cases (up to 150 iterations are needed in difficult cases, such as BEN's arms), but to a higher computational cost. Since the edge equations are already defined on Bernstein form, no conversion is needed when using Bézier clipping. It is also applicable to not only quartics, but polynomials of any order.

The tile tests described in Section 4.2 can easily be applied to the semi-analytical motion blur rasterizer. We can benefit from the early reject test to cull entire tiles, and we also gain significant performance by incrementally computing the intersection of the time intervals received from the tile tests and the time intervals from the analytical edge tests, and terminating if an empty interval is obtained. On the tests scenes in Figure 8, this gives us a performance improvement between $1.7\times$ and $4.9\times$ (average: $2.7\times$), when compared to an implementation such as initially described by Gribel et. al [2010].

7 Conclusions and Future Work

In this paper we derive mathematical background for high-order vertex motion and provide practical contributions for its implementation in a rasterization framework. Notably, we present novel tile tests for high-order motion, which greatly reduces the of number of inside tests compared to existing algorithms. In addition, we also present an INTERVAL-based algorithm generalized to higher-order motion. Interestingly, our INTERVAL-based rasterizer is faster in terms of wall clock rendering time, while our tile-based rasterizer is better in terms of sample test efficiency (STE). As we have argued, both of these may be useful, but in different contexts. Furthermore, we extend previous work on semi-analytical rasterization to higher order motion, and show that our novel tile tests are beneficial for that use case.

One important topic for future work is shading. Even though any shading model can be utilized in any of the rasterizers, we have limited ourselves to simple shading models and brute force sampling in this paper. In the setting of higher-order motion, with complex and prolonged motion, it is of particular relevance to consider reuse strategies, such as decoupled and cached shading. Also, more work can be done in terms of time-dependent filtering and mipmapping.

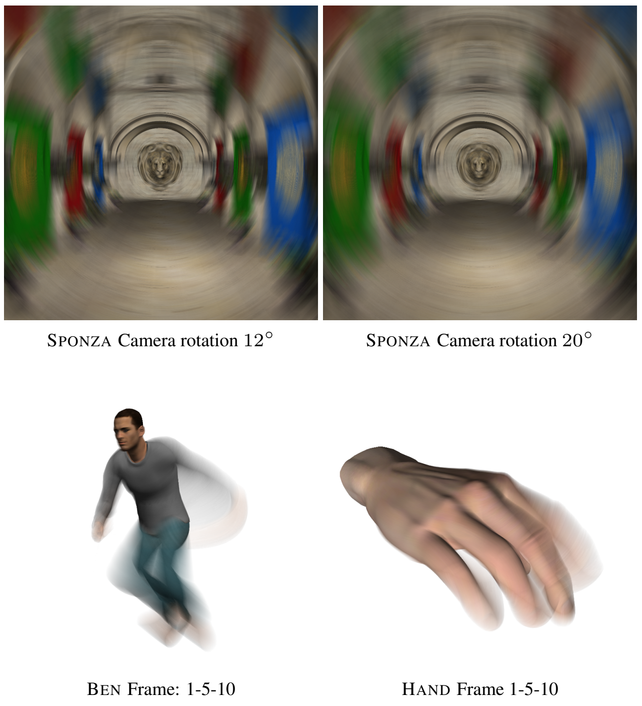


Figure 11: *Semi-analytical rendering with shading. The shading sampling rate is 64 per visible t-interval. Note the smooth visibility and absence of temporal aliasing. In static regions, some spatial aliasing can be seen, which is due to the fact that only one spatial sample is used per pixel. Also note the long motion paths of BEN's hands, along z and in the xy-plane, respectively. Measurements of these images are presented in Figure 12. The polygon counts are $262k$ for SPONZA, $78k$ for BEN, and $16k$ for HAND.*

In addition, an implementation in software running on a GPU would also be interesting for future work.

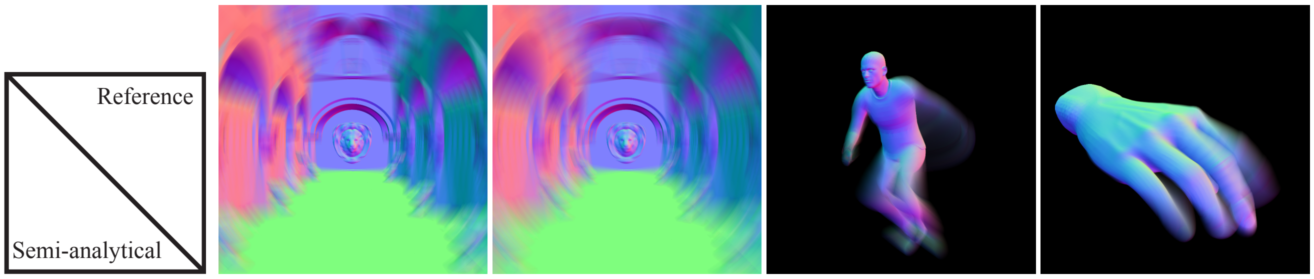
Finally, we argue that even though high-order motion has a narrow scope of application and is unnecessary in most normal settings, it is very powerful in those special cases where motion becomes aggressive and complex, and should therefore be accessible for the community nevertheless.

Acknowledgements We thank the anonymous reviewers and Intel's Advanced Rendering Technology team for their valuable feedback. We also thank Tom Piazza and Charles Lingle for supporting this research. The BEN and HAND animation sequences come from the Utah 3D Animation Repository. CLOTHBALL comes from UNC Dynamic Scene Benchmarks. HAIRBALL is courtesy of Samuli Laine. BIGGUY is courtesy of Bay Raitt, Valve. Sponza is courtesy of Marko Dabrovic/Frank Meinl. Tomas is a *Royal Swedish Academy of Sciences Research Fellow* supported by a grant from the Knut and Alice Wallenberg Foundation.

References

AKENINE-MÖLLER, T., MUNKBERG, J., AND HASSELGREN, J. 2007. Stochastic Rasterization using Time-Continuous Triangles. In *Graphics Hardware*, 7–16.

AKENINE-MÖLLER, T., TOTH, R., MUNKBERG, J., AND HASSELGREN, J. 2012. Efficient Depth of Field Rasterization using a Tile Test based on Half-Space Culling. *Computer Graphics Forum, 31*, 1, 3–18.



	SPONZA rot 12°	SPONZA rot 20°	BEN Frame: 1-5-10	HAND Frame 1-5-10
PSNR, Power solver :	42.7dB	41.1dB	49.1dB	50.7dB
PSNR, Bézier clipping:	42.7dB	41.1dB	49.1dB	50.7dB
Vis.Acc. Power solver:	98.3%	97.4%	97.7%	99.5%
Vis.Acc. Bézier clipping:	98.3%	97.4%	97.8%	99.6%

Figure 12: *Quality measurements for semi-analytical renderings. Higher PSNR indicates better visual resemblance to the reference image. The Visibility Accuracy provides the rate over a distribution of* (x, y, t)*-samples by which primitives are correctly visible, again compared to a stochastic reference. Reference images were rendered using* 512 *stochastic samples per pixel.*

ANDERSSON, M., MUNKBERG, J., AND AKENINE-MÖLLER, T. 2013. Stochastic Depth Buffer Compression using Generalized Plane Encoding. *Computer Graphics Forum, 32*, 2, 103–112.

BARRINGER, R., GRIBEL, C. J., AND AKENINE-MÖLLER, T. 2012. High-Quality Curve Rendering using Line Sampled Visibility. *ACM Transactions on Graphics, 31*, 6, 162:1–162:10.

BOULOS, S., LUONG, E., FATAHALIAN, K., MORETON, H., AND HANRAHAN, P. 2010. Space-Time Hierarchical Occlusion Culling for Micropolygon Rendering with Motion Blur. In *High Performance Graphics*, 11–18.

BURNS, C. A., FATAHALIAN, K., AND MARK, W. R. 2010. A Lazy Object-Space Shading Architecture With Decoupled Sampling. In *High Performance Graphics*, 19–28.

COOK, R. L., CARPENTER, L., AND CATMULL, E. 1987. The Reyes Image Rendering Architecture. In *Computer Graphics (Proceedings of ACM SIGGRAPH 87)*, 96–102.

FARIN, G. 2002. *Curves and Surfaces for CAGD—A Practical Guide*, 5th ed. Morgan-Kaufmann.

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 *High Performance Graphics*, 59–68.

GLASSNER, A. S. 1990. *Graphics Gems*. Academic Press, Inc.

GRIBEL, C. J., DOGGETT, M., AND AKENINE-MÖLLER, T. 2010. Analytical Motion Blur Rasterization with Compression. In *High-Performance Graphics*, 163–172.

GRIBEL, C. J., BARRINGER, R., AND AKENINE-MÖLLER, T. 2011. High-Quality Spatio-Temporal Rendering using Semi-Analytical Visibility. *ACM Transactions on Graphics, 30*, 4, 54:1–54:11.

JONES, T. R., AND PERRY, R. N. 2000. Antialiasing with Line Samples. In *Eurographics Workshop on Rendering*, 197–205.

KELLER, A., AND HEIDRICH, W. 2001. Interleaved Sampling. In *Eurographics Workshop on Rendering*, 269–276.

LAINE, S., AND KARRAS, T. 2011. Improved Dual-Space Bounds for Simultaneous Motion and Defocus Blur. NVIDIA Technical Report NVR-2011-004, Nov.

LAINE, S., AILA, T., KARRAS, T., AND LEHTINEN, J. 2011. Clipless Dual-Space Bounds for Faster Stochastic Rasterization. *ACM Transactions on Graphics, 30*, 4, 106:1–106:6.

MUNKBERG, J., AND AKENINE-MÖLLER, T. 2011. Backface Culling for Motion Blur and Depth of Field. *Journal of Graphics, GPU, and Game Tools, 15*, 12, 123–139.

MUNKBERG, J., AND AKENINE-MÖLLER, T. 2012. Hyperplane Culling for Stochastic Rasterization. In *Eurographics – Short Papers*, 105–108.

MUNKBERG, J., CLARBERG, P., HASSELGREN, J., TOTH, R., SUGIHARA, M., AND AKENINE-MÖLLER, T. 2011. Hierarchical Stochastic Motion Blur Rasterization. In *High Performance Graphics*, 107–118.

PAETH, A. W. 1995. *Graphics Gems V*. Academic Press, Inc.

PETERS, J. 2003. Mid-Structures Linking Curved and Linear Geometry. In *SIAM Conference on Geometric Design and Computing*, 1–10.

PRESS, W. H., TEUKOLSKY, S. A., VETTERLING, W. T., AND FLANNERY, B. P. 2007. *Numerical Recipes – The Art of Scientific Computing*, 3rd edition ed. Cambridge University Press.

RAGAN-KELLEY, J., LEHTINEN, J., CHEN, J., DOGGETT, M., AND DURAND, F. 2011. Decoupled Sampling for Graphics Pipelines. *ACM Transactions on Graphics, 30*, 3, 17:1–17:17.

SEDERBERG, T., AND NISHITA, T. 1990. Curve Intersection using Bézier Clipping. *Computer-Aided Design, 22*, 9, 538–549.

SUNG, K., PEARCE, A., AND WANG, C. 2002. Spatial-Temporal Antialiasing. *IEEE Transactions on Visualization and Computer Graphics, 8*, 2, 144–153.

TZENG, S., PATNEY, A., DAVIDSON, A., EBEIDA, M. S., MITCHELL, S. A., AND OWENS, J. D. 2012. High-Quality Parallel Depth-of-Field Using Line Samples. In *High Performance Graphics*, 23–31.

VAIDYANATHAN, K., TOTH, R., SALVI, M., BOULOS, S., AND LEFOHN, A. 2012. Adaptive Image Space Shading for Motion and Defocus Blur. In *High Performance Graphics*, 13–21.