Volume xx (200y), Number z, pp. 1–15

Time-Continuous Quasi-Monte Carlo Ray Tracing

C. J. Gribel[1,2] and T. Akenine-Möller[1,3]

[1]Lund University [2]Malmö University [3]Intel Corporation

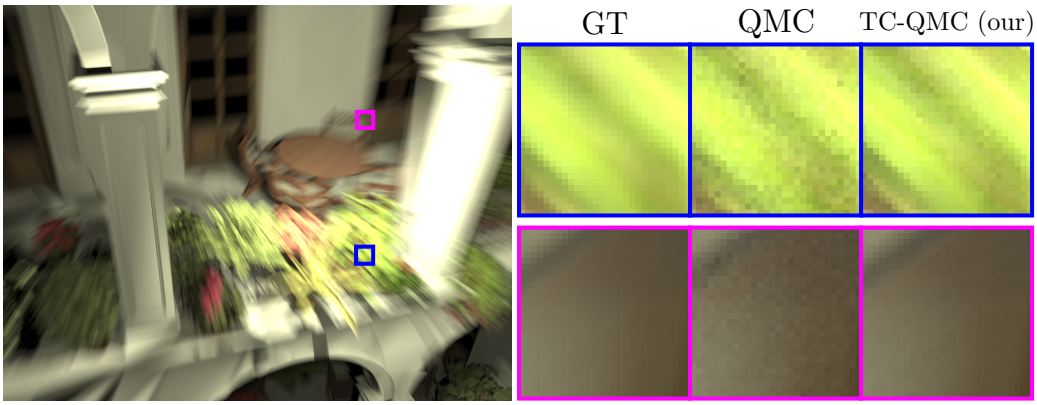


Figure 1: *This figure shows an equal-time comparison for the* SAN-MIGUEL *scene with motion blur. We compare ground truth (GT) images rendered with 2048 samples per pixel against multi-jitter quasi-Monte Carlo (QMC) rendering and against our time-continuous quasi-Monte Carlo (TC-QMC) approach. Note that the TC-QMC images are essentially noise-free in terms of visibility and so the remaining noise comes from shading evaluation, which is point sampled. QMC obtains 38.1 dB in image quality and was rendered in 22.7 seconds using 64 shading samples per pixel. Using our method (TC-QMC), the PSNR becomes 40.75 dB and rendering time 22.1 seconds using 66.8 shading samples on average, which is a substantially better result.*

Abstract

Domain-continuous visibility determination algorithms have proved to be very efficient at reducing noise otherwise prevalent in stochastic sampling. Even though they come with an increased overhead in terms of geometrical tests and visibility information management, their analytical nature provides such a rich integral that the pay-off is often worth it. This paper presents a time-continuous, primary visibility algorithm for motion blur aimed at ray tracing. Two novel intersection tests are derived and implemented. The first is for ray vs. moving triangle and the second for ray vs. moving AABB intersection. A novel take on shading is presented as well, where the time continuum of visible geometry is adaptively point sampled. Static geometry is handled using supplemental stochastic rays in order to reduce spatial aliasing. Finally, a prototype ray tracer with a full time-continuous traversal kernel is presented in detail. The results are based on a variety of test scenarios and show that even though our time-continuous algorithm has limitations, it outperforms multi-jittered quasi-Monte Carlo ray tracing in terms of image quality at equal rendering time, within wide sampling rate ranges.

Categories and Subject Descriptors (according to ACM CCS): I.3.3 [Computer Graphics]: Picture/Image Generation—Display algorithms I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Raytracing

1. Introduction

Motion blur has a profound impact in synthesized imagery and is increasingly indispensable in off-line and real-time rendering alike. Used right, motion blur helps immersing the viewer in the image while providing intuition about what moves and where. One of the major challenges with motion blur, as with other higher domain effects such as depth-

of-field, is visibility determination. As geometry moves, the geometrical content seen through a pixel increases and gets harder to determine. This problem gets substantially more difficult when the geometry is thin or complex, which is the case for, e.g., foliage and latticework. Two main approaches are available when determining temporal visibility, namely, using point sampling, e.g., quasi-Monte Carlo (QMC) [KPR12], and analytical approaches. In QMC, time is treated as a stochastic variable with a low-discrepancy distribution, and the geometry is tested at each and every one of these samples. QMC is simple and extremely general – most modern ray tracers utilize QMC as a universal integration scheme over all dimensions such as space, time, lens, area light source, etc – but may converge slowly for high frequency input. In analytical approaches, time is instead treated as a continuum, which provides a solution of temporal visibility on closed form. Analytical approaches are typically more computationally expensive and require management of visibility data, but have proved very potent in producing high quality results due to its closed form, both for motion blur rasterization [GDAM10, GBAM11] but also for depth of field rasterization [TPD*12]. Analytical approaches have been explored previously in the context of rasterization, but our work is, to the best of our knowledge the first in-depth investigation of analytical, temporal visibility for ray tracing. Our contributions include novel world space ray intersection tests and a prototype ray tracer able to trace both stochastically and analytically depending on scene content.

One advantage of ray tracing compared to rasterization, in the context of analytical visibility, is memory requirements. In general, analytical approaches produce significant amounts of visibility data (unbounded, in fact). While a rasterizer must store this data in memory for each z-buffer sample over a tile or even the entire framebuffer, a ray tracer only needs to store this data per allocated ray. Though this number between allocated z-buffer samples and rays (in streams or packets for instance) may differ between implementations, the former may well be an order of magnitude or more larger, which reduces relative memory requirements for ray tracing. Another important benefit in the context of time-continuous vs. time-discrete visibility is that visibility determination is decoupled from shading sampling. Once a visibility set has been obtained, shading can be sampled arbitrarily over the geometry without need for further traversal of the data structure. This benefit reveals itself in our results, where the cost of our method increases more slowly compared to QMC with the number of shading samples.

We show that analytical temporal visibility, despite being fundamentally different from QMC in many ways, can be readily integrated into a high-end ray tracer and outperform regular QMC under certain, though not all, circumstances. Our aim is to provide an as clear picture of this algorithm as possible, both in terms of benefits and drawbacks. Our contributions include the following:

- Derivation of two new geometrical intersection tests:
  - Ray vs. moving triangle with temporal segments.
  - Ray vs. moving AABB with temporal bounds.
- A novel time-space, feature-aware scheme for shading sampling.
- A prototype ray tracer with mixed time-discrete and time-continuous visibility for static and dynamic geometry.

Paper disposition Derivations of the required geometrical tests are made in Section 3. This section also includes details of how visibility information is managed. Section 4 explains how shading is sampled over a pixel, both in time and spatially. Section 5 contains an overview of our prototype ray tracer, including details about how the ray kernel is set up, traversal etc. Finally, results, conclusions, and future work are presented in Sections 6 and 7.

2. Previous Work

There is a lot of work already done in motion blur research and we refer the reader to the overview by Sung et al. [SPW02] as well as to the state-of-the-art report by Navarro et al. [NSG11].

The accumulation buffer technique [HA90] on the GPU is a simple method to get motion blur and other effects, but the sampling is not very efficient since an entire image is rendered from each point in time, for example, that you want to sample. Stochastic sampling [CPC84, CCC87] is then often a much better alternative, however, the downside is that the images often will have some level of noise in them. The noise can be traded off for regularity using interleaved sampling [KH01]. Early methods on analytical visibility [WA77, Cat78, Cat84, KB83], however, are excellent at getting noise-free visibility. There has been a renewed interest in these methods through work [GDAM10, GBAM11, AGJ12, AWJ13] aimed at rasterization-based graphics. Our research in this paper is about developing and adapting analytical motion blur visibility methods for ray tracing. The rest of this previous work section will therefore cover different building blocks that are related to what we use in our work. Sung et al. [SPW02] combine analytical temporal visibility at discrete spatial locations with Monte Carlo-sampled shading. They use a two-pass adaptive scheme where further spatial samples are taken when a certain level of contrast is reached between neighboring pixels. Sung et al.'s work is somewhat similar to our sampling scheme (Section 4), but they did not provide any details on how to compute analytical visibility in time, which is a rather large part of our work.

Motion Blurred Ray Tracing For ray tracing, motion blur can be added by averaging a number of images taken at unique times, however, a better approach is to adapt the spatial data structure to also handle time. Glassner [Gla88] extended the bounding volume hierarchy (BVH) to also

handle time using 4D polyhedrons to bound moving boxes. Olsson extended the kD-tree to handle time as well [Ols07], but the overhead in terms of memory was often significant. Hou et al. [HQL*10] presented an algorithm for micropolygon ray tracing with motion and defocus blur, where they used a hierarchy of oriented hyper-trapezoids and interpolated oriented bounding boxes (OBBs) to the same time of the ray, and then intersected the micropolygon using a simplified 2D test. Grünschloß et al. [GSNK11] introduced a technique so that spatial splits [SFD09] could be used in a motion-blurred BVH with a goal of minimizing overlap. Most approaches today assume that motion is linear [CFLB06, HQL*10, HKL10] and so does Embree [WWB*14], which is a set of highly optimized kernels for traversal and intersection for ray tracing.

Triangle Intersection There are many different methods for determining if and where a ray intersects a triangle [AC97, MT97, KS06]. In this paper, we extend one ray-triangle intersection algorithm [MT97] to also compute the entire intersection analytically for moving triangles. This is somewhat related to how a triangle is rasterized with motion blur, where similar computations happen in world space using edge functions [GDAM10].

Ray vs. Moving Bounding Volume Several researchers have observed that the separating axis theorem (SAT) can be applied for moving polyhedra with linear motion and constant velocity [SE02, Eri04]. By projecting both objects as well as the relative velocity vector onto an arbitrary candidate separating axis at $t = 0$, it can be determined if they initially intersect and if they do not, if or when they will start to do so. If there is no initial intersection, and the objects are moving apart, then there can be no future intersection either. This assumption does not hold, however, when the objects are allowed to scale over time. If, for instance, one object grows in size relative to the other, they may well start to intersect even though they move apart. This is a reality in our context since we use a BVH built for motion blur, where the bounding boxes, which are stored at $t = 0$ and $t = 1$, are not guaranteed to be of exactly the same size. When triangles are undergoing rotational motion, for example, it is unlikely that the bounding boxes at $t = 0$ and $t = 1$ around the triangle are of the same size. Part of our contribution is therefore to generalize previous work so that the intersection between a ray and a moving, scaling AABB (or convex polyhedra) can be found. This is done by considering each object's axis-projection as time-dependent.

For k-DOPs, which includes AABBs, it is well-known that those can be linearly interpolated based on the time parameter, and this has been used in collision detection [LAM03] to quickly update BVHs for deforming models and in ray tracing [CFLB06]. We also want to point out that an AABB vs. moving AABB test can be seen as a shaft culling operation [Hai00]. However, such a test does not produce the interval in time when the two objects overlap.

Miscellaneous Level of detail (LOD) methods can be used for ray tracing to great benefit [CLF*03, YLM06, CFLB06], i.e., one can use coarser geometry representation for most secondary rays, for example. LOD techniques have also been used to speed up motion blur rendering in Pixar's PRMan [Ryu07]. The Razor system introduced a system where continuous LOD could be used for every ray being traced [DHW*11]. We simply note that LOD techniques is something that our analytical intersection test could benefit from as well.

Lately, analytical methods for rasterization have been revived and improved and since rasterization and ray tracing have similiarities, some of that research will be mentioned here. Gribel et al. [GDAM10] present how analytical rasterization can be done in the time dimension over a single spatial pixel location using edge equations and interval lists per pixel. This work was then extended to use line samples in screen space [GBAM11]. Combining sampling along lines over the lens [AMMH07] with analytical motion blur rasterization [GDAM10], you arrive at analytical depth of field sampling [TPD*12], which also presented alternative ways for lossy compression. Auzinger et al. [AGJ12, AWJ13] presented novel ways on how to implement analytical visibility (without motion and defocus blur) entirely on the GPU. Bézier curves with varying thickness was rendered in high-quality using spatial line samples [BGAM12], and a novel interval resolve was presented as well.

Finally, several somewhat similar techniques have been proposed for tile-based stochastic rasterization [MCH*11, LAKL11, MAM12], and parts of the tile overlap testing resemble the work we do in this paper for box vs. ray testing, which is natural since the work starts from the same basic equations. Still, one difference is that rasterization processes one primitive *at a time* in a hierarchical fashion in screen space, while ray tracing usually process a one ray at a time (though there are packet-based versions of ray tracing as well) in world space. Furthermore, we derive a novel time-continuous ray vs. triangle test.

Nowrouzezahrai et al. [NBMJ13] presented a framework for fast high-quality rendering. Contour edges were stored in a dual-space BVH and an integral of visibility-masked spherical visibility was computed over u-isolines over the sphere. Their proof-of-concept implementation showed that their approach was faster than quasi Monte-Carlo ray tracing (QMCRT) on low to moderate complexity scenes, while for more complex scenes, QMCRT was faster.

3. Theory

To be able to obtain t-continuous intersection (visibility) information for each traced ray, two new tests are needed. The first is for ray vs. moving triangle intersection and the other is for ray vs. moving AABB intersection. These tests are presented in this section. The ray-triangle test is a generalization

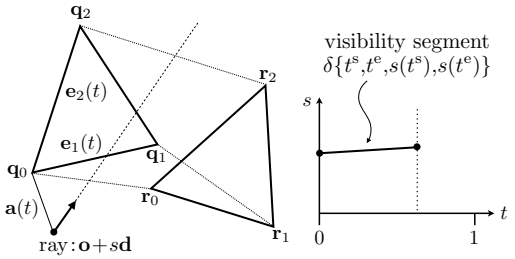


Figure 2: *Time-continuous intersection between a ray and a triangle undergoing per-vertex linear motion. The intersection gives rise to a visibility segment δ, defined from the times and ray distances at the start and end of the intersection.*

of the Möller-Trumbore test [MT97] to work for triangles with moving vertices and the ray-moving AABB test utilizes the separating-axis theorem (SAT) to intersect a ray against a moving AABB (henceforth just called a *box*). These tests are applied at leaf- and node-level, respectively, during BVH traversal.

3.1. Time-Continuous Ray vs. Moving Triangle

A ray, $\mathbf{r}(s) = \mathbf{o} + s\mathbf{d}$, is parameterized by s, which can be interpreted as the signed distance from the ray origin, $\mathbf{o}$ to a certain point on the ray, if the ray direction, $\mathbf{d}$, is normalized. A triangle can be described by three points, $\mathbf{p}_0$, $\mathbf{p}_1$, and $\mathbf{p}_2$. This is visualized in Figure 2. The intersection between a ray and a triangle can be formulated using barycentric coordinates u and v [MT97], as

$$\begin{pmatrix} -\mathbf{d} \\ \mathbf{p}_1 - \mathbf{p}_0 \\ \mathbf{p}_2 - \mathbf{p}_0 \end{pmatrix}^{\mathrm{T}} \begin{pmatrix} s \\ u \\ v \end{pmatrix} = \mathbf{o} - \mathbf{p}_0 \iff \mathbf{A} \begin{pmatrix} s \\ u \\ v \end{pmatrix} = \mathbf{a}, \tag{1}$$

which is solved using Cramer's rule, i.e.,

$$\begin{pmatrix} s \\ u \\ v \end{pmatrix} = \frac{1}{|\mathbf{A}|} \begin{pmatrix} |\mathbf{A}_0| \\ |\mathbf{A}_1| \\ |\mathbf{A}_2| \end{pmatrix} = \frac{1}{(\mathbf{d} \times \mathbf{e}_2) \cdot \mathbf{e}_1} \begin{pmatrix} (\mathbf{a} \times \mathbf{e}_1) \cdot \mathbf{e}_2 \\ (\mathbf{d} \times \mathbf{e}_2) \cdot \mathbf{a} \\ (\mathbf{a} \times \mathbf{e}_1) \cdot \mathbf{d}, \end{pmatrix}, \tag{2}$$

where $\mathbf{e}_1 = \mathbf{p}_1 - \mathbf{p}_0$, $\mathbf{e}_2 = \mathbf{p}_2 - \mathbf{p}_0$, $\mathbf{a} = \mathbf{o} - \mathbf{p}_0$, and $\mathbf{A}_i$ is the matrix formed from $\mathbf{A}$ with the i:th column replaced by $\mathbf{a}$. For an intersection to be valid, it must hold that

$$\left.\begin{matrix} u \\ v \\ 1-u-v \end{matrix}\right\} \geq 0. \tag{3}$$

An important property of this intersection test is that the $1/|\mathbf{A}|$-division in Equation 2 is not required for a binary `HIT/MISS` classification of the intersection. The sign of the determinant, $|\mathbf{A}|$, indicates facing of the triangle, that is, $|\mathbf{A}| >= 0$ for front-facing and $|\mathbf{A}| < 0$ for back-facing. If back-facing triangles are to be detected, which is common in, e.g., ray tracing, u, v, and $|\mathbf{A}|$ need to be negated in Equation 3 whenever $|\mathbf{A}| < 0$. Now assume that the triangle is moving over time in a per-vertex linear fashion, from $\{\mathbf{p}_i\}$ at $t = 0$ to $\{\mathbf{q}_i\}$ at $t = 1$, according to

$$\mathbf{x}_i(t) = \mathbf{p}_i + t(\mathbf{q}_i - \mathbf{p}_i), \quad \text{for } i \in [0,1,2], \tag{4}$$

where t is the time parameter. Motion is defined relative to the ray origin $\mathbf{o}$ and direction $\mathbf{d}$, which therefore remain time-independent. In absolute terms, the ray may undergo motion as well, e.g. primary rays originating from a moving eye position. The terms $\mathbf{e}_1$, $\mathbf{e}_2$, and $\mathbf{a}$ also become linear functions of time, that is,

$$\begin{aligned} \mathbf{e}_1(t) &= \mathbf{x}_1(t) - \mathbf{x}_0(t) = \mathbf{f} + \mathbf{g}t, \\ \mathbf{e}_2(t) &= \mathbf{x}_2(t) - \mathbf{x}_0(t) = \mathbf{h} + \mathbf{i}t, \\ \mathbf{a}(t) &= \mathbf{o} - \mathbf{x}_0(t) = \mathbf{j} + \mathbf{k}t, \end{aligned}$$

where $\mathbf{f} = \mathbf{p}_1 - \mathbf{p}_0$ and $\mathbf{g} = \mathbf{q}_1 - \mathbf{q}_0 - (\mathbf{p}_1 - \mathbf{p}_0)$, and similarly for the other terms above. The intersection in Equations 1 and 2 now becomes

$$\begin{pmatrix} -\mathbf{d} \\ \mathbf{e}_1(t) \\ \mathbf{e}_2(t) \end{pmatrix}^{\mathrm{T}} \begin{pmatrix} s \\ u \\ v \end{pmatrix} = \mathbf{a}(t) \iff \mathbf{A}(t) \begin{pmatrix} s \\ u \\ v \end{pmatrix} = \mathbf{a}(t), \tag{5}$$

with the solution, again using Cramer's rule,

$$\begin{pmatrix} s(t) \\ u(t) \\ v(t) \end{pmatrix} = \frac{1}{|\mathbf{A}(t)|} \begin{pmatrix} (\mathbf{a}(t) \times \mathbf{e}_1(t)) \cdot \mathbf{e}_2(t) \\ (\mathbf{d} \times \mathbf{e}_2(t)) \cdot \mathbf{a}(t) \\ (\mathbf{a}(t) \times \mathbf{e}_1(t)) \cdot \mathbf{d}, \end{pmatrix}, \tag{6}$$

where $|\mathbf{A}(t)| = (\mathbf{d} \times \mathbf{e}_2(t)) \cdot \mathbf{e}_1(t)$. The intersection criteria now becomes

$$\left.\begin{matrix} u(t) \\ v(t) \\ 1-u(t)-v(t) \end{matrix}\right\} \geq 0. \tag{7}$$

As expected, we have arrived at a time-dependent ray-triangle intersection test. This may be carried out as a t-stochastic intersection test by by evaluating Equation 6 using a specific time, t_s. In effect, the triangle is then interpolated to its position at t_s and then intersected as if it were static. To evaluate the t-continuous intersection however, the three polynomials in Equation 7 have to be solved for t. To this end, we rewrite $u(t)$ and $v(t)$ as dot product, i.e.,

$$u(t) = \frac{1}{|\mathbf{A}(t)|} \begin{pmatrix} (\mathbf{h} \times \mathbf{j}) \cdot \mathbf{d} \\ (\mathbf{h} \times \mathbf{k} + \mathbf{i} \times \mathbf{j}) \cdot \mathbf{d} \\ (\mathbf{i} \times \mathbf{k}) \cdot \mathbf{d} \end{pmatrix}^{\mathrm{T}} \begin{pmatrix} 1 \\ t \\ t^2 \end{pmatrix} \geq 0, \tag{8}$$

$$v(t) = \frac{1}{|\mathbf{A}(t)|} \begin{pmatrix} (\mathbf{j} \times \mathbf{f}) \cdot \mathbf{d} \\ (\mathbf{j} \times \mathbf{g} + \mathbf{k} \times \mathbf{f}) \cdot \mathbf{d} \\ (\mathbf{k} \times \mathbf{g}) \cdot \mathbf{d} \end{pmatrix}^{\mathrm{T}} \begin{pmatrix} 1 \\ t \\ t^2 \end{pmatrix} \geq 0. \tag{9}$$

The determinant of $\mathbf{A}(t)$ is given by

$$|\mathbf{A}(t)| = \begin{pmatrix} (\mathbf{h}\times\mathbf{f})\cdot\mathbf{d} \\ (\mathbf{h}\times\mathbf{g}+\mathbf{i}\times\mathbf{f})\cdot\mathbf{d} \\ (\mathbf{i}\times\mathbf{g})\cdot\mathbf{d} \end{pmatrix}^{\mathrm{T}} \begin{pmatrix} 1 \\ t \\ t^2 \end{pmatrix}, \tag{10}$$

which is used to form the third polynomial

$$1 - u(t) - v(t) \geq 0. \tag{11}$$

The t-roots to Equations 8, 9, and 11 are computed analytically, similar to what Gribel et al. did [GDAM10]. These roots represent points in time when the moving edges of the triangle either begins or seizes to intersect the ray. Whenever all polynomials simultaneously satisfy Equation 7, the triangle itself intersects the ray. The $\frac{1}{|\mathbf{A}(t)|}$-division in Equation 8 and 9 normalizes $u(t)$ and $v(t)$ and is not needed for finding the roots.

We denote each interval of intersection a *visibility segment*, $\delta\{t^{\mathrm{s}}, t^{\mathrm{e}}\}$, defined from a start time t^{s} and an end time t^{e}. Vertex motion, per Equation 4, is such that we are interested only in visibility segments where $[t^{\mathrm{s}}, t^{\mathrm{e}}] \cap [0,1] \neq \varnothing$. Due to the non-linearity of Equations 8, 9, and 11, a single ray-triangle test may give rise to multiple visibility segments. In our renderer, all visibility segments generated during ray traversal are stored in a list associated with the ray. Once traversal has finished, the list is resolved for a final visibility integral. At this point it contains the full, t-continuous set of visibility information seen by the ray over the entire valid time interval, $t \in [0,1]$. This will be explained further on Section 5.

Depth function The t-dependent ray distance $s(t)$ is given by Equation 6, i.e.,

$$s(t) = \frac{|\mathbf{A}_0(t)|}{|\mathbf{A}(t)|} = \frac{(\mathbf{a}(t)\times\mathbf{e}_1(t))\cdot\mathbf{e}_2(t)}{(\mathbf{d}\times\mathbf{e}_2(t))\cdot\mathbf{e}_1(t)}. \tag{12}$$

$s(t)$ is undefined for $|\mathbf{A}(t)| = 0$, in which case the ray and triangle lie within the same plane. From inspection of the factors, we can conclude that $s(t)$ is cubic in the nominator and quadratic in the denominator in t. This is consistent with earlier work in the context of analytical rasterization [GDAM10, AMAM13]. Our end results consist of barycentric coordinates, $(u(t), v(t))$ and a cubic-quadratic rational depth function, $s(t)$, which is similar to these earlier works. However, we approach it from world-space coordinates instead of clip-space coordinates. This makes our equations suitable for world-space applications, such as ray tracing and continuous collision detection. In our prototype ray tracer, each visibility segment makes a linear approximation over $[t^{\mathrm{s}}, t^{\mathrm{e}}]$ by storing the ray distance at the start- and end-times. Combined with the triangle index, the full visibility segment definition becomes: $\delta\{t^{\mathrm{s}}, t^{\mathrm{e}}, s(t^{\mathrm{s}}), s(t^{\mathrm{e}}), \mathrm{ID}\}$.

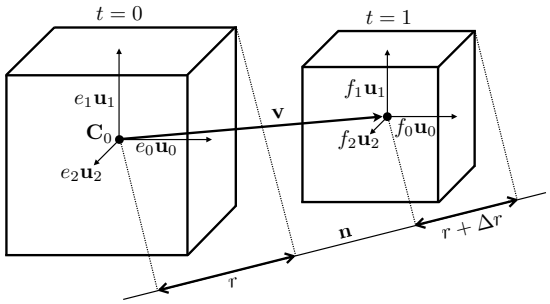


Figure 3: *An AABB undergoing scaling and rectilinear motion along* $\mathbf{v}$ *over* $t \in [0,1]$. *The local basis,* $\mathbf{u}_{\mathrm{i}}$, *corresponds to the unit frame* $(1,0,0)$, $(0,1,0)$ *and* $(0,0,1)$, *with radial extents* e_{i} *at* $t = 0$ *and* f_{i} *at* $t = 1$. *The projection of the AABB onto an arbitrary axis,* $\mathbf{n}$, *amounts to* r *at* $t = 0$ *and* $r + \Delta r$ *at* $t = 1$.

3.2. Time-Continuous Ray vs. Moving AABB with Temporal Bounds

As mentioned in Section 2, the previous polyhedron vs. moving polyhedron tests can handle only the case when the size of the polyhedra are constant. This assumption does not generally hold when the polyhedra are represented by dynamic bounding boxes (AABB's) in a BVH, since the bounds of transforming geometry often will vary over time. We generalize Levine's algorithm, described by Schneider and Eberly [SE02], in the context of AABB's to work with motion with linearly changing size of the box. In our test, the ray, $\mathbf{r}(s) = \mathbf{o} + s\mathbf{d}$, is interpreted as a degenerate, convex polyhedron with a single edge along the ray direction, $\mathbf{d}$, through the ray origin, $\mathbf{o}$. This allows us to construct a new test using the SAT as a basis.

Candidate Separating Axes When applying SAT to two convex polyhedra, the set of candidate separating planes, π_{i}, are comprised of the face normals and the cross products between all unique edge directions. Since an AABB has three unique edge directions, namely the cardinal axes $\mathbf{u}_{\mathrm{i}} = \{(1,0,0),(0,1,1),(0,0,1)\}$, $\mathrm{i} \in \{0,1,2\}$, and a ray has (is) a single edge $\mathbf{d}$, there is a total of three candidate separating axes, which are

$$\mathbf{n}_{\mathrm{i}} = \mathbf{u}_{\mathrm{i}} \times \mathbf{d}. \tag{13}$$

These axes are normals to three corresponding separating planes, π_i, given by

$$\pi_{\mathrm{i}} : (\mathbf{u}_{\mathrm{i}} \times \mathbf{d}) \cdot (\mathbf{x} - \mathbf{o}) = 0, \tag{14}$$

where $\mathbf{d}$ and $\mathbf{o}$ are the ray direction and origin, respectively, as defined in Section 3.1, and $\mathbf{x}$ is a point on the plane. Note that the cardinal axes, $\mathbf{u}_{\mathrm{i}}$, do not need to be tested to see if they are separating axes because they will always overlap with the ray, since it is infinite length. There is a special case, though, when the ray direction is parallel to one of the axes.

Plane vs. Moving AABB Consider an AABB (*box*) centred at $\mathbf{C}_0$ that moves along a trajectory $\mathbf{v}$ according to $\mathbf{C}(t) = \mathbf{C}_0 + t\mathbf{v}$. This is illustrated in Figure 3. Its radial size is (e_0, e_1, e_2) at $t = 0$ and (f_0, f_1, f_2) at $t = 1$. For an arbitrary plane $\mathbf{n} \cdot \mathbf{x} - d = 0$, the box's projection radius (half of its full projection) onto the normal, $\mathbf{n}$, at $t = 0$ is

$$r_0 = e_0\mathbf{n}_x + e_1\mathbf{n}_y + e_2\mathbf{n}_z,$$

and the projection difference from $t = 0$ to $t = 1$ is

$$\Delta r = (f_0 - e_0)\mathbf{n}_x + (f_1 - e_1)\mathbf{n}_y + (f_2 - e_2)\mathbf{n}_z.$$

The projection over time then is $r(t) = r_0 + t\Delta r$, $t \in [0,1]$.

Let t^s and t^e denote the start- and end-times when the moving AABB intersects the plane. Consequently, $t^s = 0$ if it intersects at $t = 0$, and $t^e = 1$ if it intersects at $t = 1$. In general, the box intersects the plane whenever $\mathbf{C}(t)$ is within distance $\leq r(t)$ of the plane, i.e., when

$$\begin{aligned} &|\mathbf{n} \cdot \mathbf{C}(t) - d| \leq r(t) \\ &\Longleftrightarrow \\ &\mathbf{n} \cdot (\mathbf{C}_0 + t\mathbf{v}) - d \leq \pm(r_0 + t\Delta r). \end{aligned} \tag{15}$$

We solve for t and replace the inequality with an equality,

$$t^{\pm} = \frac{\pm r_0 + d - \mathbf{n} \cdot \mathbf{C}_0}{\mathbf{n} \cdot \mathbf{v} - \pm\Delta r}, \tag{16}$$

which gives us the start and end times for when the box intersects the plane. However we do not yet know which one of t^- and t^+ is the start time and which one is the end time. This depends on how the intersection takes place. We make the following classifications of the box's location with respect to the plane:

$$|\mathbf{n} \cdot \mathbf{C}_0 - d| \leq r_0 \quad \text{(intersects at } t=0\text{)} \tag{17}$$
$$\mathbf{n} \cdot \mathbf{C}_0 - d < -r_0 \quad \text{(inside at } t=0\text{)} \tag{18}$$
$$\mathbf{n} \cdot \mathbf{C}_0 - d > r_0 \quad \text{(outside at } t=0\text{)} \tag{19}$$
$$|\mathbf{n} \cdot (\mathbf{C}_0 + \mathbf{v}) - d| \leq (r_0 + \Delta r) \quad \text{(intersects at } t=1\text{)} \tag{20}$$
$$\mathbf{n} \cdot (\mathbf{C}_0 + \mathbf{v}) - d < -(r_0 + \Delta r) \quad \text{(inside at } t=1\text{)} \tag{21}$$
$$\mathbf{n} \cdot (\mathbf{C}_0 + \mathbf{v}) - d > (r_0 + \Delta r) \quad \text{(outside at } t=1\text{)} \tag{22}$$

Now, t^s and t^e can be assigned as follows:

$$[t^s, t^e] = \quad \varnothing, \quad (18,21) \text{ or } (19,22) \tag{23}$$

$$t^s = \begin{cases} 0, & (17) \\ t^-, & (18,20) \text{ or } (18,22) \\ t^+, & (19,20) \text{ or } (19,21) \end{cases}$$
$$t^e = \begin{cases} 1, & (20) \\ t^-, & (17,21) \text{ or } (19,21) \\ t^+, & (17,22) \text{ or } (18,22) \end{cases} \tag{24}$$

If, for example, the box starts out by intersecting the plane at $t = 0$ and then ends up outside at $t = 1$, then $\{t^s, t^e\} = \{0, t^+\}$. Equation 23–24 is straightforward to vectorize using bit-masks, which is useful for SIMD-implemented BVHs, such as Embree's BVH4MB [WWB*14], where four ray vs. AABB tests are processed in parallel during traversal.

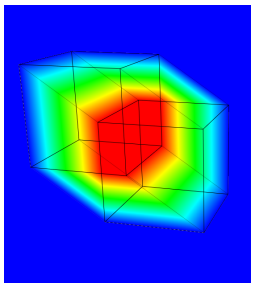


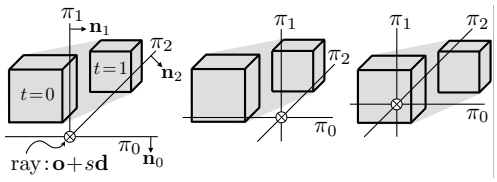
Figure 4: *Ray traced moving AABB where coloring represents duration of intersection using our ray vs. moving box test. Blue color indicates no intersection, while red color indicates intersection throughout $t = [0,1]$. Note the tight bounds of the intervals with respect to the motion of the box.*

Ray vs. Moving AABB The final step of the algorithm is to combine the resulting t-intervals from each plane test through interval intersection. Whenever a separating plane is found or the combined t-interval is empty, the test can be aborted. This part of the algorithm is summarized in Algorithm 1. It assumes the existence of a function IntersectRayPlane, which returns the (possibly empty) temporal bounds of a plane vs. moving AABB intersection given by Equation 23–24. Table 1 presents outcomes for three different rays that are intersected with a moving AABB. Planes are tested in ascending order by index. Note that the rays are pointing into the figure. In the two leftmost examples, the test exists early since separations are found. In the first case, the separation is spatial and in the second case the separation is temporal. In the rightmost example, the ray hits the AABB and the algorithm returns the resulting bounds of intersection, $\bigcap_i t_i^{s,e}$. Figure 4 illustrates a ray traced, moving AABB where the color of each pixel is set according to the bounds of intersection produced by our test. Blue areas represent empty bounds where no intersection takes place and the scale goes up to to red areas, where the box and the ray are intersecting throughout the entire time interval $[0,1]$.

Algorithm 1 Intersect ray $\mathbf{o} + s\mathbf{d}$ with a moving AABB

```
1:  function IntersectRayAABB(o, d, aabb)
2:      [t^s, t^e] ← [0, 1]
3:      for u_i : {(1,0,0), (0,1,0), (0,0,1)} do
4:          n ← d × u_i
5:          [t^s, t^e]* ← IntersectRayPlane(o, n, aabb)
6:          [t^s, t^e] ← [t^s, t^e] ∩ [t^s, t^e]*
7:          if [t^s, t^e] = ∅ then return ∅
8:          end if
9:      end for
10: return [t^s, t^e]
11: end function
```

This intersection test does not produce a depth for the intersection, and hence no intersection point. Instead, our implementation relies on a two step intersection test where rays are first tested against a static, combined box from the $t = 0$



$t_0^{s,e}$	$\varnothing$	$[0.0, 0.1]$	$[0.0, 0.6]$
$t_1^{s,e}$	...	$[0.3, 1.0]$	$[0.0, 0.5]$
$t_2^{s,e}$	...	...	$[0.0, 1.0]$
$\bigcap_{i=0}^{2}$	$\varnothing$	$\varnothing$	$[0.0, 0.5]$

Table 1: *Moving AABB intersected by three different rays with the same direction* $\mathbf{d}$ *(pointing straight into the figure) but with different origins* $\mathbf{o}$*. Three candidate separating axes* π_i *are defined according to Equation 14 and are evaluated by order of index. Corresponding temporal overlaps* $t_i^{s,e}$ *are presented in the table. Left: no intersection since* π_0 *separates the ray spatially from the AABB. Middle: no intersection since the temporal overlaps between* π_0 *and* π_1 *are disjoint. Right: ray intersects the AABB during* $[0.0, 0.5]$.

and $t = 1$ boxes [Woo90]. This test can be highly optimized and provides an over-conservative but fast test, and also a (constant) depth that is used to guide child node traversal. Boxes that are not culled in this initial test are tested again using the above algorithm, before traversal to the intersected nodes continues.

Instead of the solution we presented above, a ray can be naïvely tested t-continuously against a moving AABB by creating a new box that encloses the moving box at its start and end locations, and then intersecting it as if though it was static. This approach may work well for axis-aligned motion, but will produce many false hits whenever this is not the case. Our solution is more efficient in that it produces no false hits at all.

4. Spatio-Temporal Shading Sampling

This section addresses sampling of shading over a pixel in a practical implementation using time-continuous visibility information. In the rest of this paper, *shading* is defined as the net light color seen at a surface point after contributions from light sources, BRDFs, shadow rays, and secondary rays & beyond. We present a novel heuristic where time-continuous visibility information is gathered and then used to sample shading in addition to detecting static geometry. A separate pass is used to sample static geometry using supplemental time-discrete rays, which are spatially distributed over the pixel. Since we use QMC methods both for shading & for static geometry, and time-continuous visibility determination for primary rays, we call our type of sampling for *time-continuous QMC* (TC-QMC). In the following, let us denote a ray with a time-continuous visibility determination for a TC-ray, or $\mathring{\mathbf{r}}$. Recall that the previous section detailed how time-continuous visibility is obtained by intersecting a ray (i.e., a spatial point sample) with scene geometry simultaneously over the full temporal domain (i.e., analytical). This results in a set of visibility segments (or triangles), $\{\delta\}$, with temporal visibility intervals, $[t_i^s, t_i^e]$.

Adaptive shading sampling and reconstruction Since shading sources, such as BRDFs, lights, seconday rays, etc, typically are not available in continuous form, the shading integral is commonly computed by point sampling over time. Gribel et al. [GDAM10] and Tzeng et al. [TPD*12], for instance, sample shading at the midpoint per visibility segment. However, tying shading samples to segments also ties the shading rate to the level of tessellation of the geometry, which may actually lead to under- as well as over-sampling on a mesh level. Coarsely tessellated or slow-moving meshes may receive too few samples and highly tessellated or fast-moving meshes too many. To this end, we use a fixed baseline shading rate per TC-ray, combined with an adaptive heuristic where additional samples are generated based on depth complexity of the segments. The adaptive heuristic identifies *clusters* of C^1-coherent segments, and ensures, in order for each cluster to have a contribution, that each cluster receives at least one shading sample. If no sample from the baseline distribution is contained by the cluster, one is created. The baseline shading samples are jittered over $t = [0, 1]$. This way coherent surfaces, such as walls or floors, will form a single cluster regardless of mesh tessellation, and thus receive a predictable shading rate (the baseline distribution). Complex geometry on the other hand, with incoherency among the segments, will give rise to multiple clusters and thus (potentially) a higher number of samples to account for it.

Cluster formation Clustering takes place after occlusion culling and is the process of identifying continuous meshes in the geometry seen by a TC-ray. Cluster formation is based on C^1-continuity of the triangles, represented by visibility segments. This means that adjacent segments, which are similar within a certain threshold with respect to depth, and the slope of the depth function are considered to belong to the same cluster. The thresholds control the adaptivity of the algorithm. Low thresholds will produce more clusters and hence better image quality, since each cluster is allotted a minimum number of shading samples. High thresholds allow for more discontinuity in the input geometry and may cause shading from fine geometrical features to be lost. In the extreme end, a single cluster is used per TC-ray, in which case adaptivity is in effect disabled, and the algorithm regresses to pure QMC over the temporal domain.

Supplemental rays for static geometry Static geometry is unsuitable for TC-rays, simply because they exhibit limited variation over time. Shading might harbour time-dependent components, but the hit point between a static ray and a

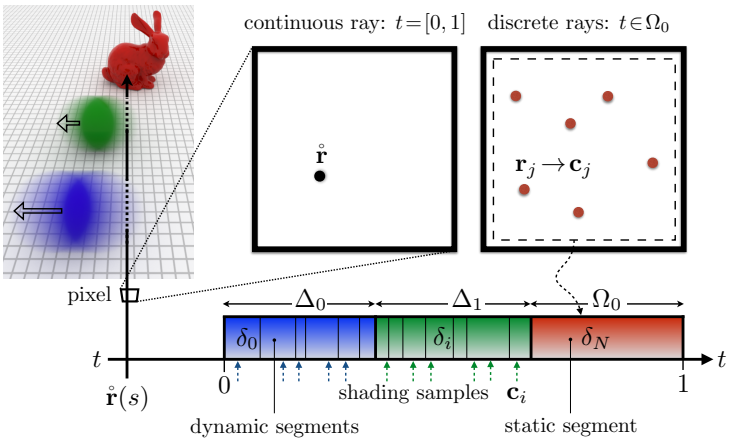


Figure 5: *Ray tracing over the temporal (continuous) and spatial (discrete) domain. A time-continuous ray is intersected with a simple scene comprised of two moving spheres and a static model. The spheres move out of view in sequence and all three objects are visible individually over approximately a third of the time span. Each object forms a distinct cluster of visibility information. Shading of the moving spheres is obtained by sampling the visible triangles with a fixed rate (BRDF-evaluation, secondary rays etc), while new spatially distributed, time-discrete rays ($\mathbf{r}_\mathrm{j}$) are launched to obtain shading ($\mathbf{c}_\mathrm{j}$) for the static bunny. The final pixel color is the weighted sum of contributions from all clusters. Note that over the analytical visibility information for the green and blue objects, we evaluate the BRDF, etc, at five and six discrete shading samples, $\mathbf{c}_\mathrm{i}$, respectively.*

static object is constant over time, which can cause spatial aliasing. To make our algorithm more versatile and allow scene input with static content, we therefore detect static triangles within the visibility set of a TC-ray and treat them differently. To obtain a better spatial distribution, static triangles are sampled using *supplemental rays*, which are regular time-discrete rays, spatially distributed over the current pixel and temporally distributed over the bounds, $[t_\mathrm{i}^\mathrm{s}, t_\mathrm{i}^\mathrm{e}]$, of the triangle's visibility segment. This is unlike the scheme used by Sung et al. [SPW02], where adaptive samples were also time-continuous. The shading contribution of the supplemental rays are then added to the accumulated shading of the TC-ray, weighted by $t_\mathrm{i}^\mathrm{e} - t_\mathrm{i}^\mathrm{s}$. This way, static objects will be sampled much the same way as in QMC.

Figure 5 illustrates how the shading process works. The shading integral over $t = [0,1]$ is the weighted sum of integrals for all segments, as seen by the ray, over their respective visibility intervals, $[t_\mathrm{i}^\mathrm{s}, t_\mathrm{i}^\mathrm{e}]$. In our adaptive approach to generate shading samples, $\mathbf{c}_\mathrm{i}$, over the time-continuous visibility segments, both fast moving, complex geometry generating many small segments, as well as large or slow moving geometry, will be sampled appropriately. If more TC-rays are used per pixel and they are spatially distributed in a uniform or stratified manner, the spatial region for each set of supplemental rays can be made smaller (to the equivalent of the cell size of the uniform grid). This improves the accuracy of our scheme.

Next, our ray tracing traversal kernel, which implements the approach described above, is described.

Hybrid-domain ray kernel

Time-continuous pass (dynamic geometry)

1. Dispatch TC-ray and gather visibility segments, while performing occlusion culling.
2. Identify clusters based on C^1-continuity.
3. For all clusters,
   generate t-samples over cluster
   if dynamic: shade at t-samples
   else: cache samples for the subsequent static pass

Time-discrete pass (static geometry)

4. For cached samples from dynamic pass,
   dispatch time-discrete rays at cached t-value and accumulate shading

Compute final pixel color

5. Combine all shading contributions, weighted by cluster lengths.

Final Pixel Color Computation Denote a cluster of dynamic segments by Δ and a cluster of static segments by Ω (containing a single segment), with temporal lengths $|\Delta|$ and $|\Omega|$, respectively. All clusters of a TC-ray, $\mathring{\mathbf{r}}$, is then the union of the two non-overlapping sets $\{\Delta\}$ and $\{\Omega\}$, with a combined temporal length of 1. Now denote the shading samples taken over a particular dynamic cluster Δ_k as $\{\mathbf{c}\}_{\Delta_k}$, and the shading samples (supplemental rays) taken over a static cluster Ω_k as $\{\mathbf{c}\}_{\Omega_k}$. The total accumulated shading, $\mathbf{C}(x,y)$, over a pixel at (x,y), is then the weighted sum of all shading samples in all clusters, over N TC-rays,

$$\mathbf{C}(x,y) = \frac{1}{N}\sum^{N}\left(\sum_{\Omega_k\in\{\Omega\}}\sum_{\mathbf{c}_j\in\{\mathbf{c}\}_{\Omega_k}}\mathbf{c}_j|\Omega_k| + \sum_{\Delta_k\in\{\Delta\}}\sum_{\mathbf{c}_i\in\{\mathbf{c}\}_{\Delta_k}}\mathbf{c}_i|\Delta_k|\right).$$

	spatial	temporal	visibility set	occluder depth
discrete ray, $\mathbf{r}$	$\mathbf{o}+s\mathbf{d}$	t_i	hit point, $\mathbf{o}+s_{\min}\mathbf{d}$	$s_{\min}$
TC-ray, $\mathring{\mathbf{r}}$	$\mathbf{o}+s\mathbf{d}$	$t=[0,1]$	visibility segments, $\{\delta_\mathrm{i}\}$	$S_{\max}$ over $\{\delta_\mathrm{i}\}$

Figure 6: *Differences between a discrete ray and a time-continuous ray (TC-ray). A ray is* $\mathbf{o}+s\mathbf{d}$*, where* $\mathbf{o}$ *is the ray origin,* $\mathbf{d}$ *is the ray direction, and* s *is the signed distance along the ray, i.e., the "depth.". Note that the TC-ray has a more complex structure, since it can contain an unbounded number of time segments. This can be worked around by, for example, quantizing the times to 8 bits [GDAM10] and using occlusion culling.*

5. Prototype Ray Tracer

We designed a prototype ray tracer around Embree v2.0 [WWB*14]. For temporal integration, Embree utilizes a data structure (BVH4MB) and builder with 4-lane SIMD-vectorized nodes and leafs. Nodes and leafs are stored at start/end locations, i.e., at $t=0$ and $t=1$, respectively. These are interpolated to the time, t, associated with each ray during traversal. BVH4MB was used with only minor modifications within the node class. We also re-used portions of the path tracer-integrator renderer included in Embree 2.0. This renderer is, as stated in Embree's documentation, not fully optimized, but it served our purposes for making comparisons and unoptimized portions do not benefit one algorithm over the other. Our prototype ray tracer primarily features a new inner kernel adapted for mixed discrete and continuous treatment of time, including sampling, traversal, intersection, visibility resolution, and shading. This section will explain how these parts work together.

Time-Continuous Rays The main purpose of a time-continuous ray is to store visibility information continuously over $t=[0,1]$. Each new visibility segment generated by a triangle intersection is stored in the TC-ray. A regular *discrete ray*, in contrast, is associated with a single time, t. The main features of the two types of ray are presented in Table 6.

BVH Traversal Our ray tracer utilizes one BVH traversal kernel for TC-rays and another for discrete rays. The discrete version is a standard, stochastic traversal kernel similar to Embree's original BVH4MB traversal kernel, while the continuous traversal kernal is a modified version which implements the ray vs. box and ray vs. triangle intersection tests presented in Section 3.1 and 3.2.

Since the ray vs. box test from Section 3.2 does not produce a minimum intersection depth, the ray vs. box procedure in the time-continuous traversal kernel is executed in two steps. First, the ray is intersected in a static test against a bounding box, which encloses both the box at the start and the box at the end position. For the root node of the BVH, these positions correspond to the boxes at $t=0$ and $t=1$, but further down the hierarchy, narrower temporal bounds will be available through inheritance, which culls the testing in a more aggressive way. This is done using Woo's fast ray vs. box test [Woo90] and its generated minimum intersection depth is used for occlusion culling. If the static box (enclosing the boxes at the start and end positions of the moving box) is hit by the ray, then we continue and execute our more accurate ray vs. moving box test from Section 3.2.

Occlusion Culling It is essential to use occlusion culling in order for ray tracing performance to scale well with scene complexity. In the time-discrete case, the closest depth, $s_{\min}$, is stored within the ray and used to cull boxes and triangles farther away during traversal. For a TC-ray, this operation is more complicated since depth is a *set* over $t=[0,1]$. We use an approach where an approximative $S_{\max}$ is computed over the set and used analogously to $s_{\min}$ during traversal. $S_{\max}$ is obtained by iterating over the sorted set of segments currently held by the ray. If a continuous sequence of segments, caused by, e.g., a moving mesh, is found, $S_{\max}$ is computed as $S_{\max}=\max_i(\max(s(t_i^\mathrm{s}),s(t_i^\mathrm{e}))$. Put another way, this is the maximum depth over all start and end depths over all segments. Note that this computation relies on the assumption that $s(t)$ is approximately linear per-segment, which is not true in theory [AMAM13], but works well in practice. Initially, $S_{\max}=\infty$ and it is then updated periodically during traversal as new segments are inserted. This algorithm can be generalized and made less conservative by the introduction of a $S_{\max}$-hierarchy, similar to culling in motion blur rasterization [AMMH07, BLF*10]. This is left for future work.

6. Results

We used image quality as a function of rendering time to compare our algorithm (TC-QMC) against multi-jittered quasi-Monte Carlo (QMC) rendering. The metric used for image quality is peak signal-to-noise ratio (PSNR), which measures direct quality of an image compared to a reference image in a logarithmic decibel scale (higher is better). The term *efficiency* will be used here to compare image quality for particular rendering times.

Another unit of measure is primary-level *shading samples per pixel* (sspp). In QMC, visibility and shading are both point sampled and their rates *coupled*, so sspp simply corresponds to the number of primary rays per pixel (spp). In TC-QMC, where visibility and shading are *decoupled*, either 1, 2, or 4 primary TC-rays are used per pixel, and sspp refers to, in this case, the total number of shading samples taken over these rays. Due to adaptiveness, sspp may vary

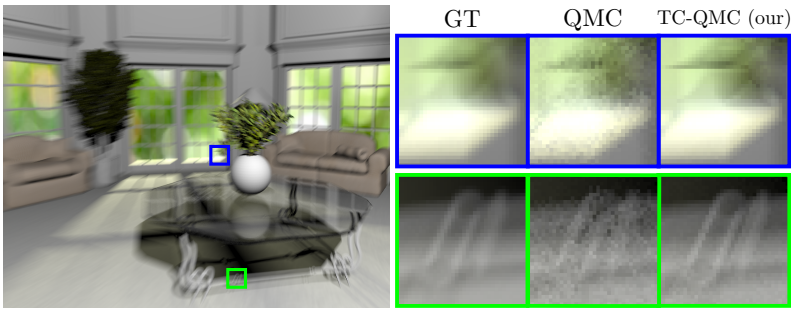


Figure 7: SALA *equal-time comparison. The scene contains one point light on the inside of the room and one on the outside. The tabletop has a dielectric material where all shading originate from reflected or refracted light. QMC obtains a PSNR of 37.8 dB in 5886 ms (32 shading samples per pixel), while TC-QMC (our) obtains 38.3 dB in 5859 ms (33 shading samples on average). The top zoom-in contains parts of a leaf and a window lath, and the bottom zoom-in is an ornamental hole in the table structure which moves diagonally. These areas showcase high image convergence compared to QMC. Other areas, such as the plant which is slower moving than the back- and foreground, exhibits some spatial alias. This, combined with the dielectric tabletop, makes this a quite hard scene which explains why the total PSNR-values are rather close.*

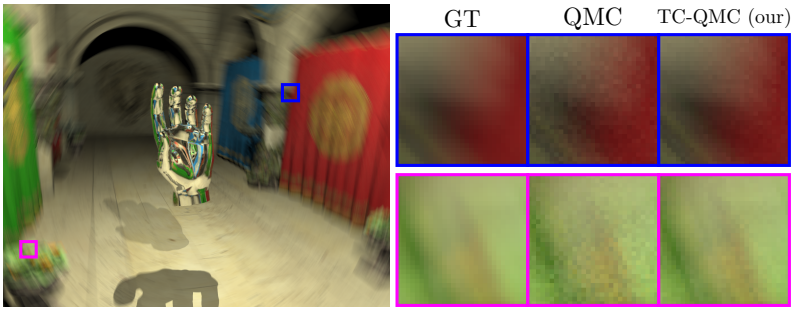


Figure 8: SPONZA+HAND *equal-time comparison. QMC obtains a PSNR of 38.4 dB in 10.0 s (32 shading samples per pixel), while TC-QMC (our) obtains 41.3 dB in 9.9 s (49.3 shading samples on average). The zoom-ins on the pillar/drape and on the plant highlight the low level of noise achieved by time-continuous visibility. The* HAND *model, which is static in camera space, is detected by our algorithm to be static, causing it to be sampled stochastically instead of time-continuously.*

between pixels and any reported sspp-value is therefore the *average* sspp-rate for all pixels in the current image (i.e., not necessarily a power of two).

Measurement data is presented in Figure 9 and 10, while renderings are displayed in Figure 1, 7, and 8.

Test Scenes The prototype ray tracer from Section 5 was used to render three scenes, namely, SPONZA+HAND, SALA, and SAN-MIGUEL, which are comprised of 262k+16k, 400k, and 7.8M triangles, respectively. All scenes contain substantial motion blur induced by linear and angular camera motion. Geometrical edges, in general, and thin geometry, in particular, are challenging from a visibility standpoint and all test scenes have significant amounts of such features, including window cross bars, furniture, pillars, and plants. The hand in SPONZA+HAND rotates in tandem with the camera and thus appear to be static. SAN-MIGUEL stands out as the most geometrically rich and complex model in the set. The tabletop in SALA has a dielectric, reflective-refractive BRDF similar to glass.

Renderings were made in 1024×768-resolution on a MacBook Pro with a 2.6 GHz Intel Core i7 CPU running OS X 10.10. Source code for the ray tracer as well as for Embree was compiled with AVX x86_64 instruction extensions enabled. Reference images were made using 1024–2048 sspp.

Shading Models The scenes were rendered using two separate illumination models, namely, normal shading and Whitted ray tracing [Whi80]. Whitted ray tracing utilizes direct lighting, shadow rays, and recursive specular rays. In normal shading, surface normals were remapped and used as colors. While of limited production interest in and by itself, normal shading was used to highlight visibility determination qualities without interference from other shading components, such as texturing, shadow rays, etc. It should be noted that normal shading may nevertheless exhibit rather sharp shading discontinuities, such as around edges, and thus is not completely devoid of shading variance.

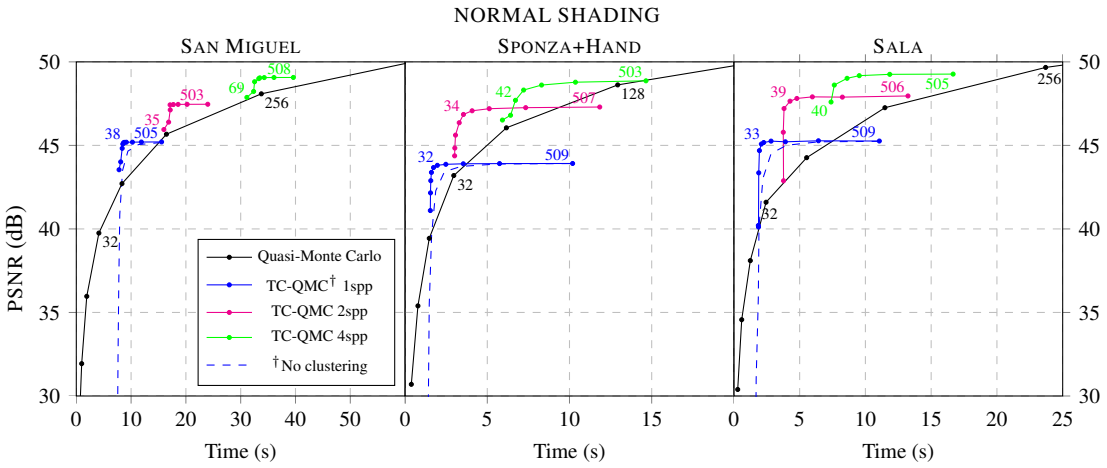


Figure 9: *Image quality as a function of rendering time for our method (TC-QMC) compared to QMC, with surface normal shading. QMC distributes discrete samples over space and time, while our method treats time continuously at 1 (blue), 2 (red), and 4 (green) discrete spatial locations, respectively. Curve annotations represent shading samples at the primary visibility level. TC-QMC exhibits a distinct pattern of faster convergence and better quality for equal rendering times within certain intervals of time. This pattern is particularly distinguished here, since shading is crude and the main source of alias is temporal, which TC-QMC addresses efficiently. Higher number of spatial samples per pixel successively raises the converged quality of our algorithm, keeping it ahead of QMC for extended ranges of rendering time. The blue dotted line represents TC-QMC for 1 spp, with clustering disabled, showing that clustering overall has an improving effect.*

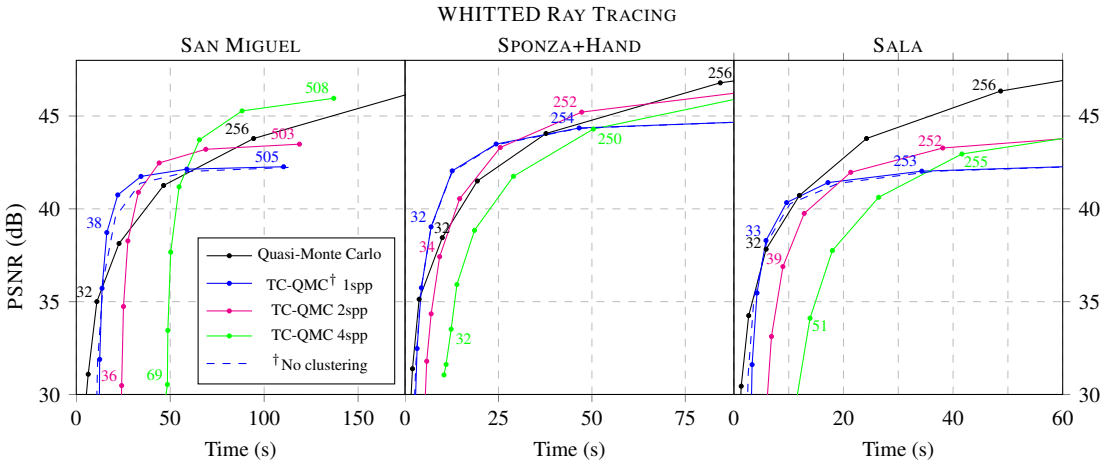


Figure 10: *Image quality as a function of rendering time for our method (TC-QMC) compared to QMC, with Whitted-style ray tracing. TC-QMC performs favourably over QMC in certain ranges of rendering time, for* SAN-MIGUEL *(the most complex scene) in particular, but overall the advantages are fewer here compared to normal shading. Even though feature-aware clustering TC-QMC addresses temporal alias well, but uses point sampling for shading, just like QMC. As variance of the image shifts from the temporal to the shading domain, there is less that sets the algorithms apart. Feature-aware clustering generally provides an improvement, however.*

Characteristics While both algorithms exhibit logarithmic increase of quality over rendering time, their profiles in doing so are quite different (Figure 9 and 10). TC-QMC tends to be less efficient at low rendering times (i.e., for lower number of shading samples per pixel) before rapidly peaking, and then plateauing to a PSNR which depends on the number of used TC-rays per pixel.

The initial, lower efficiency is explained by the higher cost of individual TC-rays compared to their time-discrete counterpart. This is most clearly visible for the curves where clustering is disabled, since clustering introduces samples in a way that offsets the first available data point from near zero in rendering time.

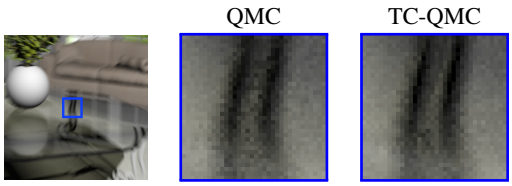


Figure 11: *Accumulated shading, at the glass tabletop in* SALA*, is comparable between our method and QMC given the same number of shading samples (64, in this case). The reasons for this are twofold. First, the tabletop geometry is mostly flat with few discernible geometrical features that can trigger adaptive sampling for our method. Second, since our algorithm considers time-continuous visibility at the primary level and then uses stochastic visibility for secondary rays and beyond, the metal bars, which are only visible after two refractions, will ultimately be sampled in the same way as in QMC (only with a lower spatial distribution).*

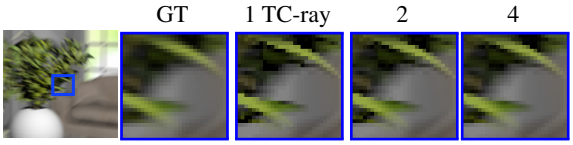


Figure 12: *Spatial alias, such as here on a plant in* SALA*, where some geometrical edges are parallel to the motion direction, is reduced when multiple TC-rays per pixel are used.*

However once computed, the visibility information of TC-rays can then be utilized to sample shading at an arbitrary rate without any further cost for visibility computations. Consequently, the higher, initial cost becomes amortized as more shading samples are used. In QMC, the cost for visibility determination (traversal and triangle intersection) is lower on a per ray basis, but since visibility determination and shading are coupled, this cost will accumulate for each shading sample. This explains why TC-QMC is able to converge significantly faster and stay more effective within certain intervals of rendering time.

The quality level at which TC-QMC then plateaus depends on how many TC-rays are used per pixel (spp). Alias originating from shading will reduce with an increasing number of shading samples, but in order for spatial alias to reduce as well, additional spatially distributed TC-rays need to be used per pixel. As evident in the TC-QMC curves, higher spp-rates lead to curves with similar profiles but with a "shift" towards higher initial cost as well as higher quality. This is depicted visually in Figure 12 as well. As spp goes to infinity, TC-QMC and QMC alike converge to ground truth. For SAN-MIGUEL/Whitted (leftmost diagram in Figure 10), there is a breakpoint at about 42dB where TC-QMC with 2 spp outperform 1 spp, and another one at around 43dB where 4 spp outperform 2 spp. Combined, these three curves outperform QMC between about 20–180 s in absolute rendering time, corresponding to 36–46 dB in image quality, for this particular scene.

These characteristics are most prominent in the normal shading renderings, where shading is simple and temporal visibility is the main source of variance. Here, TC-QMC consistently produces higher quality at equal rendering times.

For Whitted ray tracing, TC-QMC retains an edge within certain rendering times, for SAN-MIGUEL – the most complex scene – in particular, but to a lesser extent. This illumination model adds significant shading complexity in the form of BRDF's, shadow rays, and secondary ray paths. There is also an inherent cost related to shading evaluation which taxes both algorithms equally, thus making them less distinguishable. SPONZA+HAND, for instance, contains a large static object where time-continuous visibility is not utilized at all, and SALA contains a flat dielectric tabletop where all shading originate from secondary, stochastic visibility, again without benefits of time-continuity. The tabletop is presented more closely in Figure 11.

Temporal Coherency An animation was made in SAN-MIGUEL based on the hacienda view in Figure 1. The camera moves into the scene, grazing the plants and up close with the door in the background. The animation exhibits no temporal artifacts, and compares favourably to an identical sequence rendered in equal time using QMC.

Clustering This technique, as explained in Section 4, generates additional shading samples based on geometrical features of the visibility data. Thresholds for cluster formation were set to $\varepsilon_z = 10^{-3}$ and $\varepsilon_k = 10^{-2}$, for C^0- and C^1-continuity, respectively. As a reference, data series with clustering disabled were included in the Figure 9 and 10 diagrams. These cases show that adaptivity increases convergence. In SPONZA+HAND, the increase is marginal, which might be explained by the many large, relatively flat surfaces with high-frequency content textures. One area of improvement here would be be to consider shading variance reduction as a complementary condition for cluster formation.

Ray vs. Box Test To measure the performance of our ray vs. box test (Section 3.2), we compared it to a brute force ray vs. static box implementation, where the static box was combined from the moving box at $t = 0$ and $t = 1$. For rendering times of the latter test normalized to 1, our test rendered in 0.33 for SAN-MIGUEL, 0.55 for SALA and 0.95 for SPONZA+HAND. Figure 4, furthermore, illustrates tightness of the produced temporal bounds.

7. Conclusions and Future Work

A novel take on motion blur for ray tracing has been presented, where we have developed time-continuous intersection tests between rays and moving triangles & moving boxes, and used these tests for improved primary visibility.

Such methods have previously only been used in the context of rasterization. We have also presented a prototype ray tracer, which uses the proposed methods.

We believe that it will be beneficial to continue investigate methods that are continuous in one or more domains, such as our method and the one proposed by Nowrouzezahrai et al. [NBMJ13]. This type of algorithms is in its infancy, and so while the results are not yet fully competitive to point sampling, they could become so with more research in this field. There is certainly a lot of subtopics to do research on. For example, one can continue to look into other ray-triangle intersection tests, such as Plücker-based ones. It would also be interesting to investigate how to relax the definition of "static" to include motion below a certain threshold. As was discussed in Section 4, slowly moving geometry may exhibit low-variance visibility over time and is likely to benefit more from time-discrete sampling compared to time-continuous sampling. It is also important to note that for a packet ray tracer, our ray vs. moving box test can be used to narrow down the interval in time and hence cull rays using that interval. While this should be beneficial, it was deemed out of scope and left for future work.

One natural extension of the time-continuous ray-triangle test (Section 3) is to also consider shadow rays (from a static receiver) to produce noise-free shadows. The shadow ray test is similar to an ordinary intersection test, except that it generates binary occlusion data instead of detailed hit information. The main difference in a time-continuous context lies in how time-continuous occlusion data is represented (a form of binary visibility segments) and resolved. The end result would be a fraction, ranging from 0 (never occluded) to 1 (always occluded). In addition, shadows from moving geometry onto moving receivers is a difficult topic that deserves more research as well. It would also be interesting to exploit shading caches [RKLC*11, GBAM11] and level-of-detail [YLM06] for our approach in order to make rendering faster. Spatial anti-aliasing could potentially be improved by the use of contrast-driven adaptivity [SPW02].

We did some testing using hierarchically inherited t-bounds with the hope to improve performance, where the t-bounds from one ray vs. box test also serve as conservative bounds for any child boxes. This is somewhat similar to how a static ray tracer uses $s_{\min}$ and $s_{\max}$-values along the ray in order to avoid descending the BVH when it is not needed. Unfortunately, we have not been able to get any significant speedup using this approach, but with additional tweaks, this could pay off.

As future work, we want to investigate and develop reconstruction methods, similar to how Lehtinen et al. [LAC*11] reconstruct an image from a point-sampled light field, but in our case, we want to use our spatial point samples with analytical time intervals. This is a type of algorithm that is orthogonal to the work we have presented in this paper, and as such it is a very interesting avenue for future work.

So far, our implementation has been limited to time-continuous *primary* rays, but extending this to secondary or n:are rays is, in theory, straightforward as long as the receiver is static, i.e., when the ray origin and direction is time-independent. To handle time-dependent rays, i.e., moving origin and direction, the intersection test in Section 3 would need to be augmented. This seems to be difficult and our conclusion is therefore that time-continuous visibility for n:ary rays in the general case may not worthwhile pursuing at the moment. Alternatively, a novel way of attacking this problem is needed. However, the combination of using time-continuous visibility for primary rays with stochastic sampling of secondary rays may well be worth doing, especially since most of the motion blurred visibility comes from the primary rays.

Acknowledgements

Tomas is a *Royal Swedish Academy of Sciences Research Fellow*, supported by a grant from the Knut and Alice Wallenberg Foundation. Carl Johan was supported by a grant from Vetenskapsrådet. Thanks to Rasmus Barringer for help with (late-night) SIMD-optimizations. SPONZA is courtesy of Frank Meinl and Marko Dabrovic (original). The HAND mesh is part of the Utah 3D Animation Repository. The SAN MIGUEL hacienda was created by Guillermo M. Leal Llaguno. Finally, we extend our thanks to the *Computer Graphics Forum* reviewers for their valuable feedback.

References

[AC97] AMANATIDES J., CHOI K.: Ray Tracing Triangular Meshes. In *Western Computer Graphics Symposium* (1997), pp. 43–52. 3

[AGJ12] AUZINGER T., GUTHE M., JESCHKE S.: Analytic Anti-Aliasing of Linear Functions on Polytopes. *Computer Graphics Forum, 31*, 2 (2012), 335–344. 2, 3

[AMAM13] ANDERSSON M., MUNKBERG J., AKENINE-MÖLLER T.: Stochastic Depth Buffer Compression using Generalized Plane Encoding. *Computer Graphics Forum, 32*, 2 (2013), 103–112. 5, 9

[AMMH07] AKENINE-MÖLLER T., MUNKBERG J., HASSELGREN J.: Stochastic Rasterization using Time-Continuous Triangles. In *Graphics Hardware* (2007), pp. 7–16. 3, 9

[AWJ13] AUZINGER T., WIMMER M., JESCKE S.: Analytic Visibility on the GPU. *Computer Graphics Forum, 32*, 2 (2013), 409–418. 2, 3

[BGAM12] BARRINGER R., GRIBEL C. J., AKENINE-MÖLLER T.: High-quality Curve Rendering using Line Sampled Visibility. *ACM Transactions on Graphics, 31*, 6 (2012), 162:1–162:10. 3

[BLF*10] BOULOS S., LUONG E., FATAHALIAN K., MORETON H., HANRAHAN P.: Space-Time Hierarchical Occlusion Culling for Micropolygon Rendering with Motion Blur. In *High-Performance Graphics* (2010), pp. 11–18. 9

[Cat78] CATMULL E.: A Hidden-Surface Algorithm with Anti-Aliasing. In *Computer Graphics (Proceedings of SIGGRAPH 78)* (1978), vol. 12, pp. 6–11. 2

[Cat84] CATMULL E.: An Analytic Visible Surface Algorithm for Independent Pixel Processing. In *Computer Graphics (Proceedings of SIGGRAPH 84)* (1984), vol. 18, pp. 109–115. 2

[CCC87] COOK R. L., CARPENTER L., CATMULL E.: The Reyes Image Rendering Architecture. In *Computer Graphics (Proceedings of SIGGRAPH 87)* (1987), vol. 21, pp. 96–102. 2

[CFLB06] CHRISTENSEN P. H., FONG J., LAUR D. M., BATALI D.: Ray Tracing for the Movie Cars. In *IEEE Symposium on Interactive Ray Tracing* (2006), pp. 1–6. 3

[CLF*03] CHRISTENSEN P. H., LAUR D. M., FONG J., WOOTEN W. L., BATALI D.: Ray Differentials and Multiresolution Geometry Caching for Distribution Ray Tracing in Complex Scenes. *Computer Graphics Forum, 22*, 3 (2003), 543–552. 3

[CPC84] COOK R. L., PORTER T., CARPENTER L.: Distributed Ray Tracing. In *Computer Graphics (Proceedings of SIGGRAPH 84)* (1984), vol. 18, pp. 137–145. 2

[DHW*11] DJEU P., HUNT W., WANG R., ELHASSAN I., STOLL G., MARK W. R.: Razor: An Architecture for Dynamic Multiresolution Ray Tracing. *ACM Transactions on Graphics, 30*, 5 (2011), 115:1–115:26. 3

[Eri04] ERICSON C.: *Real-Time Collision Detection*. Morgan Kaufmann Publishers Inc., 2004. 3

[GBAM11] GRIBEL C. J., BARRINGER R., AKENINE-MÖLLER T.: High-Quality Spatio-Temporal Rendering using Semi-Analytical Visibility. *ACM Transactions on Graphics, 30*, 4 (2011), 54:1–54:11. 2, 3, 13

[GDAM10] GRIBEL C. J., DOGGETT M., AKENINE-MÖLLER T.: Analytical Motion Blur Rasterization with Compression. In *High-Performance Graphics* (2010), pp. 163–172. 2, 3, 5, 7, 9

[Gla88] GLASSNER A. S.: Spacetime Ray Tracing for Animation. *IEEE Computer Graphics & Applications, 8*, 2 (1988), 60–70. 2

[GSNK11] GRÜNSCHLOSS L., STICH M., NAWAZ S., KELLER A.: MSBVH: An Efficient Acceleration Data Structure for Ray Traced Motion Blur. In *High Performance Graphics* (2011), pp. 65–70. 3

[HA90] HAEBERLI P., AKELEY K.: The Accumulation Buffer: Hardware Support for High-quality Rendering. In *Computer Graphics (Proceedings of SIGGRAPH 90)* (1990), pp. 309–318. 2

[Hai00] HAINES E.: A Shaft Culling Tool. *Journal of Graphics Tools, 5*, 1 (2000), 23–26. 3

[HKL10] HANIKA J., KELLER A., LENSCH H.: Two-Level Ray Tracing with Reordering for Highly Complex Scenes. In *Graphics Interface* (2010), pp. 145–152. 3

[HQL*10] HOU Q., QIN H., LI W., GUO B., ZHOU K.: Micropolygon Ray Tracing with Defocus and Motion Blur. *ACM Transactions on Graphics, 29*, 4 (2010), 64:1–64:10. 3

[KB83] KOREIN J., BADLER N.: Temporal Anti-Aliasing in Computer Generated Animation. In *Computer Graphics (Proceedings of SIGGRAPH 83)* (1983), pp. 377–388. 2

[KH01] KELLER A., HEIDRICH W.: Interleaved Sampling. In *Eurographics Workshop on Rendering* (2001), pp. 269–276. 2

[KPR12] KELLER A., PREMOZE S., RAAB M.: Advanced (Quasi) Monte Carlo Methods for Image Synthesis. In *ACM SIGGRAPH Courses* (2012), pp. 21:1–21:46. 2

[KS06] KENSLER A., SHIRLEY P.: Optimizing Ray-Triangle Intersection via Automated Search. In *IEEE Symposium on Interactive Ray Tracing* (2006), pp. 33–38. 3

[LAC*11] LEHTINEN J., AILA T., CHEN J., LAINE S., DURAND F.: Temporal Light Field Reconstruction for Rendering Distribution Effects. *ACM Transactions on Graphics, 30*, 4 (2011), 55:1–55:12. 13

[LAKL11] LAINE S., AILA T., KARRAS T., LEHTINEN J.: Clipless Dual-Space Bounds for Faster Stochastic Rasterization. *ACM Transactions on Graphics, 30*, 4 (2011), 106:1–106:6. 3

[LAM03] LARSSON T., AKENINE-MÖLLER T.: Efficient Collision Detection for Models Deformed by Morphing. *The Visual Computer, 19*, 2/3 (2003), 164–174. 3

[MAM12] MUNKBERG J., AKENINE-MÖLLER T.: Hyperplane Culling for Stochastic Rasterization. In *Eurographics – Short Papers* (2012), pp. 105–108. 3

[MCH*11] MUNKBERG J., CLARBERG P., HASSELGREN J., TOTH R., SUGIHARA M., AKENINE-MÖLLER T.: Hierarchical Stochastic Motion Blur Rasterization. In *High-Performance Graphics* (2011), pp. 107–118. 3

[MT97] MÖLLER T., TRUMBORE B.: Fast, Minimum Storage Ray-triangle Intersection. *Journal of Graphics Tools, 2*, 1 (1997), 21–28. 3, 4

[NBMJ13] NOWROUZEZAHRAI D., BARAN I., MITCHELL K., JAROSZ W.: Visibility Silhouettes for Semi-Analytic Spherical Integration. *Computer Graphics Forum, 33*, 1 (2013), 105–117. 3, 13

[NSG11] NAVARRO F., SERÓN F. J., GUTIERREZ D.: Motion Blur Rendering: State of the Art. *Computer Graphics Forum, 30*, 1 (2011), 3–26. 2

[Ols07] OLSSON J.: *Ray-Tracing Time-Continuous Animations using 4D KD-Trees*. Master's thesis, Lund University, 2007. 3

[RKLC*11] RAGAN-KELLEY J., LEHTINEN J., CHEN J., DOGGETT M., DURAND F.: Decoupled Sampling for Graphics Pipelines. *ACM Transactions on Graphics, 3*, 30 (2011), 17:1–17:17. 13

[Ryu07] RYU D.: 500 Million and Counting: Hair Rendering on Ratatouille. In *ACM SIGGRAPH 2007 Sketches* (2007). 3

[SE02] SCHNEIDER P. J., EBERLY D.: *Geometric Tools for Computer Graphics*. Morgan Kaufmann Publichers, 2002. 3, 5

[SFD09] STICH M., FRIEDRICH H., DIETRICH A.: Spatial Splits in Bounding Volume Hierarchies. In *High-Performance Graphics* (2009), pp. 7–13. 3

[SPW02] SUNG K., PEARCE A., WANG C.: Spatial-Temporal Antialiasing. *IEEE Transactions on Visualization and Computer Graphics, 8*, 2 (2002), 144–153. 2, 8, 13

[TPD*12] TZENG S., PATNEY A., DAVIDSON A., EBEIDA M. S., MITCHELL S. A., OWENS J. D.: High-Quality Parallel Depth-of-Field Using Line Samples. In *High Performance Graphics* (2012), pp. 23–31. 2, 3, 7

[WA77] WEILER K., ATHERTON P.: Hidden Surface Removal using Polygon Area Sorting. In *Computer Graphics (Proceedings of SIGGRAPH 77)* (1977), pp. 214–222. 2

[Whi80] WHITTED T.: An Improved Illumination Model for Shaded Display. *Communications of the ACM, 23*, 6 (1980), 343–349. 10

[Woo90] WOO A.: Graphics Gems. 1990, ch. Fast Ray-box Intersection, pp. 395–396. 7, 9

[WWB*14] WALD I., WOOP S., BENTHIN C., JOHNSON G. S., ERNST M.: Embree: A Kernel Framework for Efficient CPU Ray Tracing. *ACM Transactions on Graphics, 33*, 4 (2014), 143:1–143:8. 3, 6, 9

[YLM06] YOON S.-E., LAUTERBACH C., MANOCHA D.: R-LODs: Fast LOD-based Ray Tracing of Massive Models. *Visual Computer, 22*, 9 (2006), 772–784. 3, 13