

# Time Interval Ray Tracing for Motion Blur

Konstantin Shkurko, Cem Yuksel, Daniel Kopta, Ian Mallett, and Erik Brunvand, *Member, IEEE*

**Abstract**—We introduce a new motion blur computation method for ray tracing that provides an analytical approximation of motion blurred visibility per ray. Rather than relying on timestamped rays and Monte Carlo sampling to resolve the motion blur, we associate a time interval with rays and directly evaluate when and where each ray intersects with animated object faces. Based on our simplifications, the volume swept by each animated face is represented using a triangulation of the surface of this volume. Thus, we can resolve motion blur through ray intersections with stationary triangles, and we can use any standard ray tracing acceleration structure without modifications to account for the time dimension. Rays are intersected with these triangles to analytically determine the time interval and positions of the intersections with the moving objects. Furthermore, we explain an adaptive strategy to efficiently shade the intersection intervals. As a result, we can produce noise-free motion blur for both primary and secondary rays. We also provide a general framework for emulating various camera shutter mechanisms and an artistic modification that amplifies the visibility of moving objects for emphasizing the motion in videos or static images.

**Index Terms**—motion blur, ray tracing, sampling

## 1 INTRODUCTION

MOTION blur plays a vital role in realistic simulation of the camera image capturing process as well as in production of smooth and natural appearance of motion. Furthermore, it is a powerful artistic tool for expressing motion in both videos and static images.

In the context of ray tracing for high-quality rendering, motion blur has been typically handled using Monte Carlo sampling by attaching a random timestamp to each ray sample [1]. This approach adds an extra dimension to the sampling process and often reduces the effectiveness of adaptive sampling techniques commonly used for anti-aliasing. Furthermore each ray must intersect the scene geometry at its timestamp, which requires on-the-fly reconstruction of the scene geometry on a per-ray basis. This is cumbersome, particularly for deforming objects, and requires specialized acceleration structures, which are often inefficient at handling large motion with deformations. Moreover, like any Monte Carlo integration, increasing the number of samples reduces the noise, but never completely eliminates it.

In this paper, we propose *time interval ray tracing*, providing an analytical approximation for computing motion-blurred visibility. Our approach is based on a simplification of the concept of intersecting rays with the volumes swept by moving triangles [2]. We invoke four simplifying assumptions that allow us to efficiently evaluate the spatio-temporal intersections of a given ray with a time *interval* (as opposed to a single timestamp) using traditional ray intersection tests with *stationary* triangles. Hence, we can use any ray tracing acceleration structure *without modification* to handle dynamic geometry. As a result, we can produce *noise-free* motion blur for both primary and secondary rays (including those for shadows, reflections, and global illumination) using only a single ray sample. The technical contributions in this paper are:

- Simplifications that permit efficient intersections of a ray with moving geometry using only stationary triangles,
- An adaptive subdivision strategy to efficiently shade the intersection intervals of rays,
- A general framework to emulate various camera shutter mechanisms, and
- An artistic modification to amplify the visibility of motion-blurred objects.

Our results show that we can produce *noise-free* motion blur (Figure 1) very effectively and that our approach outperforms time-sampling strategies, especially in scenes with considerable motion.

## 2 BACKGROUND

In a camera, a *sensor* (film or electronic) integrates incident light over both space and time. A *shutter* controls the amount of light that reaches the sensor by modulating how long the sensor is exposed to light (*exposure*). Mechanical shutters, which are present in all analog and some high-end digital cameras, block light with a moving panel or with moving blades. Electrical “shutters” vary the time over which the sensor accumulates charge: a *global shutter* reads out the entire image at the same time while a *rolling shutter* reads scanlines sequentially. Relative motion between the camera and an object produces the photographic effect called *motion blur* as a result of the object being visible to different areas of the sensor over time. Since a shutter modulates the sensor’s exposure to light, its construction, type of motion, and speed affect the appearance of motion blur.

Methods simulating motion blur in computer graphics typically model continuous motion and deformation of objects by a series of snapshots in time, referred to as *keyframes*. Each keyframe stores all information necessary to reconstruct the object at its timestamp. Methods can be classified broadly into image-space and object-space techniques, which we summarize briefly. For a more complete

• K. Shkurko, C. Yuksel, D. Kopta, I. Mallett, and E. Brunvand are with the School of Computing, University of Utah, Salt Lake City, UT, 84112. E-mail: {kshkurko, dkopta, imallett, elb}@cs.utah.edu, cem@cemyuksel.com

Manuscript received April 19, 2005; revised August 26, 2015.

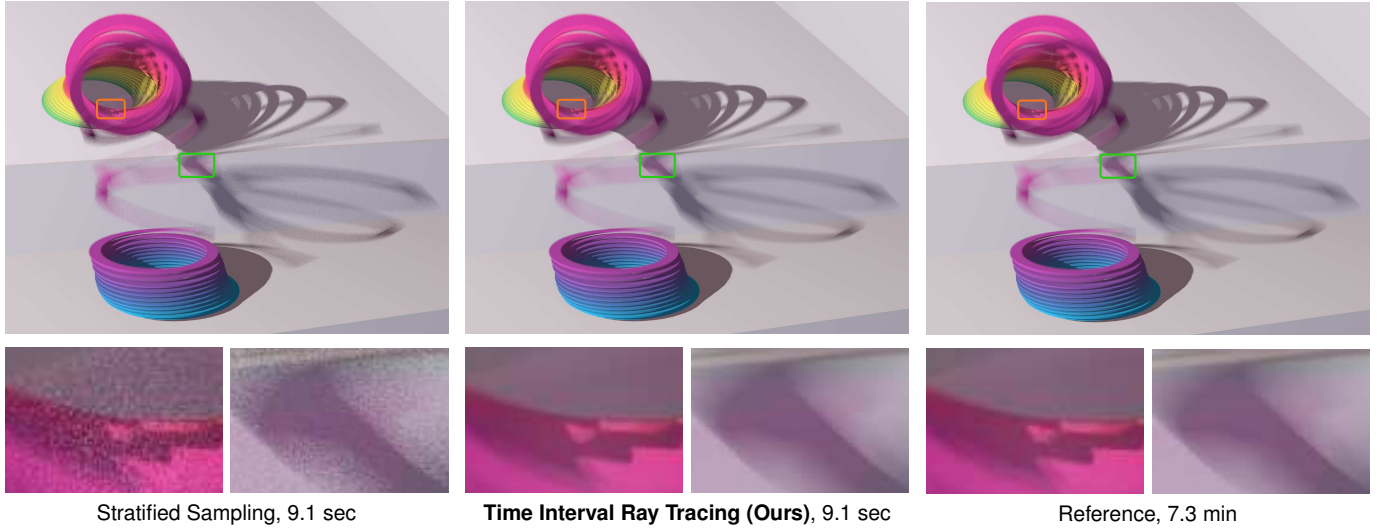


Fig. 1. A deforming slinky falling down a staircase generates complex interaction between blurred visibility and shadows. Our time interval ray tracing method produces noise-free motion blur in time similar to stratified sampling.

treatment we refer the reader to a report on the state-of-the-art by Navarro et al. [3].

Image-space techniques post-process rendered images with motion information, most commonly using per-pixel motion vectors [5], [6], [7], [8]. These algorithms are efficient to compute (especially on GPUs), and find wide use in real-time applications. Unfortunately, because they operate on rendered images, they are both inaccurate and unable to produce motion blur for secondary effects such as shadows and reflections.

Alternatively, object-space methods use object motion directly for more accurate simulation of motion blur, but at a greater expense due to maintaining and sampling the dynamic scene varying over time. Objects undergoing rigid motion can be handled simply by applying a transformation computed analytically at a particular time instant. Deformations, on the other hand, are represented by either time-dependent paths or transformations per face. Accelerating queries for mesh faces requires modifying existing acceleration structures to handle a notion of time. Object-space methods can be classified based on the underlying rendering algorithms they use: rasterization or distribution ray tracing.

Rasterization-based methods determine visibility by rasterizing the volume swept by a moving triangle [2]. These methods generally separate shading from visibility; therefore, they typically cannot account for illumination changes or secondary effects such as shadows, reflections, and refractions. Current stochastic techniques first rasterize bounding volumes of moving triangles and then evaluate whether they intersect pixel samples via ray tracing [9] or form time-dependent edge equations per-triangle [10], [11], [12] that can be extended to handle curved motion [13]. Storing *all* spatio-temporal intersections for each pixel is an important problem with rasterization and a simple compression technique has been proposed that combines neighboring intersection intervals [11]. Image-space line samples [14] can be used to compute spatio-temporal anti-aliasing and even approximate motion-blurred ambient occlusion [15]. A recent GPU algorithm uses micropolygons with an analyti-

cal visibility computation for offline rendering with motion blur [16]. Researchers have proposed augmenting graphics hardware for higher-dimensional rasterization that could seamlessly handle motion and defocus blur for primary rays [17].

Distribution ray tracing randomly selects a timestamp for each primary ray [1]. This can require a large number of samples to generate low-noise images. Furthermore, it needs on-the-fly reconstruction of the scene’s entire geometry for each timestamped ray. Glassner [18] proposed modifying existing acceleration structures to account for time dependencies of geometry and nodes. Modifications have been applied to grids [19], k-d trees [20], [21], and bounding volume hierarchies [22], [23], [24], [25], [26].

Motion blur computation can be improved by various non-uniform sampling and reconstruction strategies. Samples can be distributed over multiple dimensions by using a k-d tree to reduce error estimates and then fed into an anisotropic reconstruction filter [27]. Other methods apply anisotropic reconstruction filters to high-dimensional light-field samples [28], [29], [30], [31]. Reconstructing images from a wavelet basis can be incorporated to reduce variance [32]. Fourier domain analysis allows sampling both image and time domains adaptively before applying shear filters to reconstruct the motion-blurred image [33]. Covariance matrices storing 5D frequency information can guide this process [34]. Furthermore, it is possible to reconstruct the motion-blurred image using compressed sensing analysis on a sparse set of image samples [35]. Recently, Sun et al. [36] proposed a blue-noise sampling strategy that extends line-segment sampling [14] to incorporate motion blur.

Researchers have also proposed non-photorealistic rendering techniques for motion blur. Schmid et al. [37] built on prior ideas [2], [38] to generate motion traces. Jones and Keyser [39] proposed a method that generates additional geometry to visualize the motion silhouette.

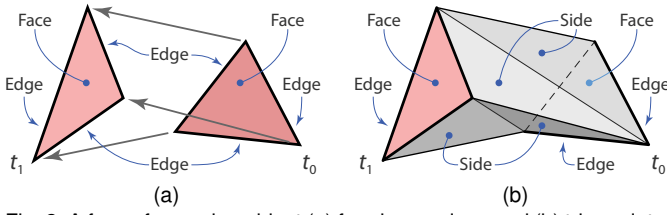


Fig. 2. A face of a moving object (a) forming a prism, and (b) triangulated sides traced by edges forming the prism surface.

### 3 SIMPLIFYING ASSUMPTIONS

Our time interval ray tracing approach for computing motion blur is based on intersecting rays with the volume swept by a dynamic triangular face (Figure 2a). We refer to this volume as a *prism*. In general, this intersection can be arbitrarily complex depending on the complexity of the motion, but virtually all motion blur computation methods rely on some basic assumptions. The traditional assumptions that are commonly used are the following:

**Assumption 1:** *Ray origins and directions remain constant in time and space.* This assumption is satisfied by primary rays in camera space and secondary rays generated on objects static in camera space. Secondary rays generated on dynamic objects, however, do not satisfy this assumption; hence, we treat them differently.

**Assumption 2:** *Each dynamic vertex has a linear motion between keyframes.* This is a common assumption employed in almost all motion blur computation methods. Additional keyframes can be introduced to reduce the discretization error associated with faces that undergo rotation or non-linear deformation.

To formulate our motion blur solution, we introduce two new assumptions that sufficiently simplify the problem:

**Assumption 3:** *The intersection (hit) point of a ray with a dynamic triangular face moves linearly over the face surface.* Similar to Assumption 2, this assumption is satisfied for linear motion involving translation and scale. Rotations and arbitrary deformations, however, can produce a non-linear intersection movement over the face. However, introducing additional keyframes reduces the discretization error. This assumption allows us to significantly simplify the intersection computation. If the movement of the intersection point is linear, we only need to compute the intersection of a ray with the surface of the prism, instead of computing the coordinates of the intersection through the prism volume. The entry point indicates where and when the ray begins intersecting with the face, and the exit point indicates where and when the intersection ends. Based on this assumption, all intersection points in-between these two points can be calculated using linear interpolation. Since the interpolation is done per-face, the interpolated intersection path approaches the ideal path as the resolution of the dynamic object increases.

**Assumption 4 (Optional):** *The bilinear patch generated by dynamic edges between two keyframes can be approximated using two triangles.* This is analogous to the triangulation of surfaces for rendering and it allows us to approximate the prism surfaces using only stationary triangles (Figure 2b). Similar to Assumptions 2 and 3, this one is perfectly satisfied

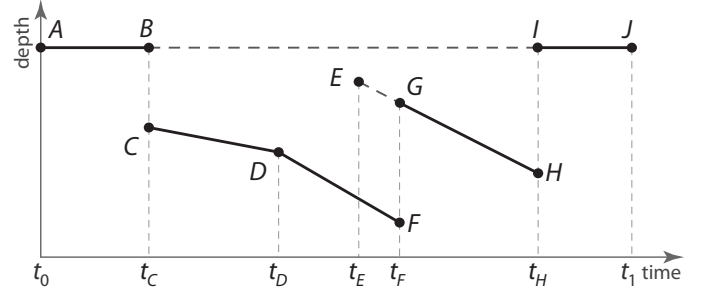


Fig. 3. An example space-time hit record shown as a depth-time graph. The hit interval A-J corresponds to a static face, which is occluded by the three dynamic faces with hit intervals C-D, D-F, and E-H. The hit interval E-H is partially occluded by D-F. The points C, D, E, F, and H correspond to ray intersection points with three prism surfaces.

for motion involving translation and scale. Using higher resolution triangulations (with more keyframes) increases the accuracy of the ray intersection with the prism sides. Note that this assumption can be eliminated by simply storing the prism sides as bilinear patches and computing intersections without triangulation [40]. Therefore, this assumption is optional, but it greatly simplifies the implementation of our method by handling all intersections as triangles without introducing a noticeable difference in practice.

### 4 TIME INTERVAL RAY TRACING

Our approach associates each camera ray with a time interval  $[t_0, t_1]$ , where  $t_0$  is the time the shutter opens and  $t_1$  is the time it closes. Hit tests with static faces are handled in the traditional way. To find the intersection with a dynamic face, rays intersect the surface triangles making up the face's prism. The hit points with a prism indicate where and when the ray begins and ends intersecting with the corresponding dynamic face. Note that the hit interval with a dynamic face does not necessarily cover a ray's entire time interval. Hence, a ray can intersect with multiple dynamic faces within its time interval. These intersections are stored in a *space-time hit record* and then shaded to compute the accumulated color of the ray within its interval.

The space-time hit record keeps a list of *hit intervals* for static and dynamic faces that intersect the ray. Figure 3 shows the depth-time graph of an example space-time hit record, where *depth* represents the distance of the hit point to the ray origin (similar to [11]). Each hit interval spans the time between when the ray begins and when it ends intersecting the corresponding face. There can be at most a single static face in the hit record that spans the entire time interval of the ray, but this *static hit interval* can be occluded by any number of *dynamic hit intervals* corresponding to dynamic faces. The end points of the dynamic hit intervals are determined by the ray intersections with the prism surfaces. All hit data between these end points are interpolated linearly based on Assumption 3. If all faces are opaque, the hit record keeps a disjoint set of hit intervals, such that no intervals overlap in time. At any time within a ray's space-time hit record, the individual hit interval closest to the ray origin occludes the others. If there are semi-transparent faces, however, hit intervals may overlap.



#### 4.1 Shading Hit Intervals

To compute the accumulated radiance of a ray within its time interval, we must integrate the space-time hit record over time by shading each hit interval. Prior methods using distribution ray tracing blindly compute this integral using Monte Carlo sampling by tracing multiple rays, each of which would find a single random hit point within the space-time hit record. In our case, however, the space-time hit record is already populated with *all* intersections of the ray within the entire time interval before we begin shading. Therefore, we can leverage this information to strategically pick the points that will be shaded to minimize the number of shading operations necessary to approximate this integral.

For static hit intervals a single shading call is sufficient. Secondary rays generated while shading a static hit interval (such as shadow, reflection, or global illumination rays) are assigned the time interval of the static hit interval. This way, we can easily compute motion-blurred secondary effects on static objects.

On the other hand, a dynamic hit interval cannot be shaded with a single shading call, as the hit information (including the hit point, texture coordinates, and surface normal) can change within the interval. Moreover, we cannot assign time intervals for the secondary rays generated on dynamic objects, because the hit location can change over time, which violates Assumption 1. Therefore, we must shade the hit interval by shading instantaneous points in time.

Shading a hit interval essentially integrates over a 1D path on a face where the ray intersects it. We employ an adaptive shading strategy to minimize the number of shading operations, while dedicating enough shading operations to approximate this integral. We begin by shading both end points of the hit interval. Let  $t_A$  and  $t_B$  be the two end point times of a hit interval A-B and  $F(\mathbf{q}, t)$  be the shading function that returns a radiance value  $\mathbf{L}$ , where  $\mathbf{q}$  is the hit information used during shading (such as the surface normal and texture coordinates). After we compute  $\mathbf{L}_A = F(\mathbf{q}_A, t_A)$  and  $\mathbf{L}_B = F(\mathbf{q}_B, t_B)$ , we decide whether to subdivide the interval based on  $\Delta\mathbf{L} = \mathbf{L}_B - \mathbf{L}_A$  and  $\Delta\mathbf{q} = \mathbf{q}_B - \mathbf{q}_A$ . If  $\Delta\mathbf{L}$  and  $\Delta\mathbf{q}$  are below user-defined error tolerances, we approximate the value of the integral as  $\Delta t(\mathbf{L}_A + \mathbf{L}_B)/2$ , where  $\Delta t = t_B - t_A$ , which corresponds to linearly changing radiance  $L$  from  $A$  to  $B$ . Otherwise, we split the hit interval into two halves at time  $t_C = (t_A + t_B)/2$ . In this case, we rely on Assumption 3 to compute  $\mathbf{q}_C$  at  $t_C$  using linear interpolation of  $\mathbf{q}_A$  and  $\mathbf{q}_B$ . To limit the level of subdivision, we stop splitting intervals when  $\Delta t$  is below a user-defined threshold  $\Delta t_{min}$ . To achieve a minimum level of subdivision, we split intervals with  $\Delta t$  above a user-defined threshold  $\Delta t_{max}$ . For all test results presented in this paper, we used only a radiance difference threshold  $\Delta\mathbf{L}_{min}$ ; we did not take the variation in shading parameters  $\Delta\mathbf{q}$  into account for subdivision decisions.

In general, the intersection point of a ray with a dynamic object moves over that object's surface with respect to time. Therefore, when the intersection point leaves a face of the object, it typically moves onto a neighboring face. As a result, two neighboring hit intervals often have a common

end point (such as the point  $D$  in Figure 3). To avoid shading the same point on the object surface multiple times, we can cache the values of  $\mathbf{L}$  at the end points shared by neighboring intervals. To reduce the number of shading operations further, we can combine multiple connected hit intervals and begin our adaptive subdivision by shading the two end points of the joint set of hit intervals (such as points  $C$  and  $F$  in Figure 3).

#### 4.2 Shutter Simulation

A mechanical camera shutter takes some time to open and close. To account for this when computing the accumulated radiance of a ray, we use a shutter response function  $S$  that indicates the percentage of incident radiance allowed to pass through the shutter at any given time. Hence, the radiance that reaches the sensor can be computed as

$$L(\mathbf{x}, t_0, t_1) = \int_{t_0}^{t_1} S(\mathbf{x}, t) L_i(\mathbf{x}, t) dt, \quad (1)$$

where  $L_i(\mathbf{x}, t)$  is the incoming radiance that arrives at time  $t$  through point  $\mathbf{x}$  on the image plane. Note that  $S$  can also be used to emulate time-varying exposure and time-dependent film/sensor response to light.

When shading hit intervals for static objects, we can move the shutter function into the shading equation. Since the hit point information for a static hit interval remains constant, we can apply the shutter function to the incoming illumination during shading. Thus, the rendering equation can be written as

$$L_o(\omega_o, t_0, t_1) = \int_{\Omega} L_{is}(\omega_i, t_0, t_1) f_s(\omega_i, \omega_o) (\omega_i \cdot \mathbf{n}) d\omega_i \quad (2)$$

with

$$L_{is}(\omega_i, t_0, t_1) = \int_{t_0}^{t_1} S(\mathbf{x}, t) L_i(\omega_i, t) dt, \quad (3)$$

where  $L_o$  is the reflected radiance,  $L_{is}$  is the incoming radiance factoring the shutter function  $S$ ,  $\Omega$  is the hemisphere,  $f_s$  is the surface BRDF with incoming  $\omega_i$  and outgoing  $\omega_o$  directions, and  $\mathbf{n}$  is the surface normal. This formulation assumes that for static faces the BRDF is static as well. Note that  $L_i$  in Equation 3 might be coming from yet another static hit interval. In that case, we can compute  $L_{is}$  similarly as in Equation 2 by applying  $S$  to the incoming light.

This leads to a very simple rule: the shutter function is applied while computing the incoming radiance of any time interval. When shading a particular timestamp (used for shading dynamic hit intervals), we do not consider the shutter function. Instead, we apply the shutter function to each dynamic interval outside of the shading call using Equation 1.

#### 4.3 Amplified Motion Blur

As an object moves faster, motion blur stretches and the object becomes less visible. Sometimes, it is desirable to exaggerate the visibility of an object's motion for artistic purposes, such as creating motion trails. In our system, we can amplify the motion blur by selectively boosting visibility



of dynamic intervals. A ray's accumulated radiance can be written as a weighted sum over the radiances of the dynamic and static intervals  $L_{dyn}$  and  $L_{stat}$ , using

$$L(\mathbf{x}, t_0, t_1) = \alpha_{dyn} L_{dyn}(\mathbf{x}, t_0, t_1) + \alpha_{stat} L_{stat}(\mathbf{x}, t_0, t_1), \quad (4)$$

where  $\alpha_{dyn}$  and  $\alpha_{stat}$  are the fractions of the shutter interval occluded by dynamic and static intervals, respectively. We amplify the visibility of the dynamic objects using a user-defined parameter  $\gamma \in (0, \infty)$ , defining instead a modified visibility as  $\alpha'_{dyn} = (\alpha_{dyn})^{1/\gamma}$ . This formulation ensures that  $\alpha'_{dyn} \in [0, 1]$  for  $\alpha_{dyn} \in [0, 1]$  regardless of the chosen  $\gamma$  value. Since this increases the total visibility of the ray, we must scale the visibility of the static objects accordingly, using  $\alpha'_{stat} = (1 - \alpha'_{dyn}) / (1 - \alpha_{dyn})$ . Note that  $\gamma = 1$  disables amplification,  $\gamma > 1$  amplifies visibility, and values  $\gamma \in (0, 1)$  weaken visibility. Note that such adjustments are also automatically applied to secondary effects (such as shadows, reflections, occlusion, and global illumination) caused by dynamic objects.

## 5 IMPLEMENTATION DETAILS

The rendering algorithm is very similar to traditional ray tracing. Rays with timestamps traverse a BVH to find the closest hit before shading it. Interval rays traverse a BVH with prism triangles (Figure 2). During traversal, each ray builds a space-time hit record (Figure 3). Then, each hit interval is shaded using adaptive subdivision. Hit intervals of static objects generate secondary rays with intervals. Hit intervals of dynamic objects are shaded at specific time points, producing secondary rays with timestamps.

While the underlying theory of our motion blur approximation is conceptually simple, an efficient implementation requires a number of non-trivial modifications to a ray-tracing-based renderer. In this section, we explain the details of these modifications.

The input to our rendering system is a collection of triangular meshes that include the positions of all dynamic vertices for all keyframes. In a typical renderer, the mesh data includes a list of faces, each with three vertex indices. Our method also generates a list of edges, each with two vertex indices and up to two face indices. Note that the prism of a face has eight triangles (Figure 2b): two of them correspond to the face at times  $t_0$  and  $t_1$  (*face triangles*), and a pair of triangles per edge form the sides of the prism (*edge triangles*). The edge list allows neighboring faces to share edge triangles so they are not duplicated and the prism triangulations are consistent across neighboring faces. All of these triangles are placed into an ordinary acceleration structure for ray tracing.

The acceleration data structure merely keeps a list of triangle indices. We use one bit of the triangle index to indicate whether it is a face or an edge triangle. If the hit triangle is a face triangle, the second bit determines whether it is the face at the beginning or end of the keyframe. If the hit triangle is an edge triangle, the second bit determines which one of the two triangles that approximate the bilinear patch is hit. The remaining bits keep the corresponding face or edge index. Thus, given a triangle index, we can determine the vertex positions of the triangle using either the face or the edge list. Note that the vertex positions of

all these triangles can be gathered directly from the input mesh data using these triangle indices. There is no need to explicitly store the vertex indices of each triangle.

During ray traversal, we keep a list of hit points as a ray intersects triangles. Each intersection with a face triangle is recorded as a single hit point associated with the face index. When the ray intersects an edge triangle, however, we must treat it differently. If the edge is in-between two faces (i.e. the edge structure stores two face indices), it means that the ray exits one prism and enters another. Therefore, we record up to two hit points: one for each of the two faces sharing the corresponding edge. We group the hit points using their face indices. For each pair of hit points with the same face index, we generate a hit interval and place it in the space-time hit record. Note that a ray can intersect with the same prism at multiple different intervals. Therefore, if there are more than two hit points associated with the same face, they are grouped in pairs ordered by the closest hit times, so that we can produce the correct set of intervals. Note that grouping by ray depth instead can result in incorrect reconstruction of the intervals in some cases.

When a ray hits a face triangle, the hit point can be found using the barycentric coordinates of the intersection, and the hit time of the intersection is merely the time of the face (either  $t_0$  or  $t_1$ ). On the other hand, when a ray hits an edge triangle, we cannot simply rely on the barycentric coordinates. This is because the barycentric coordinates on a single triangle may not be enough to approximate the bilinear coordinates of the hit point on the corresponding bilinear patch for the moving edge. Instead, we compute the hit point and time using all four vertices that define the bilinear patch. Let  $i$  and  $j$  be the two vertex indices of the edge with vertex positions  $\mathbf{v}_0^i$  and  $\mathbf{v}_0^j$  at time  $t_0$ , and  $\mathbf{v}_1^i$  and  $\mathbf{v}_1^j$  at time  $t_1$ . The hit position  $\mathbf{p}$  on the bilinear patch defined by these vertices can be written using the bilinear mapping as

$$\mathbf{p} = (1 - u) \left( (1 - v) \mathbf{v}_0^i + v \mathbf{v}_0^j \right) + u \left( (1 - v) \mathbf{v}_1^i + v \mathbf{v}_1^j \right), \quad (5)$$

where  $u$  and  $v$  are the bilinear coordinates, which correspond to time and position along a moving edge, respectively. Solving for  $u$  and  $v$  reveals the barycentric coordinates of the hit point on the dynamic face  $\lambda_{hit} = [v, 1 - v, 0]^T$  and the time of the hit  $t_{hit} = t_0 + (t_1 - t_0)u$ . Equation 5 defines three equations for two unknowns, so a closed-form solution can be found by using two of the three dimensions. However, depending on the motion and the chosen two dimensions, the solution can be numerically unstable. Pathological cases exist when the edge direction or the motion is perpendicular to both dimensions. Furthermore, since we are approximating the bilinear patch using two triangles, the intersection point on the edge triangle may not reside exactly on the bilinear patch. As a solution, in our implementation we approximate  $u$  and  $v$  using mean value coordinates [41]  $w_0^i, w_0^j, w_1^i$ , and  $w_1^j$  that correspond to  $\mathbf{v}_0^i, \mathbf{v}_0^j, \mathbf{v}_1^i$ , and  $\mathbf{v}_1^j$  respectively, such that

$$u \approx (w_1^i + w_1^j) / (w_0^i + w_0^j + w_1^i + w_1^j), \text{ and} \quad (6)$$

$$v \approx (w_0^j + w_1^j) / (w_0^i + w_0^j + w_1^i + w_1^j). \quad (7)$$

To bound the error in approximating bilinear patches as two triangles, one must consider both possible trian-

gulations of the bilinear patch. No matter the shape, the bilinear patch is contained within the volume defined by these triangulations. The maximal error between either triangulation and the bilinear patch occurs at time  $(t_0 + t_1)/2$  and evaluates to

$$e_{max} = \left\| \mathbf{v}_0^j - \mathbf{v}_0^i + \mathbf{v}_1^j - \mathbf{v}_1^i \right\| / 2. \quad (8)$$

When we insert a hit interval into the space-time hit record, we rely on depth values for occlusion culling. An inserted interval may overlap temporally with an existing interval. When the entire time interval of the ray is covered by a set of intervals in the space-time hit record, we use the maximum depth value in the hit record for early termination during ray traversal.

Our prism structure can also be used to trace rays with timestamps. In that case, it is possible that the ray origin can be inside one or more prisms, resulting in an odd number of hit points. By sending another ray in the opposite direction to find the other hit point, we can generate an interval for those prisms. Finally, we check if this interval overlaps with the ray's timestamp. Note that when tracing rays with timestamps, the space-time hit record keeps only a single hit point.

Finally, when using complex shutter functions that do not depend on the position of the sample on the image plane  $\mathbf{x}$ , we can build lookup tables to efficiently integrate dynamic hit intervals using Equation 1. Let  $L_A = L_i(\mathbf{x}, t_A)$  and  $L_B = L_i(\mathbf{x}, t_B)$  be the incoming radiance for a shaded interval A-B between times  $t_A$  and  $t_B$ . The effective radiance incorporating the shutter function

$$L(\mathbf{x}, t_A, t_B) = \int_{t_A}^{t_B} S(t) \left( L_A + (L_B - L_A) \frac{t - t_A}{t_B - t_A} \right) dt \quad (9)$$

can be written as

$$L(\mathbf{x}, t_A, t_B) = \frac{(t_B L_A - t_A L_B) \Delta S' + (L_B - L_A) \Delta S''}{t_B - t_A}, \quad (10)$$

where

$$\Delta S' = S'(t_B) - S'(t_A) = \int_{t_A}^{t_B} S(t) dt, \text{ and} \quad (11)$$

$$\Delta S'' = S''(t_B) - S''(t_A) = \int_{t_A}^{t_B} t S(t) dt. \quad (12)$$

Equation 10 can be computed quickly for any shutter function using lookup tables for  $S'(t)$  and  $S''(t)$ . For relatively simple shutter functions, the integrals in Equations 11 and 12 can be evaluated directly from a closed-form expression.

## 6 RESULTS

We tested our implementation of time interval ray tracing using a simple multi-threaded non-packetized CPU ray tracer. The performance results are from a computer with an Intel Core i7-5820K processor with 32GB RAM. Our implementation relies on two flavors of BVH to accelerate ray traversal. Rays with time intervals traverse through a

regular BVH built with prism triangles for each dynamic object and pairs of keyframes separately, using a top-down greedy builder based on SAH cost [24]. Rays with timestamps rely on an interpolating BVH [23], where each node keeps track of one bounding box for the beginning and one for the end of the motion, and are intersected with the bounds interpolated at the timestamp. Motion blur is not amplified unless otherwise specified (i.e.  $\gamma = 1$ ; see Section 4.3).

For anti-aliasing, we use adaptive subdivision sampling, similar to a popular anti-aliasing sampler within the V-Ray rendering software [42]. This anti-aliasing method strategically distributes primary ray samples to high-frequency areas of the image and, for some scenes, produces high-quality anti-aliasing with less than a single sample per-pixel on average. Our implementation uses no texture filtering; therefore, our anti-aliasing method over-samples areas with texture discontinuities, introducing some performance penalty for our method in our tests.

Figures 4 and 5 show the scenes we used for performance and quality comparisons. As expected, our time interval ray tracing method produces noise-free visibility. Moreover, due to our adaptive shading strategy, we eliminate noise in shading as well. Thus, with our method even a single sample per pixel fully resolves the motion blur with no noise. To resolve anti-aliasing with our method, extra samples per-pixel can be allocated completely independently of time. Therefore, parts of the images that correspond to fast motion can be resolved using relatively few samples per pixel. While adaptive anti-aliasing may reduce the total number of primary rays, the render time does not necessarily scale proportionally. This is because primary ray samples placed near image discontinuities to resolve anti-aliasing can be more expensive to compute than others, since they typically visit more BVH nodes and test more triangles.

In comparison, stratified sampling produces a substantial amount of noise in time approximately equal to our anti-aliased rendering results. This noise is visible in both primary visibility and secondary effects like shadows. The reference images are generated using a large number of samples at a substantial computation cost, and they mostly (but not fully, especially for large motion) resolve the noise.

The performance numbers in Figures 4 and 5 provide the necessary information for comparing the cost of time interval ray tracing to time sampling. Comparing performance in millions of rays per second (MRPS), we can see that our ray traversal is several times more expensive for most scenes ( $3\times$  on average, excluding the Dragon-Sponza scene at  $17\times$ ). On average, our time interval ray tracing method uses  $4.8\times$  more ray-box and  $1.9\times$  more ray-triangle intersection tests than stratified sampling for all scenes but Dragon-Sponza, which uses  $28\times$  and  $2.8\times$  more respectively. There are a number of factors that increase the cost of a ray sample with our method. First, the increase in triangle count and the BVH size has a relatively minor impact. More importantly, rays with timestamps merely find the first hit, but our rays with time intervals find multiple hits and shade as many intervals as necessary to resolve the motion (all shading computation is included in the cost of each ray).

One important observation is that our method with adaptive anti-aliasing substantially reduces the total num-



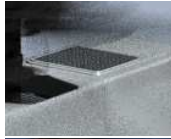
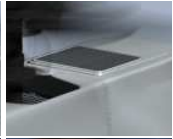







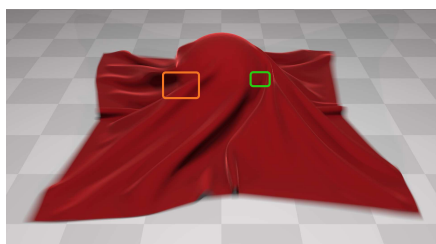









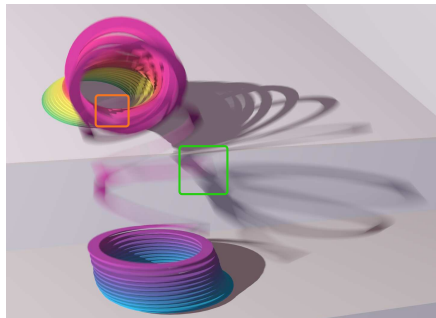






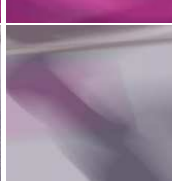

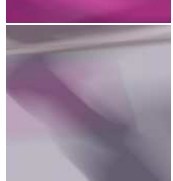
Time Interval Ray Tracing (Ours)		Stratified Sampling			Time Interval Ray Tracing		
anti-aliased		1 spp	equal-time	reference	1 spp	anti-aliased	
Helicopter							
							
	71K static & 5.8K dynamic faces	1 spp	12 spp	1K spp	1 spp	avg. 0.48 spp	
	9 keyframes	Render Time	1.25 sec	14.6 sec	20.1 min	4.98 sec	14.25 sec
	240K prism triangles	MRPS	4.83	4.96	5.01	3.27	2.70
	1920 × 1080 resolution	Total Rays	6.04M	72.5M	6.04B	16.3M	38.4M
	$\Delta L_{min} = 0.05$	Tri Tests/Ray	5.6	5.6	5.6	3.0 + 11.6	2.7 + 14.5
	$\Delta t_{min} = 0.04$	Box Tests/Ray	45.2	45.2	45.2	22.5 + 24.9	20.5 + 35.7
	$\Delta t_{max} = 1$	Shading Calls	0.82M	9.8M	819.9M	3.4M	8.2M
Clothball							
							
	100K dynamic faces	1 spp	7 spp	512 spp	1 spp	avg. 0.15 spp	
	4 keyframes	Render Time	1.2 sec	8.1 sec	9.7 min	14.1 sec	8.1 sec
	1.5M prism triangles	MRPS	6.93	7.05	7.15	3.59	3.20
	1920 × 1080 resolution	Total Rays	8.2M	57.2M	4.2B	50.7M	26M
	$\Delta L_{min} = 0.05$	Tri Tests/Ray	2.0	2.0	2.0	2.0 + 3.5	1.9 + 4.9
	$\Delta t_{min} = 0.02$	Box Tests/Ray	27.4	27.4	27.4	38.6 + 12.0	37.1 + 15.5
	$\Delta t_{max} = 0.1$	Shading Calls	2.1M	14.5M	1.1B	17.3M	9.4M
Slinky							
							
	40K dynamic faces	1 spp	21 spp	1K spp	1 spp	avg. 0.86 spp	
	2 keyframes	Render Time	0.46 sec	9.13 sec	7.3 min	3.10 sec	9.14 sec
	197K prism triangles	MRPS	6.82	7.16	7.14	1.97	2.08
	1440 × 1080 resolution	Total Rays	3.1M	65.3M	3.1B	6.1M	19.1M
	$\Delta L_{min} = 0.05$	Tri Tests/Ray	4.1	4.1	4.1	5.6 + 18.8	8.2 + 13.2
	$\Delta t_{min} = 0.003$	Box Tests/Ray	22.8	22.8	22.8	28.2 + 28.4	43.2 + 21.1
	$\Delta t_{max} = 0.2$	Shading Calls	1.6M	32.7M	1.56B	4.5M	16.2M

Fig. 4. Comparison of our time interval ray tracing method to time sampling with interpolating BVHs using Monte-Carlo sampling stratified in time. The number of intersection tests *Tri Tests/Ray* and *Box Tests/Ray* are provided as the sum of the values for *time sample rays* and *interval rays*. *MRPS* stands for millions of rays per second.



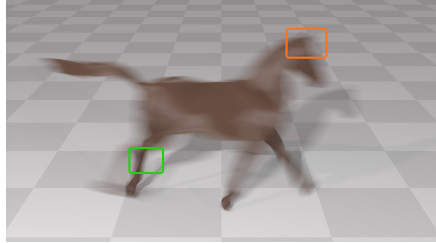



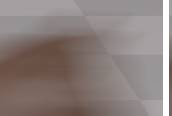
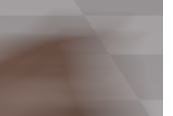
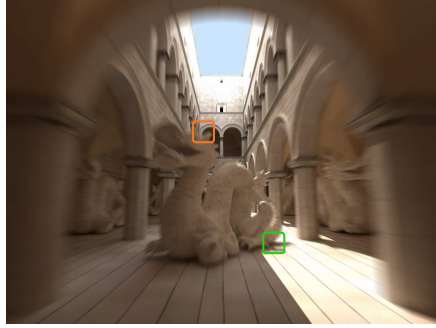





Time Interval Ray Tracing (Ours)			Stratified Sampling			Time Interval Ray Tracing	
anti-aliased			1 spp	equal-time	reference	1 spp	anti-aliased
Horse							
	17K dynamic faces		1 spp	8 spp	512 spp	1 spp	avg. 0.08 spp
	2 keyframes	Render Time	1.25 sec	4.8 sec	5.06 min	7.35 sec	4.17 sec
	84K prism triangles	MRPS	6.81	13.8	14.0	2.65	2.64
	1920 × 1080 resolution	Total Rays	8.5M	66.4M	4.25B	19.5M	11.0M
	$\Delta L_{min} = 0.05$	Tri Tests/Ray	2.19	0.87	0.87	2.32 + 15.04	2.66 + 14.42
	$\Delta t_{min} = 0.05$	Box Tests/Ray	20.98	9.58	9.58	21.66 + 18.34	25.58 + 15.67
	$\Delta t_{max} = 0.1$	Shading Calls	2.1M	16.6M	1.06B	5.8M	3.4M
Dragon-Sponza							
	6M dynamic faces		1 spp	27 spp	512 spp	1 spp	avg. 1.14 spp
	2 keyframes	Render Time	0.51 sec	13.4 sec	4.2 min	9.05 sec	13.6 sec
	31M prism triangles	MRPS	3.04	3.14	3.16	0.172	0.177
	1440 × 1080 resolution	Total Rays	1.6M	42M	796.3M	1.6M	2.4M
	$\Delta L_{min} = 0.05$	Tri Tests/Ray	3.83	3.83	3.83	0 + 108.58	0 + 108.13
	$\Delta t_{min} = 0.002$	Box Tests/Ray	102.60	102.61	102.61	0 + 284.13	0 + 296.77
	$\Delta t_{max} = 0.2$	Shading Calls	1.5M	41.0M	778.9M	44.8M	77.9M

Fig. 5. Comparison of our time interval ray tracing method to time sampling with interpolating BVHs using Monte-Carlo sampling stratified in time. The Dragon-Sponza scene features moving camera and pre-computed illumination stored in textures. The number of intersection tests *Tri Tests/Ray* and *Box Tests/Ray* are provided as the sum of the values for *time sample rays* and *interval rays*. *MRPS* stands for millions of rays per second.

ber of shading calls. Our method can fully resolve motion blur using only a fraction of the shader calls in most scenes, as compared to stratified sampling with equal render time, which fails to resolve motion blur. In fact, producing acceptably low-noise motion blur with stratified sampling requires 2 to 3 orders of magnitude more shading calls in most of our tests (except for the Dragon-Sponza scene, which requires only one order of magnitude more). Note that in our tests we used very simple shaders with low computation cost. However, in an actual production scene with expensive shaders, the shading cost can often dominate the render time [43]. Therefore, we would expect that time interval ray tracing would provide a more significant savings in render times for scenes with much more expensive shaders typically used in production.

We also compare our time interval ray tracing method to random parameter filtering (RPF) [30], an image-space reconstruction technique that can produce smooth motion blur from a noisy input sample set. RPF relies on the statistical dependency between random input parameters and ren-

dered output to apply an image-space, cross-bilateral filter to remove noise. Figure 6 shows the comparison images for the Slinky scene. We use stratified sampling with 20 spp as the input for RPF. Although RPF is effective in determining the location of the motion blur and filtering out the noise, it causes over-blur and fails to reproduce the reference image perfectly. Note that image-space filtering methods like RPF can also be used with our method for filtering various types of Monte Carlo sampling noise, though our method does not produce any noise due to motion blur.

The render time of our method not only depends on the scene but also the motion. Figure 7 shows the change in render time with increasing camera motion. As can be seen, slow motion is computed efficiently, but as the motion gets faster, and thereby the edge triangles get elongated, the efficiency of the BVH structure declines (and more shading computations are introduced). It is interesting to note that when more intermediate keyframes are introduced (with identical overall motion), even though the triangle count substantially increases, the render times can be shorter for

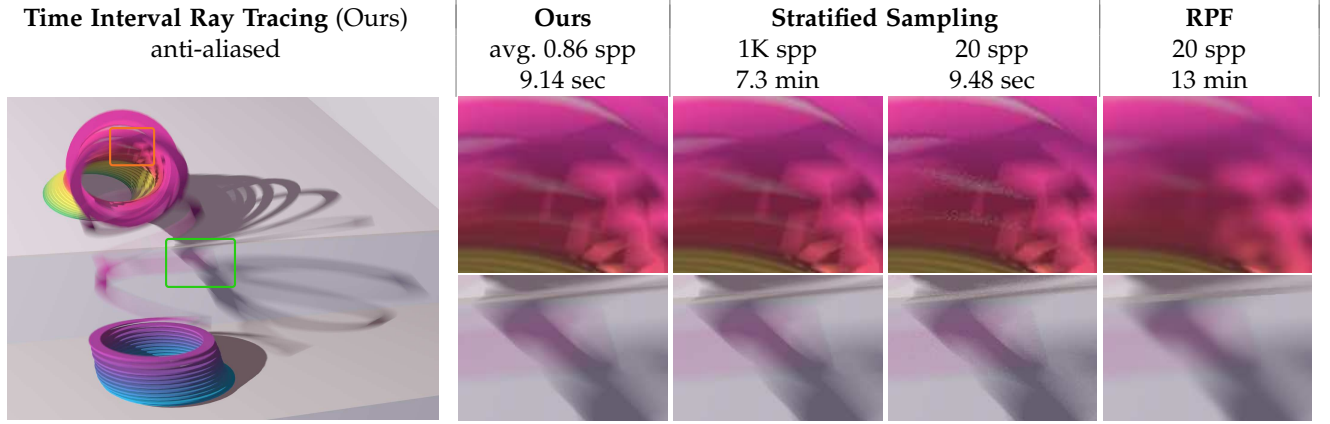


Fig. 6. Comparison of our time interval ray tracing to stratified sampling, including reconstruction via random parameter filtering (RPF) [30]. RPF uses the 20 spp stratified sampling image as input, and reconstruction takes 13 min.

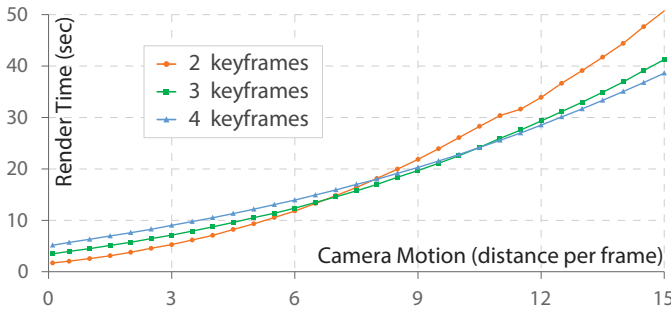


Fig. 7. Render times for different camera motions with different numbers of keyframes for the Dragon-Sponza scene.

fast motion. This shows that a BVH structure with splits [44] can produce a more efficient acceleration structure and would be particularly beneficial for our method.

Considering a dynamic object with  $F$  triangles and  $E$  edges represented using two keyframes, computing motion blur with time sampling produces  $2F$  triangles. In our approach, we generate  $2F + 2E$  triangles, which is  $5F$  triangles for closed objects. Even though these extra triangles are not stored, their indices are placed into the acceleration structure and they are intersected against rays. Therefore, the build time and the size of the acceleration data structure scale roughly linearly (about  $2.5\times$  as compared to interpolating BVHs).

We demonstrate the effect of various shutter functions in Figure 8. Notice that the shutter function also affects secondary effects such as shadows. Our method can handle any shutter function without any apparent performance penalty, including numerically challenging ones, such as the sharp peak that produces results similar to the photography trick “second-curtain flash.”

This shutter function is also featured in Figure 9, showing the effect of amplified motion blur. Notice that the trail behind the car is substantially more visible with amplified motion blur. Figure 10 shows another amplified motion blur example but using the instant shutter function with the slinky animation. In this case, the shadows cast by the slinky become substantially darker with amplified motion blur, including the self-shadowing of moving parts.

A challenging case for our method is shown in Fig-

ure 11, including half a million prisms with highly elongated triangles stretched half-way across the image. Therefore, the BVH without splits provides an extremely poor space partitioning for our method. On the other hand, this is a trivial case for interpolating BVHs, since the motion is a mere translation of the entire object. Therefore, stratified sampling can render this scene with 1,500 spp within the same two minutes it takes to render using our method, but resolving the noise in the trail requires more than an order of magnitude more rays.

Indeed, resolving the noise due to motion blur with time sampling is highly challenging for high-dynamic-range (HDR) rendering. Figure 12 shows an example where a thrown lightsaber is moving across the image. This example requires an extremely large number of samples to resolve motion blur using time sampling. In comparison, our method quickly produces noise-free results.

Since our method is based on ray tracing, it can be used directly with distribution ray tracing frameworks to compute indirect illumination or ambient occlusion. Figure 13 shows an example with ambient occlusion and glossy reflections where random ray samples generated during shading are traced using time intervals. This way, the resulting secondary effects include motion as well. The visible noise in this image comes from Monte Carlo sampling of ambient occlusion and glossy reflections.

## 7 DISCUSSION

The idea of intersecting rays with volumetric prisms has been explored in prior research, primarily in the context of rasterization. Intersections with prisms also appear in other contexts, such as computing caustic volumes using beam-tracing [45], [46], [47]. Our main contribution in this work is a framework to avoid costly volumetric intersections and allow efficient motion blur computation using intersections with merely stationary triangles stored in an ordinary acceleration structure without modifications to account for time.

The main advantage of time interval ray tracing is that it eliminates sampling along time, thereby reducing the dimensionality of rendering. This is particularly useful for adaptive under-sampling methods used for anti-aliasing. While such anti-aliasing approaches are very effective at

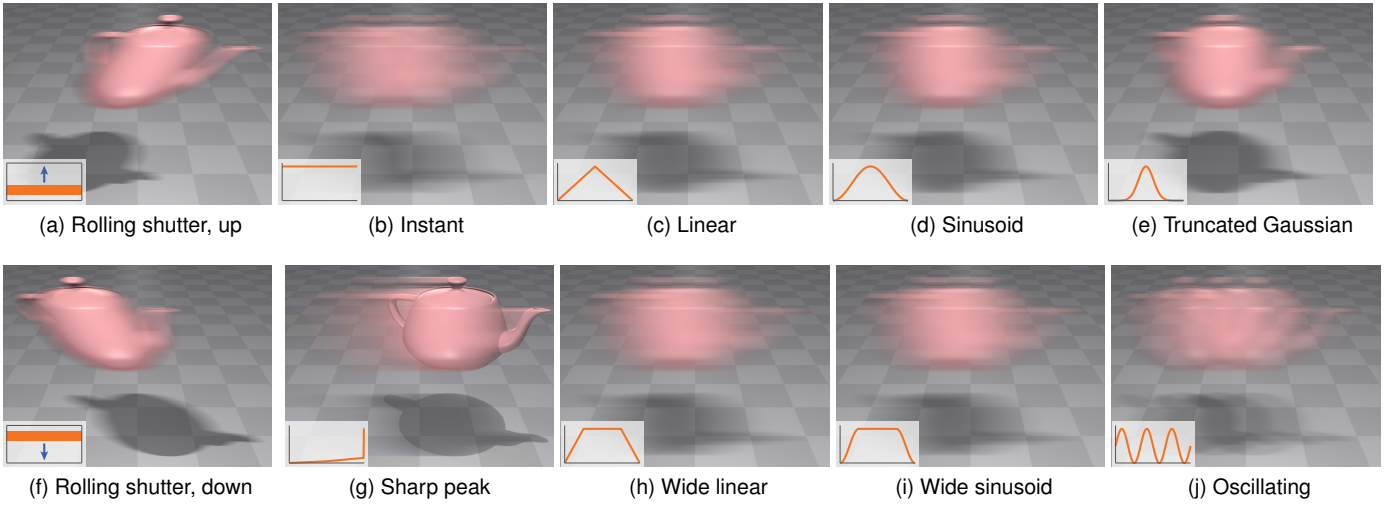


Fig. 8. This scene applies various shutter functions to a teapot moving left-to-right. Insets show the outline of each shutter function. Subfigures (a) and (f) show the effect of a rolling shutter. Subfigures (b)-(d), (h) and (i) show typical shutters used in computer graphics. More artistically-driven shutters can generate wildly varying effects, from (g) a sharp peak to (j) oscillating shutters.

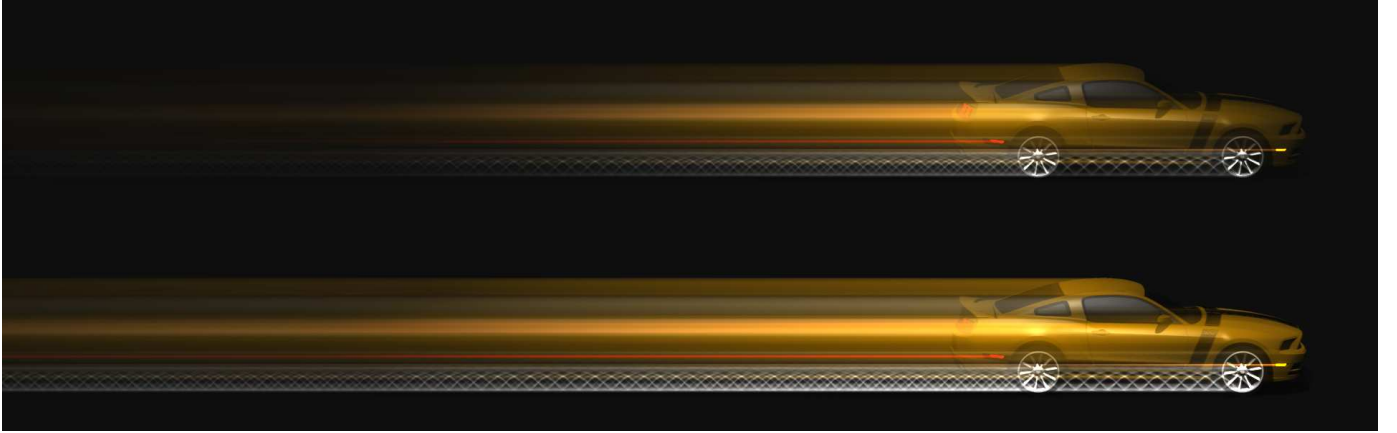


Fig. 9. Amplified motion blur: (top) no amplification with  $\gamma = 1$  and (bottom) amplification with  $\gamma = 2$ . The images were rendered using the sharp peak shutter function in Figure 8g.

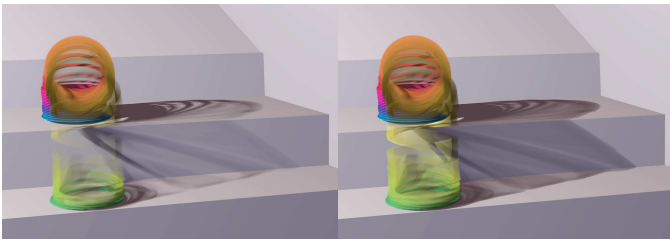


Fig. 10. Amplified motion blur: (left) no amplification with  $\gamma = 1$  and (right) amplification with  $\gamma = 3$ .

reducing the total number of primary rays without sacrificing image quality, they are only useful if each primary ray sample returns a converged result. Therefore, they cannot be paired with time-sampled motion blur.

On the other hand, if the rendering method already generates a large number of rays per pixel (such as depth of field sampling or traditional path tracing), simply assigning timestamps to each ray and changing the acceleration

structure accordingly can be good enough to resolve motion blur with low noise. However, time interval ray tracing can still be favorable for efficient path tracing implementations that minimize the number of primary rays, but introduce additional secondary rays (to avoid excessive shader calls). Furthermore, time interval ray tracing is ideal for rendering algorithms that aim to completely resolve each computed sample, such as irradiance caching [48].

Like any sampling method that uses adaptive subdivision, our shading approach can miss changes within an interval when the end points are similar. Figure 14 shows a *worst-case* scenario for our method. In this scene, the mirror on the left is stationary, so the reflections are properly resolved using reflection rays with time intervals. The mirror on the right, however, is moving along its plane, such that the reflections appear stationary in camera-space. Since the mirror on the right is a dynamic object, the reflections are handled using rays with timestamps. Thus, the motion blur in this reflection is *not* computed with time interval ray tracing, but with adaptive shading only. As a result, when



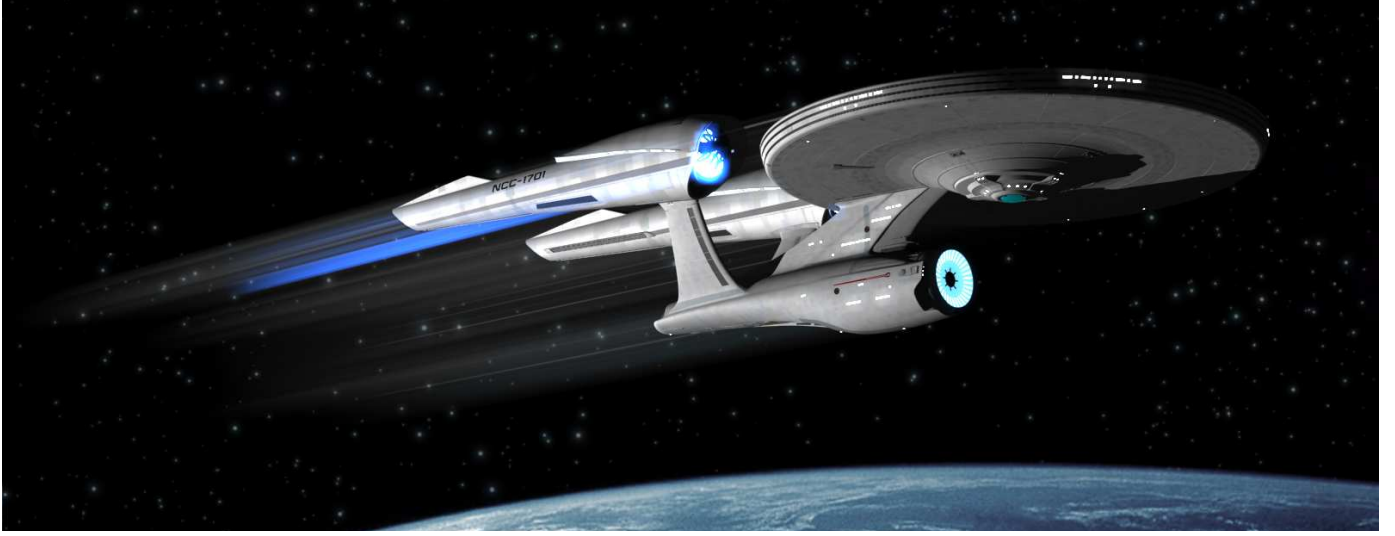


Fig. 11. A space ship coming out of warp speed, including half a million highly elongated prisms stretched half-way across the image. Model from Ricky “MadMan1701A” Wallace ([www.madshipyard.com](http://www.madshipyard.com)).

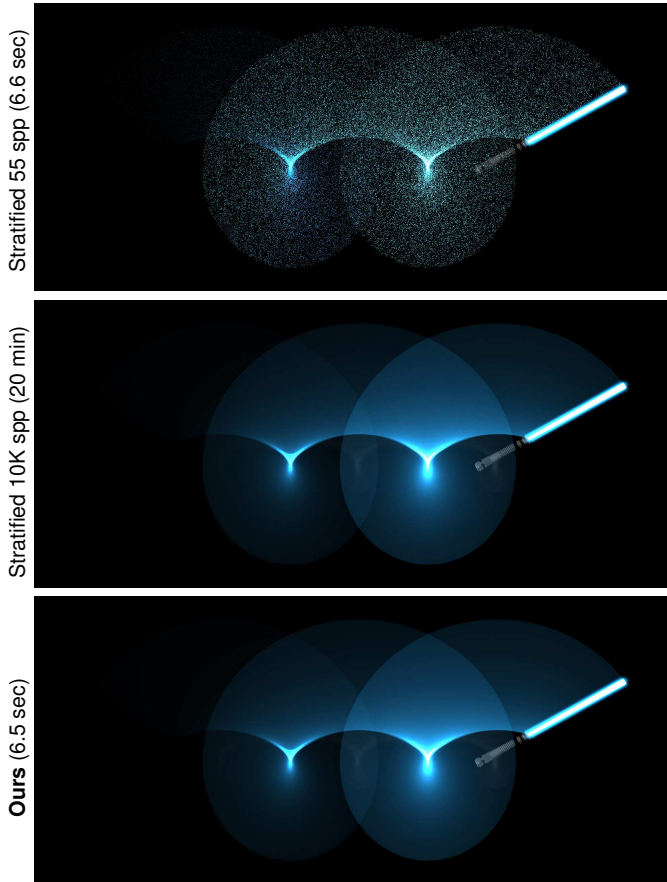


Fig. 12. A lightsaber that moves across the image while rotating around its center of mass. It is frozen in the air in the last quarter of the time interval. There are 128 keyframes and the images include an image-space bloom effect as a post-process, producing the glow around the lightsaber.

both reflection rays originating at the two endpoints of the hit interval miss the moving teapot or its shadow, the hit interval is not subdivided and the motion is missed (circled areas in Figure 14-left). Yet, such exceptional cases can be easily resolved by subdividing all intervals that are longer

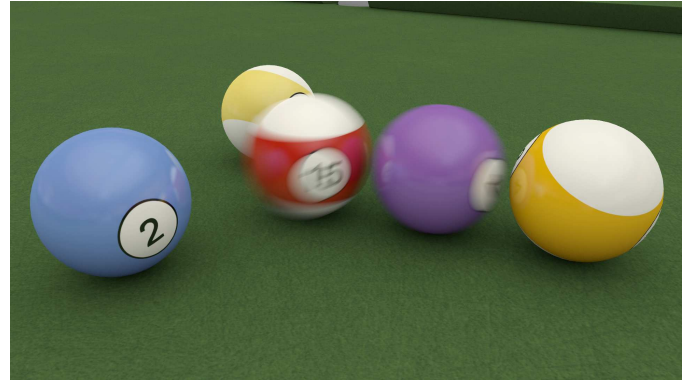


Fig. 13. Moving billiard balls rendered using our method with motion-blurred ambient occlusion and glossy reflections.

than a user-defined threshold, regardless of the differences between the end points, as shown in Figure 14-right.

Our assumptions allow us to aggressively simplify the motion blur computation, but they do not impose additional restrictions on the types of motion that can be represented properly. Similar to virtually all motion blur algorithms, handling non-linear motion and rotation requires additional keyframes. If an insufficient number of keyframes is provided, the surface of the triangulated prism (Figure 2b) can substantially deviate from the actual prism (Figure 2a) and lead to incorrect intervals. To demonstrate this, we prepared an extreme example, where a triangle is both substantially rotated and translated, shown in Figure 15. Notice that in this example our triangulated prism produces incorrect visibility with few keyframes, as compared to the reference generated using time sampling. This is because the triangulation of the bilinear patches forming the sides of the prism does not have enough resolution to properly approximate the shapes of the bilinear patches.

As a solution, we can compute the intersections with the prism by directly using the bilinear patches [40], which produces results identical to the reference, as it completely eliminates our Assumption 4. In Figure 16, we show more

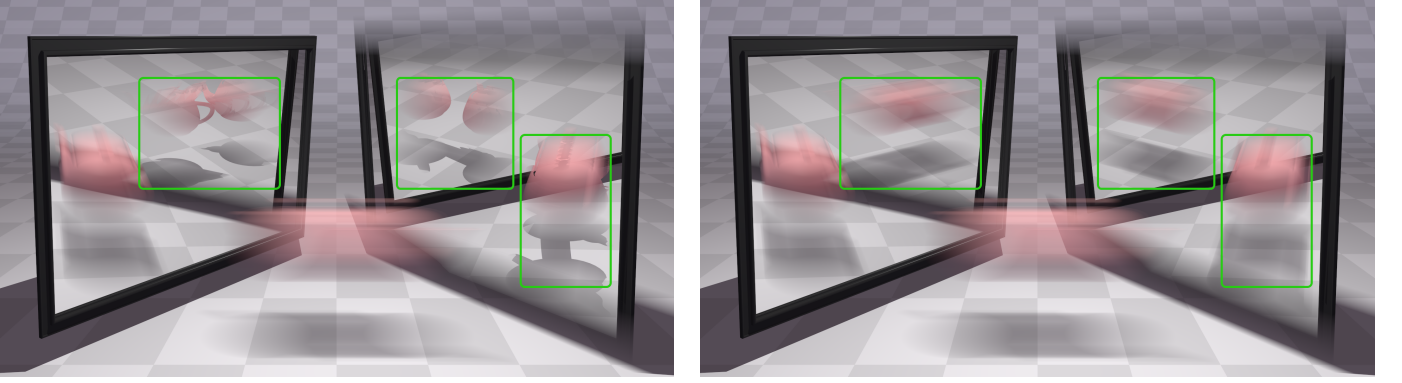


Fig. 14. A moving teapot reflecting in two mirrors: the left one is stationary and the right one is moving downward. The moving mirror provides a difficult case for our method because the reflection is stationary in camera-space but must be resolved by adaptive shading using discrete timestamps (left). Enforcing subdivision for hit intervals that are longer than a user-defined threshold  $\Delta t_{max}$  fixes the artifacts (right).

examples from our stress test with extremely deformed prisms, which are uncommon in typical scenes. Notice that using bilinear patches, our method produces results virtually identical to the stratified sampling reference. Triangulation (using Assumption 4) fails to match the outcome of the reference images, but the inaccuracies it produces would be extremely difficult to notice in a scene even if such an extreme deformation exists. Regardless, none of these methods (including the stratified sampling reference) can handle such extreme deformations correctly without introducing more keyframes to properly represent rotational motion, which would also eliminate the extreme deformation of the prisms.

In practice, however, such extreme motions are unlikely to take place within a single frame of an animation. When the deformation of the bilinear patch is not as extreme, our triangulation provides a good approximation, as evident in our test results including numerous faces that go through non-linear motion (the slinky, helicopter, clothball, and horse scenes). To evaluate the effect of triangulation (Assumption 4) we use mean structural similarity (MSSIM) [49] that provides a localized perceptual error metric. Generally, MSSIM values above 0.97 – 0.99 indicate images that are visually indistinguishable. In our tests, we found that the MSSIM values comparing results with and without Assumption 4 for every example in Figures 4 and 5 are above 0.999. This indicates that triangulation is indeed an acceptable assumption in practice.

An application of our method that would like to handle extreme motion, as in Figure 15, can use either higher-resolution triangulations or ray intersections directly with the bilinear patches. Although excluded from our implementation, it is also possible to automatically detect the triangulation error (Equation 8) and use higher-resolution triangulation or bilinear patches only for some edges in the scene, eliminating any potential error due to Assumption 4.

On the other hand, we found no evidence that any visible artifacts are linked to our Assumption 3, which allows calculating the hit point data by interpolating the two end points of an interval. In fact, the error introduced by this assumption is correlated to the resolution of the moving object, and the related triangulation error exists for all rendering algorithms that use triangles.

Our time interval ray tracing method can produce motion blur due to camera motion by treating all objects as dynamic, since they are dynamic in camera-space (as in Figure 5, Dragon-Sponza). However, when it comes to animated lights, our method must also rely on time sampling to compute blur in shadows and shading from these lights. Similarly, we cannot handle blur due to the changes of the internal camera parameters (such as field of view) without sampling them.

## 8 CONCLUSION

We have introduced an efficient method for ray tracing using time intervals to produce noise-free motion blur for both primary and secondary rays. Most significantly, our simplifying assumptions reduce the problem of motion blur computation to ray intersections with stationary triangles, which permits using any traditional acceleration structure without modifications to account for the notion of time. We have also described an adaptive shading strategy for shading dynamic hit intervals, a mathematical framework for incorporating any shutter function, and a simple modification for amplifying the visibility of dynamic objects for artistic purposes. By separating the time dimension from sampling, we have shown that our method can effectively use adaptive under-sampling for anti-aliasing. Our time interval ray tracing approach can produce high-quality images with minimal primary rays and a reduced number of shading calls for a variety of animations, including objects undergoing rapid deformations.

## ACKNOWLEDGMENTS

This material is supported in part by the National Science Foundation under Grant No. 1409129. Thiago Ize and Peter Shirley provided helpful feedback. Cem Yuksel provided Slinky, Clothball, and Lightsaber scenes, and combined Sponza atrium by Marko Dabrovic with the Stanford Dragon for the Dragon-Sponza scene. We also thank the anonymous reviewers for their time and helpful feedback.

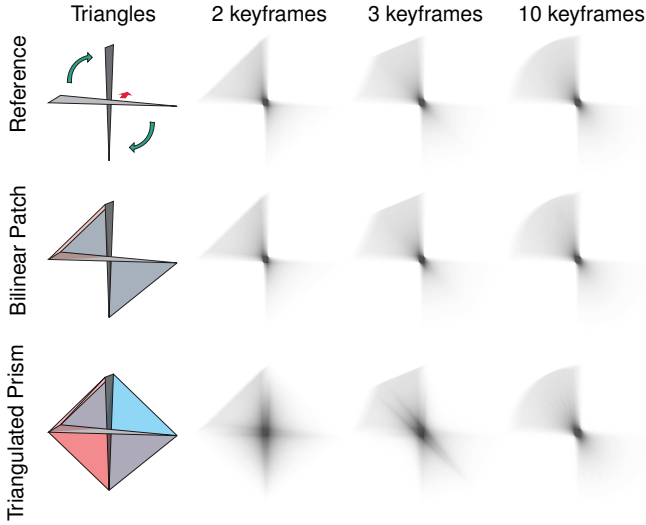


Fig. 15. A triangle rotating clockwise and moving into the page shows a possible worst-case scenario for triangulating prism sides: (top) reference using time sampling, (middle) time interval ray tracing using bilinear patches to avoid the triangulation in Assumption 4, and (bottom) time interval ray tracing using triangulated bilinear patches based on Assumption 4. In this case, the error in triangulating the prism sides introduces incorrect visibility with our method due to Assumption 4. This problem is mitigated by introducing additional keyframes, which is already necessary to properly resolve the motion.

## REFERENCES

- [1] R. L. Cook, T. Porter, and L. Carpenter, “Distributed ray tracing,” in *Proceedings of SIGGRAPH*, 1984, pp. 165–174.
- [2] C. W. Grant, “Integrated analytic spatial and temporal anti-aliasing for polyhedra in 4-space,” *SIGGRAPH Comput. Graph.*, vol. 19, no. 3, pp. 79–84, Jul. 1985.
- [3] F. Navarro, F. J. Sern, and D. Gutierrez, “Motion blur rendering: State of the art,” *Comput. Graph. Forum*, vol. 30, no. 1, 2011.
- [4] K. Sung, A. Pearce, and C. Wang, “Spatial-temporal antialiasing,” *IEEE Transactions on Visualization and Computer Graphics*, vol. 8, no. 2, pp. 144–153, Apr. 2002.
- [5] M. Potmesil and I. Chakravarty, “Modeling motion blur in computer-generated images,” *SIGGRAPH Comput. Graph.*, vol. 17, no. 3, pp. 389–399, Jul. 1983.
- [6] M. McGuire, P. Hennessy, M. Bukowski, and B. Osman, “A reconstruction filter for plausible motion blur,” in *Symposium on Interactive 3D Graphics and Games*, ser. I3D ’12, 2012.
- [7] J.-P. Guertin, M. McGuire, and D. Nowrouzezahrai, “A fast and stable feature-aware motion blur filter,” in *High Performance Graphics*, ser. HPG ’14, June 2014.
- [8] J.-P. Guertin and D. Nowrouzezahrai, “High Performance Non-linear Motion Blur,” in *Eurographics Symposium on Rendering - Experimental Ideas & Implementations*, 2015.
- [9] M. McGuire, E. Enderton, P. Shirley, and D. Luebke, “Real-time stochastic rasterization on conventional GPU architectures,” in *High Performance Graphics*, ser. HPG ’10, 2010.
- [10] T. Akenine-Möller, J. Munkberg, and J. Hasselgren, “Stochastic rasterization using time-continuous triangles,” in *Symposium on Graphics Hardware*, ser. GH ’07, 2007, pp. 7–16.
- [11] C. J. Gribel, M. Doggett, and T. Akenine-Möller, “Analytical motion blur rasterization with compression,” in *High Performance Graphics*, ser. HPG ’10, 2010, pp. 163–172.
- [12] J. Munkberg, P. Clarberg, J. Hasselgren, R. Toth, M. Sugihara, and T. Akenine-Möller, “Hierarchical stochastic motion blur rasterization,” in *High Performance Graphics*, ser. HPG ’11, 2011, pp. 107–118.
- [13] C. J. Gribel, J. Munkberg, J. Hasselgren, and T. Akenine-Möller, “Theory and analysis of higher-order motion blur rasterization,” in *High Performance Graphics*, ser. HPG ’13, 2013, pp. 7–15.
- [14] T. R. Jones and R. N. Perry, “Antialiasing with line samples,” in *Rendering Techniques 2000*, ser. Eurographics, B. Proche and H. Rushmeier, Eds. Springer Vienna, 2000, pp. 197–205.

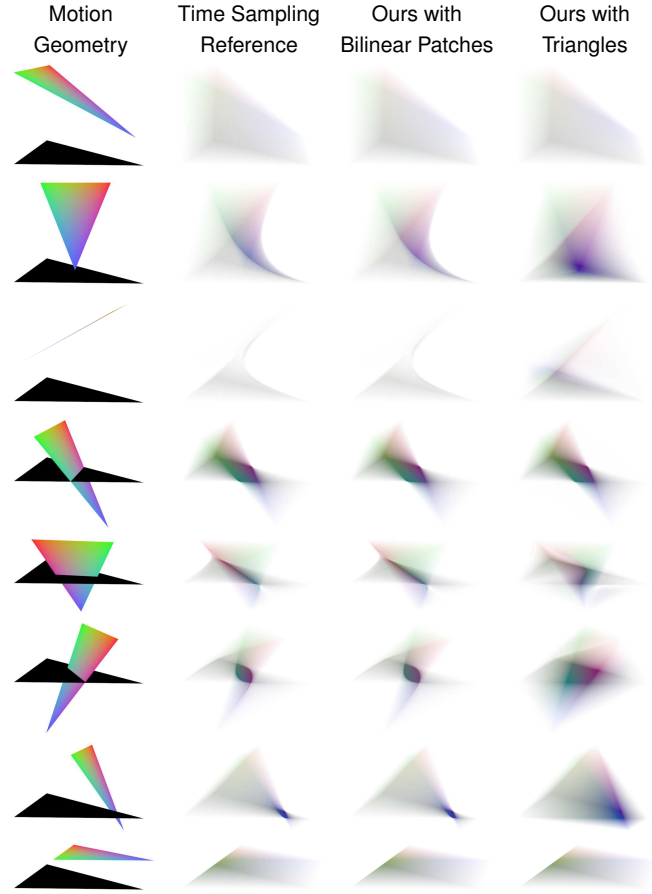


Fig. 16. A selection of frames from the stress test, where a single triangle undergoes different transformations, each over the duration of a single frame. The *Motion Geometry* column shows the triangle keyframes, where the black triangle indicates the color and the position of the triangle at the start of the motion, and the colored triangle indicates the end of the motion. The *Time Sampling Reference* column shows the image when using stratified sampling. The last two columns show the results of our method using bilinear patches and triangulated prisms. Using the bilinear patches produces results virtually identical to the reference. While triangulation provides reasonable results for majority of the tested motions, in the extreme cases shown here it can deviate from the reference.

- [15] C. J. Gribel, R. Barringer, and T. Akenine-Möller, “High-quality spatio-temporal rendering using semi-analytical visibility,” *ACM Trans. Graph.*, vol. 30, no. 4, Jul. 2011.
- [16] X. Huang, Q. Hou, Z. Ren, and K. Zhou, “Scalable programmable motion effects on GPUs,” *Computer Graphics Forum*, vol. 31, no. 7, pp. 2259–2266, 2012.
- [17] J. Nilsson, P. Clarberg, B. Johnsson, J. Munkberg, J. Hasselgren, R. Toth, M. Salvi, and T. Akenine-Möller, “Design and novel uses of higher-dimensional rasterization,” in *Proceedings of High-Performance Graphics*, ser. HPG’12, 2012, pp. 1–11.
- [18] A. Glassner, “Spacetime ray tracing for animation,” *Computer Graphics and Applications, IEEE*, vol. 8, no. 2, pp. 60–70, March 1988.
- [19] I. Wald, T. Ize, A. Kensler, A. Knoll, and S. Parker, “Ray tracing animated scenes using coherent grid traversal,” in *ACM SIGGRAPH*, vol. 25, 2006, pp. 485–493.
- [20] J. Günther, H. Friedrich, I. Wald, H.-P. Seidel, and P. Slusallek, “Ray tracing animated scenes using motion decomposition,” *Computer Graphics Forum*, vol. 25, no. 3, Sep. 2006.
- [21] P. Djeu, W. Hunt, R. Wang, I. Elhassan, G. Stoll, and W. R. Mark, “Razor: An architecture for dynamic multiresolution ray tracing,” Department of Computer Sciences, The University of Texas at Austin, Tech. Rep. TR-07-52, January 2007.
- [22] C. Lauterbach, S.-E. Yoon, D. Tuft, and D. Manocha, “RT-DEFORM: Interactive ray tracing of dynamic scenes using BVHs,” in *Interactive Ray Tracing (IRT06)*, 2006.



- [23] P. H. Christensen, J. Fong, D. M. Laur, and D. Batali, "Ray tracing for the movie 'Cars'," in *Interactive Ray Tracing IRT06*, September 2006, pp. 1–6.
- [24] I. Wald, S. Boulos, and P. Shirley, "Ray Tracing Deformable Scenes using Dynamic Bounding Volume Hierarchies," *ACM Transactions on Graphics*, vol. 26, no. 1, 2007.
- [25] Q. Hou, H. Qin, W. Li, B. Guo, and K. Zhou, "Micropolygon ray tracing with defocus and motion blur," in *ACM SIGGRAPH*, 2010, pp. 64:1–64:10.
- [26] L. Grünschloß, M. Stich, S. Nawaz, and A. Keller, "MSBVH: An efficient acceleration data structure for ray traced motion blur," in *High Performance Graphics*, ser. HPG '11, 2011.
- [27] T. Hachisuka, W. Jarosz, R. P. Weistroffer, K. Dale, G. Humphreys, M. Zwicker, and H. W. Jensen, "Multidimensional adaptive sampling and reconstruction for ray tracing," in *ACM SIGGRAPH*, 2008, pp. 33:1–33:10.
- [28] J. Lehtinen, T. Aila, J. Chen, S. Laine, and F. Durand, "Temporal light field reconstruction for rendering distribution effects," in *ACM SIGGRAPH 2011 Papers*, ser. SIGGRAPH '11, 2011, pp. 55:1–55:12.
- [29] J. Lehtinen, T. Aila, S. Laine, and F. Durand, "Reconstructing the indirect light field for global illumination," *ACM Trans. Graph.*, pp. 51:1–51:10, 2012.
- [30] P. Sen and S. Darabi, "On filtering the noise from the random parameters in Monte Carlo rendering," *ACM Trans. Graph.*, vol. 31, no. 3, pp. 18:1–18:15, Jun. 2012.
- [31] J. Munkberg, K. Vaidyanathan, J. Hasselgren, P. Clarberg, and T. Akenine-Möller, "Layered reconstruction for defocus and motion blur," *Computer Graphics Forum*, vol. 33, no. 4, pp. 81–92, 2014.
- [32] R. S. Overbeck, C. Donner, and R. Ramamoorthi, "Adaptive wavelet rendering," in *ACM SIGGRAPH Asia*, 2009, pp. 140:1–140:12.
- [33] K. Egan, Y.-T. Tseng, N. Holzschuch, F. Durand, and R. Ramamoorthi, "Frequency analysis and sheared reconstruction for rendering motion blur," in *SIGGRAPH 2009*, 2009.
- [34] L. Belcour, C. Soler, K. Subr, N. Holzschuch, and F. Durand, "5D covariance tracing for efficient defocus and motion blur," *ACM Trans. Graph.*, vol. 32, no. 3, pp. 31:1–31:18, Jul. 2013.
- [35] P. Sen and S. Darabi, "Compressive estimation for signal integration in rendering," *Computer Graphics Forum*, vol. 29, no. 4, 2010.
- [36] X. Sun, K. Zhou, J. Guo, G. Xie, J. Pan, W. Wang, and B. Guo, "Line segment sampling with blue-noise properties," *ACM Trans. Graph.*, vol. 32, no. 4, pp. 127:1–127:14, Jul. 2013.
- [37] J. Schmid, R. W. Sumner, H. Bowles, and M. Gross, "Programmable motion effects," in *ACM SIGGRAPH*, ser. SIGGRAPH '10, 2010, pp. 57:1–57:9.
- [38] J. Korein and N. Badler, "Temporal anti-aliasing in computer generated animation," *SIGGRAPH Comput. Graph.*, vol. 17, no. 3, pp. 377–388, Jul. 1983.
- [39] N. Jones and J. Keyser, "Real-time geometric motion blur for a deforming polygonal mesh," in *Computer Graphics International*, ser. CGI '05, 2005, pp. 14–18.
- [40] S. Ramsey, K. Potter, and C. Hansen, "Ray bilinear patch intersections," *Journal of Graphics Tools*, vol. 9, no. 3, pp. 41–47, 2004.
- [41] M. S. Floater, "Mean value coordinates," *Computer Aided Geometric Design*, vol. 20, no. 1, pp. 19–27, 2003.
- [42] Chaos Group, "V-ray documentation: Image sampler & anti-aliasing," <http://docs.chaosgroup.com/pages/viewpage.action?pageId=7897184>, 2015.
- [43] C. Eisenacher, G. Nichols, A. Selle, and B. Burley, "Sorted deferred shading for production path tracing," *Computer Graphics Forum*, vol. 32, no. 4, 2013.
- [44] M. Stich, H. Friedrich, and A. Dietrich, "Spatial splits in bounding volume hierarchies," in *High Performance Graphics*, ser. HPG '09, 2009, pp. 7–13.
- [45] M. Watt, "Light-water interaction using backward beam tracing," in *Proceedings of the 17th Annual Conference on Computer Graphics and Interactive Techniques*, ser. SIGGRAPH '90, 1990, pp. 377–385.
- [46] M. Ernst, T. Akenine-Möller, and H. W. Jensen, "Interactive rendering of caustics using interpolated warped volumes," in *Proceedings of Graphics Interface 2005*, ser. GI '05, 2005, pp. 87–96.
- [47] G. Liktov and C. Dachsbacher, "Real-time volume caustics with adaptive beam tracing," in *Symposium on Interactive 3D Graphics and Games*, ser. I3D '11, 2011, pp. 47–54.
- [48] G. J. Ward, F. M. Rubinstein, and R. D. Clear, "A ray tracing solution for diffuse interreflection," in *ACM SIGGRAPH*, 1988, pp. 85–92.

- [49] Z. Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli, "Image quality assessment: from error visibility to structural similarity," *IEEE Transactions on Image Processing*, vol. 13, no. 4, 2004.



**Konstantin Shkurko** received the BA in mathematics and physics and an MS in computer graphics from Cornell University, in 2007 and 2010, respectively. He is currently working toward the PhD in computer graphics from the School of Computing at the University of Utah. His research focuses mainly on ray tracing hardware, but also includes acceleration structures, rendering algorithms, and scientific visualization.



**Cem Yuksel** is a faculty member in the School of Computing at the University of Utah. Previously, he was a postdoctoral fellow at Cornell University, after receiving his PhD in Computer Science from Texas A&M University in 2010. His research interests are in computer graphics and related fields, including physically-based simulations, rendering techniques, global illumination, sampling, GPU algorithms, graphics hardware, knitted structures, and hair modeling, animation, and rendering.



**Daniel Kopta** received his PhD from the University of Utah in 2016, researching ray-traced computer graphics and GPU architecture. Since then, he has worked as a Senior OptiX engineer at NVIDIA, developing the OptiX GPU ray tracing framework. He is currently a faculty member in the School of Computing at the University of Utah.



**Ian Mallett** received BS degrees in computer science and pure mathematics from the University of New Mexico in 2014. He is currently working toward a PhD in computer graphics at the School of Computing, University of Utah. His research focuses on rendering algorithms and light transport, with excursions to ray tracing hardware.



**Erik Brunvand** Received his PhD from Carnegie Mellon University in 1990. Since then he has been a faculty member in the School of Computing at the University of Utah where his interests include the design of application-specific computers, graphics processors, physical computing, asynchronous systems, VLSI integrated circuit design, and arts/technology collaboration and integration in both research and education.