Volume 35 (2016), Number 7

E. Grinspun, B. Bickel, and Y. Dobashi
(Guest Editors)

Decoupled Space and Time Sampling of Motion and Defocus Blur for Unified Rendering of Transparent and Opaque Objects

S. Widmer[1], D. Wodniok[1], D. Thul[2], S. Guthe[2] and M. Goesele[1,2]

[1] Graduate School of Computational Engineering, TU Darmstadt, Germany [2] TU Darmstadt, Germany

Abstract

We propose a unified rendering approach that jointly handles motion and defocus blur for transparent and opaque objects at interactive frame rates. Our key idea is to create a sampled representation of all parts of the scene geometry that are potentially visible at any point in time for the duration of a frame in an initial rasterization step. We store the resulting temporally-varying fragments (t-fragments) in a bounding volume hierarchy which is rebuild every frame using a fast spatial median construction algorithm. This makes our approach suitable for interactive applications with dynamic scenes and animations. Next, we perform spatial sampling to determine all t-fragments that intersect with a specific viewing ray at any point in time. Viewing rays are sampled according to the lens uv-sampling for depth-of-field effects. In a final temporal sampling step, we evaluate the pre-determined viewing ray/t-fragment intersections for one or multiple points in time. This allows us to incorporate all standard shading effects including transparency. We describe the overall framework, present our GPU implementation, and evaluate our rendering approach with respect to scalability, quality, and performance.

Categories and Subject Descriptors (according to ACM CCS): I.3.6 [Computer Graphics]: Methodology and Techniques—Graphics data structures and data types I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Raytracing

1. Introduction

Defocus and motion blur are integral components to render photo-realistic images. They are often used in movies or games to highlight an important situation or object for storytelling purposes. Distribution ray tracing [CPC84] is a widely employed unified technique to render these phenomena in production quality applications. In contrast, real-time systems cannot until now use a unified algorithm to solve both effects and instead provide separate solutions for defocus and motion blur: Depth-of-field can be solved via approximate ray tracing [LES09, WPS*15], warping light fields [YWY10], or applying a multi-layer filter [SRP*15]. Most modern algorithms use multiple layers to handle disocclusions. Motion blur is mostly implemented using filter kernels as post-processing effects [GMN14]. This can reach a visual quality similar to distribution ray tracing without sacrificing performance. However, these approaches cannot handle defocus and motion blur at the same time, especially if a pixel consists of fragments with different depth, i.e. if it is a combination of opaque and transparent fragments.

Stochastic rasterization [AMMH07, FLB*09] extends the rasterization algorithm and hardware with stochastic sampling to enable defocus and motion blur from the camera and object perspective as well as motion blurred shadow maps [NCJ*12]. McGuire et al. [MESL10] presented a hybrid algorithm for rendering approximate defocus and motion blur with stochastic visibility evaluation within a modern GPU architecture.

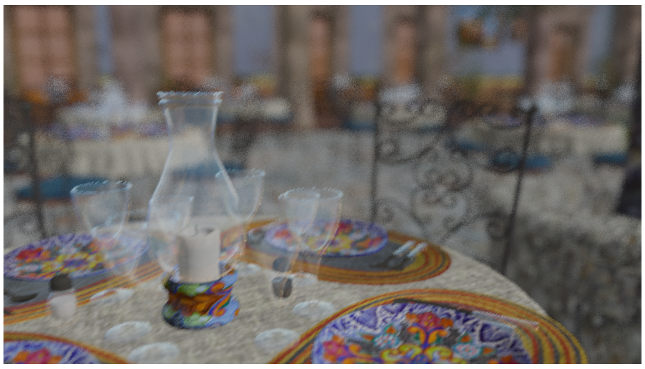

Figure 1: *Decoupling spatial and temporal sampling allows us to produce physically-based, plausible defocus and motion blur at interactive frame rates (rendered at 1024×576 pixel resolution, 4 dof-samples, 8 motion blur sample per dof-sample in 222ms on a GTX TITAN X).*

We present a unified rendering pipeline for interactive defocus and motion blur rendering for transparent and opaque fragments (see Fig. 1). We decouple the visibility test, used to generate depth-of-field in ray tracing, from motion sampling and employ late shading. This reduces ray traversal and shading cost without quality loss compared to other real-time or interactive algorithms. In addition, late shading allows to use a deferred shading approach in combination with correct alpha blending of blurred and transparent fragments to produce physically plausible results. Our contributions are:

- Decoupling time sampling and visibility test, decomposing the traditional 5D sampling into a 4D spatial (xy, uv) and 1D temporal (t) sampling step.
- An intermediate scene representation using temporally-varying fragments (t-fragments) that represent the spatially sampled scene for temporal sampling.
- A disocclusion map that approximates the motion differences between two depth layers in order to identify potentially visible t-fragments for a correct trace result, including a simple edge filter to reduce depth-of-field artifacts at disocclusions.
- A specialized intersection test for temporally-varying axis-aligned bounding boxes (t-AABBs).
- A unified sorting and late shading pass using t-fragments as shading primitives that enables physically plausible transparency with defocus and motion blur at interactive frame rates.

2. Related work

Fundamentally, our proposed algorithm shares similarities with defocus and motion blur rendering for micropolygons by Hou et al. [HQL*10]. They construct an acceleration structure using object aligned bounding boxes in 3D space for both the start time t_0 and end time t_1 of a frame. Assuming linear motion, each pair of bounding boxes forms a 4D hyper-trapezoid in space-time that tightly bounds the object for the entire time interval. A bounding volume hierarchy (BVH) is constructed with the SAH-based BVH construction algorithm of Wald [Wal07] for bounds at $t = 0.5$. During traversal, rays are associated with a time stamp and intersected with the corresponding interpolated bounds. In contrast to Hou et al. [HQL*10] we operate on fragments rather than micropolygons. As rasterization can generate several million fragments, a key to better ray traversal performance is fast and aggressive culling without introducing artifacts. At the same time, we aim at interactive rebuilds of the acceleration structure. We also directly intersect 4D hyper-trapezoids to collect fragments for the whole frame.

2.1. Defocus and motion blur

Algorithms computing defocus and motion blur fall roughly into two categories: Ray tracing based approaches approximate the effect in a physically based manner while real-time approaches try to create a perceptually plausible effect.

Ray Tracing Cook et al. [CPC84] present a unified framework for simulating defocus and motion blur with distribution ray tracing (DRT). They use stochastic sampling to create phenomena such as depth-of-field, motion blur, and shadow penumbras. All effects are simulated by simultaneously sampling in space and time. Images created by distribution ray tracing exhibit a certain level of noise due to the stochastic sampling. Thus, they are usually post-processed using filtering or reconstruction techniques. While early approaches filtered the final rendered images, the state-of-the-art is reconstructing surface lightfields for sample points along each ray [LAC*11, MVH*14, HMV15]. While initially designed for distribution ray tracing , these reconstruction filtering approaches can be used for all rendering algorithms that provide samples using a stochastic process.

In order to reduce the noise prior to filtering without increasing the computational costs, samples should be generated where they improve the rendered image the most. Vaidyanathan et al. [VTS*12] propose an adaptive sampling approach that is based on frequency information obtained through shaders together with the amount of defocus and motion blur in a certain area of the rendered output. They spend more samples in areas of high frequency while smooth areas are filtered more aggressively. Belcour et al. [BSS*13] improve on this by estimating the covariance along rays instead of combining frequency information from different sources in screen space. Tracing covariance has, however, problems with partial occlusion and transparency. Also, this method becomes less efficient in regions that contain both motion blur and defocus.

A slightly different approach was presented by Gribel et al. [GBAM11]. In order to calculate motion blur, they sample line segments in 4D space-time rather than points in 3D space. The visibility along each line segment is solved analytically. However, adding defocus blur would require multiple line segments or tracing finite patches.

Real-Time Rendering A broad overview of existing techniques to solve motion blur is given by Navarro et al. [NSG11]. In general, real-time approaches usually fall into two categories. Guertin et al. [GMN14], for example, perform motion blur as a post-processing filter. There are, however, issues with combining this approach with real-time defocus as both filters require per fragment information and therefore assume that each pixel corresponds to a single fragment. This assumption is no longer true once the first filter has been applied. Selgrad et al. [SRP*15] on the other hand use multi-layer filtering which is unable to handle motion blur at the same time. In contrast, we only sample in 4D (xy, uv) space to solve visibility for depth-of-field. The time domain is later sampled on the resulting t-fragments, reducing the ray tracing overhead.

2.2. Stochastic rasterization

As we rely on rasterization for line-segment generation we share some similarities with stochastic rasterization from Akenine-Möller et al. [AMMH07]. Since the edges of time-continuous triangles (TCT) are bi-linear patches rather than planes, their final rendering calculation is very expensive. Therefore, Akenine-Möller et al. [AMMH07] use a screen space acceleration structure that is created on-the-fly by rendering an OBB around each TCT . In addition, they use Zmax-culling to conservatively reject triangles and fragments before performing actual sample evaluation. Based on this, Fatahalian et al. [FLB*09] partition time into intervals to keep track of moving micropolygons using different approaches for cases with no motion, slow motion and fast motion. Hou et al. [HQL*10] further improve on this by presenting a unified approach for all cases that usually produces higher quality images. Finally, Laine et al. [LAKL11] improve over Fatahalian et al. [FLB*09] with a better sampling pattern defined in dual space. For better depth test performance, Boulos et al. [BLF*10] define the tz-pyramid. It stores not only Zmax for a certain instance in time but also builds a hierarchy over time, based on the maximum z-values in each node of the z-pyramid.

McGuire et al. [MESL10] propose a hybrid algorithm based on stochastic rasterization that can do motion blur and defocus at the same time, running on conventional GPUs. While the authors use motion blur and defocus as examples, their focus is on implementing stochastic rasterization on GPUs. To improve performance, Clarberg and Munkberg [CM14] propose a deferred shading approach. They first create a per-pixel list of primitives that contribute to each pixel. In a separate shading pass, the primitives are shaded and the colors are averaged. In this setting, the per-pixel list already accounts for motion blur and defocus. In order to increase the stochastic rendering performance in general, Wu et al. [WWTS15] implement efficient sample culling for motion blur and defocus.

2.3. Acceleration structure

Acceleration structures are a key component for fast and efficient ray tracing. Most widely used structures are bounding volume hierarchies (*BVHs*) and *k*d-trees, which can be extended to support motion blur. Cook et al. [CPC84] introduce stochastic sampling to solve distribution effects but leave the question of an efficient acceleration structure open. Glassner [Gla88] proposes an acceleration structure that is based on an octree over objects. Each node is split until it contains only a single object or a maximum split level is reached. Each partial object contained entirely in a single octree node is then bound by 4D k-DOPs.

***k*d-Tree** Olsson [Ols07] extended *k*d-trees by adding a temporal split. The increasing number of object references introduce, however, a significant memory overhead, which limits its practical applicability?

BVH The fastest BVH construction algorithm on the GPU is currently the LBVH [LGS*09] with optimizations from Karras [Kar12]. It uses Morton codes for the center of each primitive and applies Radix sort to create a linear list of primitives. This linear list is then used to construct the final bounding volume hierarchy. While being extremely fast to create, its efficiency during rendering is up to 85% lower when compared to SAH based construction. Grünschloß et al. [GSNK11] propose a 4D space-time extension to the spatial split BVH algorithm [SFD09] called MSBVH. It is mainly suited for irregularly tesselated polygonal scenes and the high construction time renders it mainly relevant for production rendering settings.

3. Architecture

Our unified rendering pipeline is focused on combining transparent and opaque fragments with defocus and motion blur. We thus decouple spatial and temporal sampling to reduce the rasterization effort. We also separate visibility and shading by employing late shading after the motion sampling. This allows us to perform correct alpha blending of blurred and transparent fragments. Accurate reflection of the environment for all fragments is accomplished as the reflection vector is computed for the position of the fragment in time. Also, our unified rendering pipeline can easily be combined with tile-based deferred rendering to reduce shading overhead.

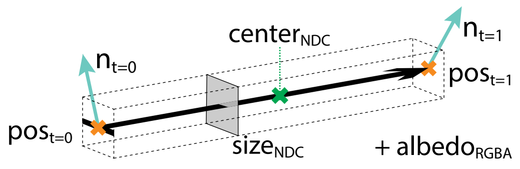

Figure 2: *All components of a t-fragment in NDC space.*

A central concept in this paper is the temporally-varying fragment or t-fragment (see Fig. 2). Without loss of generality, we define the duration of a frame as $[0,1]$ and we assume that the motion of fragments is linear within frames [MESL10]. Given a regular fragment created at $t \in [0,1]$, a t-fragment represents the oriented line connecting the fragment's positions at $t = 0$ and $t = 1$ in normalized device coordinate (NDC) space. The t-fragment contains all shading information (time dependent normals, albedo and transparency) and implicitly the size of its footprint. Using the linear motion assumption, we can interpolate these attributes linearly and we derive a capsule with an ε radius as ideal object oriented bounding volume for each t-fragment.

3.1. Overview

Our algorithm consists of three main steps (see Fig. 3): The first step is the rasterization that creates the disocclusion map and the t-fragments. The second step includes the depth-of-field raytracing to resolve general visibility and the motion blur time sampling to determine time-dependent visibility. The last step is the final shading and tone-mapping.

The initial pass in the rasterization renders all opaque objects to create the disocclusion map, as well as the first and second layer depth buffer. The disocclusion map (see Fig. 4 (c)) marks possible disocclusion resulting from camera and object motion or depth-of-field. We create a new depth buffer by combining the first and second layer using the disocclusion map. This buffer is used for early fragment culling during the rasterization of the t-fragments in the next pass. Since a t-fragment expands over several pixels with high overlap depending on the amount of motion, a fragment linked-list is not a suitable data structure. Thus, we separate opaque and transparent fragments into two different unordered arrays.

In the ray tracing step we construct two bounding volume hierarchies using the unordered arrays. The hierarchies have two different bounding volumes, namely capsules at leaf nodes and temporally-varying axis-aligned bounding boxes (t-AABBs) for the inner nodes to resolve visibility by ray tracing. We implemented a naïve depth-of-field algorithm that samples the lens aperture [SC97] and trace a fixed number of rays per pixel according to the thin-lens model. This yields a set of t-fragments per viewing ray containing all t-fragments intersected at some point in time $t \in [0,1]$. In the motion sampling pass, we instantiate each capsule for one or multiple $t \in [0,1]$ as a small sphere with radius ε at the interpolated location. Fragments are created for all instances that intersect with the viewing ray, sorted according to their depth, and shaded with correct alpha blending for transparency.

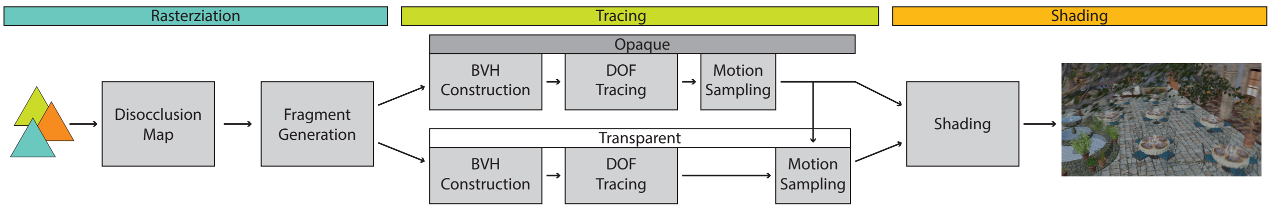

Figure 3: *Overview of our rendering pipeline consisting of rasterization (generation of t-fragments), tracing (spatial and temporal sampling) and shading (lighting and transparency handling). Note that all of these steps are running on the GPU.*

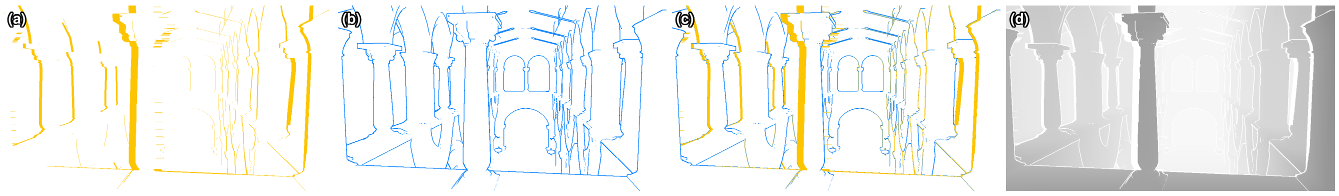

Figure 4: *Disocclusion map construction: Given the velocity of fragments, we generate an* ***(a)*** *occlusion buffer by propagating the velocity differences to neighboring pixels. Next, depth edges are detected and stored in the* ***(b)*** *Laplacian buffer. We combine both buffers to the* ***(c)*** *disocclusion map, which is used to generate the* ***(d)*** *modified depth buffer. This depth buffer is used for early fragment culling. Initially the depth of the first depth layer is used. At pixel positions marked by the disocclusion map the depth value of the second depth layer is used.*

3.2. Generation of *t*-Fragments

To generate t-fragments, we rasterize the scene at $t = 1$. This will remove artifacts caused by objects appearing during the rendered frame while potentially loosing disappearing objects. Applying a guard band can reduce the artifacts further. In general, any time between 0 and 1 can be chosen in this step. In order to save resources in later stages of the pipeline, we propose a number of additional steps that allow us to reduce the number of t-fragments while at the same time ensuring approximate correctness of the output images.

Disocclusion Map The purpose of the disocclusion map is to efficiently cull t-fragments which will not contribute to the final image (see Fig. 4). We first perform depth only rendering of all front-facing opaque primitives at $t = 1$ to extract the first and second depth layers. Without disocclusion we would only store opaque t-fragments which correspond to the first depth layer and transparent t-fragments in front of the first layer. However, we have to store all t-fragments which are not visible at $t = 1$ but might become visible due to motion in the frame or depth-of-field rays that "look behind" edges. We detect these disocclusions on a per pixel basis and set the values of the first layer depth map to the respective values of the second layer depth map at these locations.

To detect disocclusions caused by motion we compute a velocity map (motion field) for the first opaque layer. Next we compute forward differences of the velocity map in x and y direction. Motion disocclusions can only occur at pixels that have a positive velocity difference. By looking at the depth of the neighboring (right and top) pixels in relation to the current pixel's depth we can decide in which direction the disocclusion needs to be resolved. The velocity difference value corresponds to the severity of the disocclusion and is converted to pixel units. Each pixel now carries information on whether and how big of a disocclusion occurs, thus specifying a rectangular area of disocclusion around the pixel. Such a region is specified with parameters l, r, t, and b for the extent of the disocclusion in left, right, top, and bottom direction respectively. Pseudocode for motion disocclusion map initialization is shown in Algorithm 1 in the supplemental material.. To obtain the binary occlusion map the disocclusion area information from each pixel needs to be collected. This is done by pixels iteratively spreading their disocclusion information over the entire buffer. In the first iteration each pixel will spread its disocclusion information to its direct neighbors and with each iteration the distance that the information is spread will be doubled until the disocclusion information has spread over the entire occlusion map. Pseudocode for disocclusion spreading is shown in Algorithm 2 in the supplemental material..

Correct depth-of-field rendering requires access to geometry that is occluded in the traditional pinhole camera setting. In an effort to balance efficiency and accuracy, we allow for a limited amount of disocclusion in the depth-of-field case. We detect depth discontinuities by thresholding the response of a Laplacian filter on the depth map and store the resulting value in a binary Laplacian map.

In the final step we simply combine the binary occlusion map and the Laplacian map with a simple **OR** operation to yield our novel disocclusion map. Each pixel in the disocclusion map marks whether it is a source of disocclusion or not. Using this information we modify the first layer depth map. For every pixel marked in the disocclusion map we set the depth value back to the second layer. Otherwise we use the first depth layer's value. Figure 15 shows the difference between using one and two layers.

Generation In a final rasterization pass we create the actual t-fragments. We again start by rendering the scene at $t = 1$ but use the now read-only *modified* depth buffer to enable early fragment culling (at $t = 1$) in hardware. For all fragments that pass the depth test we create a corresponding t-fragment and store it in an unordered array.

3.3. Ray tracing acceleration structure

To accelerate collecting t-fragments for depth-of-field samples we construct a temporally-varying bounding volume hierarchy (BVH). Since t-fragments are stored in NDC space, the BVH is built in NDC space as well. Viewports with an aspect ratio $\neq 1$ cause an anisotropic scaling of t-fragment coordinates in NDC space. To avoid this distortion we store positions in an *anisotropic* NDC space, preserving the viewport's aspect ratio in the x- and y-dimension. In this undistorted space, we can bound a t-fragment with a moving sphere for cheap intersection testing. We define the radius ε of the sphere as the radius of the circumsphere of a cube with the side length of a pixel in anisotropic NDC space. Projecting t-fragment motion into 3D-space results in a capsule as the bounding volume, which is defined by the two t-fragment positions points and a (constant) radius. Thus, no additional memory is needed for t-fragment bounds.

We use a fast bottom-up approach for hierarchy construction inspired by LBVH [LGS*09]. First, all capsules are sorted according to the Morton code of their mid-point in NDC space. We then generate the topology of the hierarchy from this sorted list with the fast construction algorithm of Karras [Kar12]. t-Fragments with identical Morton code belong to the same leaf node. The resulting topology corresponds to a BVH constructed with a spatial-median split strategy. As t-fragments are roughly uniformly distributed in x and y direction, this yields a good quality BVH.

Next, we compute bounds for the leaves and inner nodes. We first tried to use tight irregular capsules with different radii at the end points as node bounds. This requires computing tight bounding spheres for a leaf's t-fragment bounding spheres at $t \in \{0,1\}$, which is non-trivial for more than two spheres. For inner nodes computation of irregular capsules from children bounds is simple but bounding efficiency proved to be suboptimal and decreased with each level up the hierarchy. Furthermore, intersecting an irregular capsule is non-trivial and expensive. Thus we decided to use temporally-varying axis aligned bounding boxes (t-AABBs) for node bounds. t-AABBs are defined by a pair $aabb_{t=0}$ and $aabb_{t=1}$ of AABBs for $t \in \{0,1\}$. For leaf nodes computing tight AABBs for t-fragment bounding spheres at $t \in \{0,1\}$ is fast and simple, as is propagating t-AABBs up the hierarchy. At the same time bounding efficiency is higher than for irregular capsules.

Intersection Intersecting a t-AABB with a ray at $t \in [0,1]$ only requires to linearly interpolate $aabb_{t=0}$ and $aabb_{t=1}$ and intersect the interpolated AABB (see Fig. 5, left). We are instead interested in intersecting a ray against the temporal projection of a t-AABB into 3D-space. One possible approach for this is to construct a shaft for $aabb_{t=0}$ and $aabb_{t=1}$ [HW94], and intersect the ray with the resulting polyhedron. Since an efficient implementation requires to store a significant amount of precomputed data per node, we propose a slightly conservative but simpler t-AABB intersection test, which needs no additional memory.

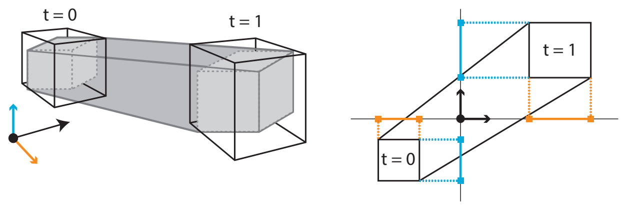

Figure 5: *Intersection of a ray with a t-AABB. The t-AABB is transformed into a local ray coordinate system, which reduces the intersection test to a 2D problem. The AABBs at $t \in \{0,1\}$ are conservatively replaced with axis aligned rectangles. Now intersection time intervals are computed separately for projections on both local coordinate axes. A non-empty intersection of the time intervals indicates an intersection of the t-AABB.*

The key idea is to transform the t-AABB into a local ray space where the intersection test can be reduced to a two dimensional problem. For this, we first construct an arbitrary 3D orthonormal basis B from the ray direction (e.g. [HM99, Fri12]) once before traversal of the BVH. Using B and the ray origin, we perform an affine transformation of the t-AABB into local ray space, where the ray origin is at $(0,0,0)$ and the ray direction corresponds to the z-axis. At this point we can ignore the local z-dimension and reduce the intersection test to a 2-dimensional problem in the local x-y-plane. We could perform a point-in-convex-hull test but extraction of the convex hull is too costly to be performed frequently during traversal. Instead, we test for inclusion of the local origin in the temporally varying projection of the t-AABB on the local x-y-plane. To simplify this test we conservatively replace the projections of $aabb_{t=0}$ and $aabb_{t=1}$ with tight temporally varying axis aligned bounding rectangles. This allows us to further reduce the problem to two one dimensional intersection problems, where we simply have to compute the time intervals for which the local origin is separately contained in the projection of the bounding rectangle onto the local x- and y-axis. If the intersection of both time intervals is non-empty, we intersect the temporally varying axis aligned bounding rectangle and conservatively assume that the t-AABB has been intersected (see Fig. 5, right). The intersection test is exact if the basis vectors of B are parallel to the NDC coordinate axes. Important for the efficiency of our approach is the fact, that we do not have to actually transform all eight corners of $aabb_{t=0}$ and $aabb_{t=1}$ to compute the projections on the local x- and y-axis. Analyzing the signs of the components of the basis vectors of B we can derive two extreme points for each projection axis, and $aabb_{t=0}$ and $aabb_{t=1}$ separately which suffice to compute the projection bounds, and thus greatly reduce computational cost.

The ray tracer uses a simple stack based ray traversal algorithm. Rays are less coherent for depth-of-field rays. This can cause different rays to find leafs at different points in time. To improve SIMD efficiency in such situations we employ the two-phase while-while traversal from Aila et al. [AL09].

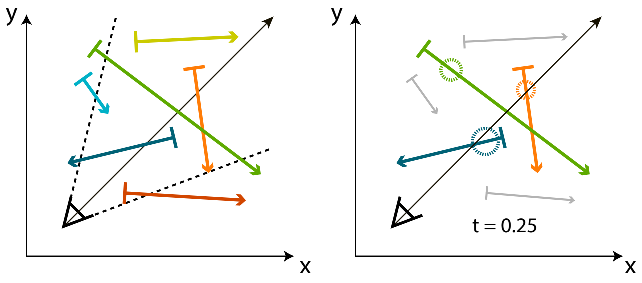

Figure 6: Left: *A set of t-fragments and their linear movement between $t = 0$ and $t = 1$. Three of the t-fragments are potentially visible for the given viewing ray.* Right: *To sample the t-fragments at a specific time $t = 0.25$, we create spherical candidate fragments and check for intersection with the viewing ray.*

3.4. Motion sampling and shading

After collecting all t-fragments we sample in time to generate a final fragment. Given a time sample $\tau \in [0,1]$ we first find the closest intersection with an opaque t-fragment by intersecting the given ray with the time sampled bounding circumspheres of all t-fragments at $t = \tau$ (see Fig. 6 and Fig. 16 in the supplemental material for details). The sphere test is used for fast early rejection. In case of a hit, a tight 3D cube with side length of a pixel in anisotropic NDC is used for the final intersection.

The distance of the closest opaque hit point yields the maximum distance for sampling transparent motion: We collect a fixed number (up to 16 in practice) of intersected transparent t-fragments in a similar way to opaque t-fragments. These fragments are sorted back-to-front for shading using an odd-even mergesort sorting network. To reduce memory overhead, we only store the fragment IDs of the opaque and transparent t-fragments plus the sampled time needed to reconstruct the intersection point.

In the shading pass we interpolate the time varying fragment attributes, in our case the position and normal, and shade the fragment according to the material parameters. We first shade the opaque fragment. Afterwards we successively shade the back-to-front sorted transparent t-fragments and blend them together.

4. Evaluation

We evaluate our approach using a system equipped with an Intel Core i7-3930K, 64GB of RAM, and an NVIDIA Geforce GTX TITAN X with 12GB of RAM. Unless otherwise noted, timings are measured at a resolution of 1024×576 pixels (1120×672 pixels including the 96 pixels guard band). We use different test scenes and animations with varying amount of camera and object motion, number of transparent layers, and geometric complexity (see Table 1 and the additional provided video). Besides the SPONZA scene we use the CHALET and SAN MIGUEL scenes, which have higher geometric complexity. The CHALET scene contains many transparent objects (small detailed leafs of the trees, windows, and the balconies), thus creating many transparent fragments and disocclusions to be handled by our algorithm.

SPONZA	Chalet	San Miguel
262K: 2ms + 0.75ms	4,750k: 2ms + 5.5ms	6,650k: 2ms + 4.7ms

Table 1: *Scenes used for the evaluation, including triangle count, average disocclusion map and t-fragment generation time.*

	Fragments	Scenes		
		Sponza	Chalet	San Miguel
Unordered array	OPAQUE	46.0	275.0	
	TRANSPARENT	46.0	183.0	
FLBVH	OPAQUE - TREE	252.6	758.0	
	OPAQUE - TMP CONSTRUCTION	63.0	190.6	
	TRANSPARENT - TREE	252.6	505.3	
	TRANSPARENT - TMP CONSTRUCTION	63.0	127.1	
Tracing result	JOINT STRUCTURE	991.6		
Motion sampling result	OPAQUE	31.5		
	TRANSPARENT	51.9	260.5	

Table 2: *Maximum memory consumption for each scene in Mbyte at a resolution of 1024×576 (plus guard band) pixels. The amount needed depends primarily on the resolution, the amount of disocclusion, depth complexity, and the number of transparent objects.*

4.1. Memory consumption

In addition to the memory used by the scene itself, we need buffers for the temporary data which are allocated in advance and must handle the peak usage. The early fragment culling using our disocclusion map significantly reduces the number of opaque fragments without losing quality. The size of the unordered array for storing the t-fragments depends on the resolution and amount of disocclusion. A t-fragment consists of 64 bytes: the positions and normals for $t \in \{0,1\}$ as well as the unique albedo. A fixed amount of memory is consumed by the temporary result buffers of the depth-of-field tracing. A ray consists of 16 bytes plus 8 bytes for each t-fragment it collects and stored in a linked list. We use a batchsize of one million rays and limit the number of t-fragments per ray to 128. As seen in Fig. 17, this has no impact on the maximum motion vector. A motion sample uses at least 16 bytes. If the sample contains transparent fragments an additional 68 bytes for at most 16 transparent layers are needed. To limit the memory consumption, we use a batch size of two million motion samples. In addition we assume that only 10% of the samples in a batch contain transparent fragments in case of the Sponza and 50% for the Chalet and San Miguel scene.

4.2. Construction time

For performance reasons and since the t-fragments are not consistent between frames, we need to rebuild the BVH for every frame from scratch. Figure 7 shows the BVH construction time for our test scenes. The construction time for the opaque and transparent BVH for the sponza scene is mostly constant over the camera path. The scene has only a small number of transparent fragments and the

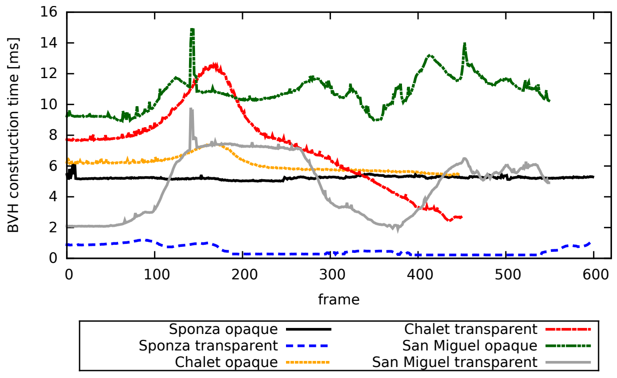

Figure 7: *BVH construction times in milliseconds for each scene.*

disocclusions between the depth layers are not large enough to generate a significantly higher amount of additional opaque fragments. In contrast, the highly detailed leafs of the trees in the Chalet and San Miguel scenes with many transparent fragments create more disocclusions, resulting in more t-fragments and therefore in an increased construction time (see Table 1). As the camera motion increases the amount of disocclusion increases and thereby the number of fragments (see Figure 18 in the supplemental materials for details).

4.3. Motion sampling

First, we compare our algorithm using motion sampling only against a time sampled ray tracing approach. The ray is associated with a time sample t_i during ray traversal and intersected against the BVH at time t_i. Instead we trace between $t = 0$ and $t = 1$ and collect all possible t-fragments that can be intersected during that time and sample in time afterwards. The main benefit is to reduce ray traversal costs while creating additional motion samples. Both approaches work on t-fragments.

Performance Fig. 9 shows timings for construction of the BVH, ray tracing (one ray per pixel), and motion sampling using 4 and 32 samples (top for the Sponza scene and bottom for Chalet). As stated in Sect. 4.2 BVH construction time is nearly constant over the animation. Performance peaks occur around frame 250 and 500 in Sponza as the camera motion between two frames becomes larger (see supplemental video). As a consequence our approach has to collect and process more t-fragments. The same behavior is also visible in the comparison between the time sampled ray tracing and our decoupled motion sampling. Our approach is slower with 4 samples but scales better with higher number of motion samples. Most GPU threads are reading the same data during the motion sample stage and can offset the higher ray traversal costs. The Chalet scene shows similar performance behavior. Overall, performance is slower than in the Sponza scene as more t-fragments are generated (see Fig. 18) due to the camera motion and more complex geometry, especially the leafs and transparent objects.

Quality Fig. 8 illustrates a qualitative comparison between our approach and the time sampled ray tracing. The close-up views (Fig. 8 **(a)** and **(b)**) has no noticeable differences. The difference image (Fig. 11) shows small discrepancy due to the conservative intersection test during our tracing and motion sampling steps. Small errors (in Fig. 11 one pixel in size) are visible at edges due to inaccuracies introduced by the ε radius approximation used in the intersection test.

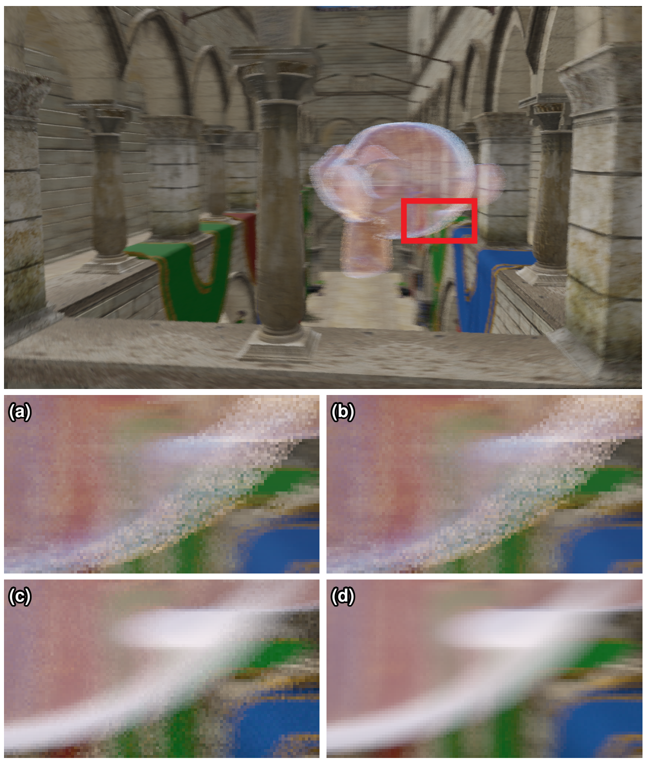

Figure 8: *Quality comparison of motion blur sampling with **(a)** 32spp for our approach compared to time sampled motion blur tracing **(b)**, respectively. The discrepancy compared to the references ((**c**) with 32spp and **(d)** 256spp) rendered with Blender and Cycles resulting from different shading models in particular for the transparent object. The noise on the wall in the detail images ((**a**) and (**c**)) looks similar. The reference images were rendered in 1.5 and 11.5 minutes respectively, using an Intel Core i7-4870HQ.*

Scalability Fig. 12 shows the scalability of our motion sampling approach with respect to resolution. While increasing resolution, more coherent primary rays are generated per ray batch. These follow similar paths in the acceleration structure during ray traversal which results in better cache utilization and execution flow. A similar cache effect can be observed for motion sampling. Therefore, the performance efficiently scales with increasing number of rays and increasing resolution.

4.4. Depth-of-Field

We implemented a simple depth-of-field algorithm that samples the lens aperture [SC97] and traces a fixed number of rays per pixel according to the thin-lens model.

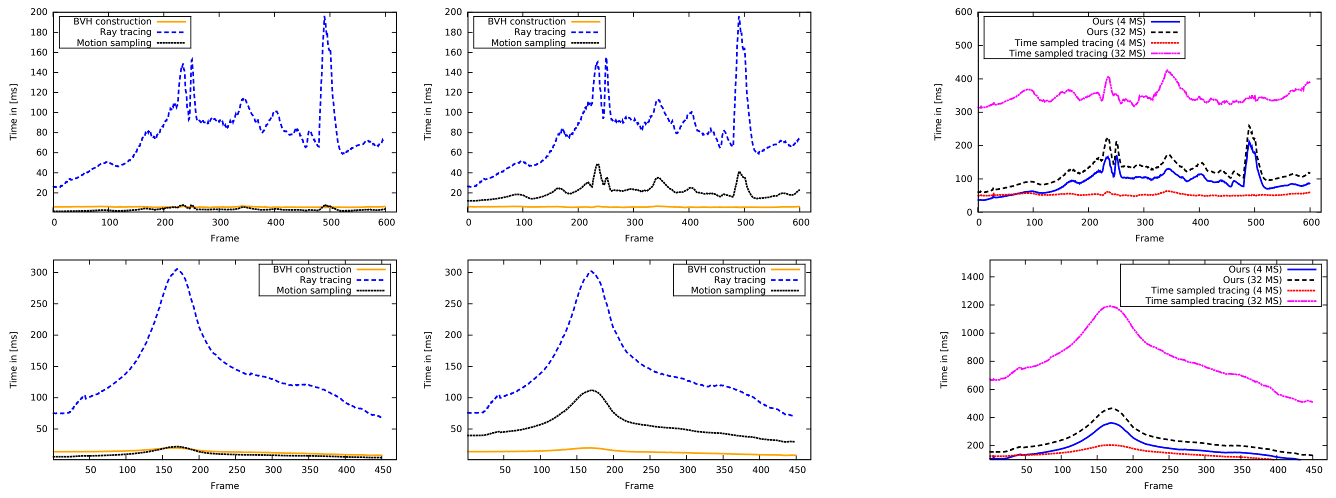

Figure 9: *The figure shows timings for the BVH construction, ray tracing, and motion sampling passes using* ***(left)*** *4 motion samples and* ***(middle)*** *32 motion samples, respectively. Top row plots give timings for the Sponza and in the bottom the Chalet scene. The* ***(right)*** *figure compares overall run-time of our approach with time sampled ray tracing.*

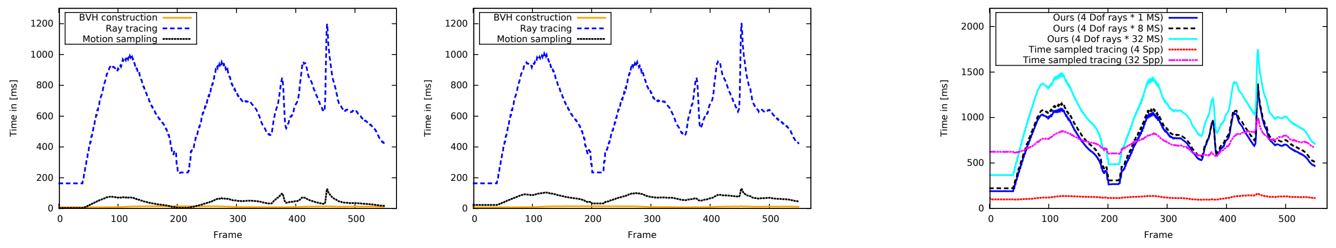

Figure 10: *Timings for the main passes: BVH construction, ray tracing, and motion sampling using* ***(left)*** *4 DoF times 1 motion samples and* ***(middle)*** *4 DoF times 8 motion samples for San Miguel. The* ***(right)*** *figure compares our approach's run-time with time sampled ray tracing.*

Performance The detailed timings for our approach using 4 depth-of-field samples show (see Fig. 10 **(left)** and **(middle)**) that the time for the ray tracing pass increases significantly in this scenario. The dominant part is intersection of t-AABBs compared to interpolated AABBs. The number of incoherent depth-of-field rays (e.g. in case of a large lens radius) reduces cache efficiency and execution flow coherence during ray traversal. The motion sampling pass cannot compensate the tracing overhead as neighboring rays collect less similar t-fragments. Small performance peaks during the motion sampling pass arise when more transparent fragments are generated and shaded. The cost of the BVH construction is negligible. Compared to time sampled ray tracing (Fig. 10 **(right)**) with similar sample set-up our approach is slower during fast motion as the number of collected t-fragments increases which explains the variation in rendering performance. On the other hand it still shows better scalability with increasing number of motion samples.

Quality Figure 13 shows that our approach does not achieve the same quality compared to time sampled ray tracing in the absence of motion. Full 5D sampling (combined spatial and temporal) reduces noise significantly as for each sample new lens positions are generated (see Fig. 14).

5. Discussion and limitations

Since our technique relies on hardware rasterization, co-planar triangles at $t = 1$ are culled by the hardware. This will lead to artifacts or missing samples in the motion blur and depth-of-field reconstruction. Conservative rasterization can reduce the number of missing triangles but not fully solve the issue. A general problem for real-time motion blur algorithms are fast moving objects that are not visible for $t = 1$ as they pop into the next frame. A guard band can reduce this artifact but cannot handle objects coming from behind the camera.

While our approach cannot compete with specialized defocus or motion blur algorithms for real-time applications (e.g. [GMN14]) in terms of performance, it uses a physically plausible approach with opaque and transparent fragments in a unified pipeline. This results a higher rendering quality. Note that these specialized approaches do not fit into our pipeline and can thus not be easily implemented.

Shadows are an integral component when rendering photorealistic images. In the case of renderings with motion blur, hard shadow edges can disturb the visual appearance of the image. While we have not implemented motion blurred shadows, our approach can

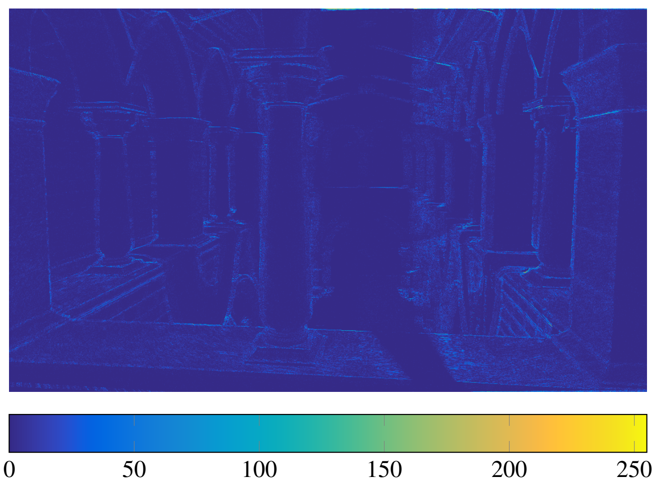

Figure 11: *Absolute differences of the grayscale images in Fig. 8 with 32 samples between our approach and time sampled tracing.*

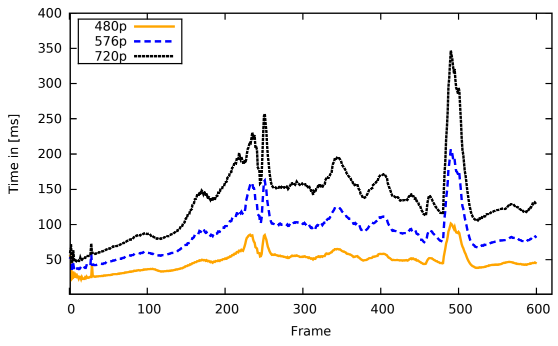

Figure 12: *Scalability w.r.t. resolution for the Sponza scene for solving motion blur only using 1 DoF ray with 32 motion samples.*

be combined with time-dependent shadow maps (*TSM*) as proposed by Akenine-Möller et al. [AMMH07]. Multiple shadow maps have to be rendered where each layer represents a time slice. In the shading pass a texture lookup is performed using the time sample of the fragment to evaluate the visibility. Stratified sampling can be applied to ensure that the samples in the TSM and during shading are close in time.

Since generating depth-of-field rays is expensive in our approach, it does not perform optimal with large depth-of-field, i.e. extensive blur. However, the other parts of our pipeline, in particular the disocclusion map, the *t*-fragments and the BVH construction can be combined with a non decoupled space-time sampling to jointly handle motion blur, transparency and defocus.

If late shading becomes the bottleneck, a shading cache [RKLC*11] could be used but it will lead to artifacts for fast moving objects with reflective materials.

The noise can be reduced by reusing the set of collected *t*-fragments. A new ray with a small jittered lens position has to be created which is very similar to the original ray. For the new ray we

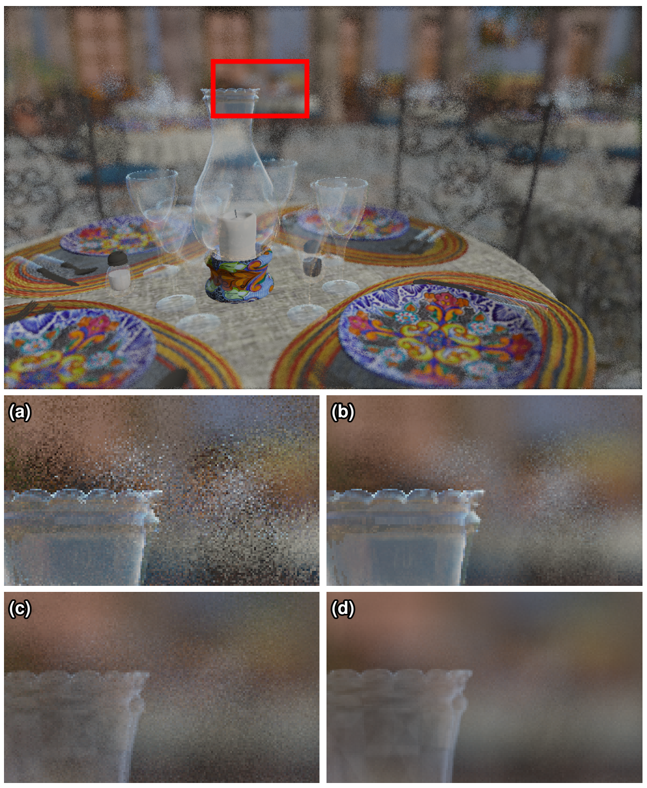

Figure 13: *Quality comparison of our approach* ***(a)*** *using 4 Dof samples times 8 motion samples with time sampled ray tracing* ***(b)*** *using 32 samples. The differences of the Blender references (*(**c**)* 32spp and* ***(d)*** *256spp) to our approach are due to proper 5D sampling. Cycles jitters lens as well as time samples. The reference images were rendered in 2.0 and 14.5 minutes respectively.*

only have to do the motion sampling pass. The results may differ as we cannot resolve all changes in the depth-of-field case and missing disocclusions may generate small artifacts. Geometric aliasing introduced by the rasterization of the *t*-fragments can be reduced by using MSAA and a conservative rasterizer.

6. Conclusion

We presented a rendering approach that unifies handling of order independent transparency, motion blur and defocus in the context of rasterization. Our key idea is to split the sampling phase into independent parts for spatial and temporal sampling. Though gathering all moving fragments, which are potentially relevant for a pixel, before time-sampling causes some overhead, this overhead can be quickly amortized with cheap motion samples. Rendering many of those cheap motion samples reduces the noise introduced by motion blur significantly, strongly improving visual quality. In addition our technique is capable of tracing depth-of-field rays with multiple cheap motion samples for improved visual quality.

Acknowledgements

The work of S. Widmer and D. Wodniok is supported by the 'Excellence Initiative' of the German Federal and State Governments and

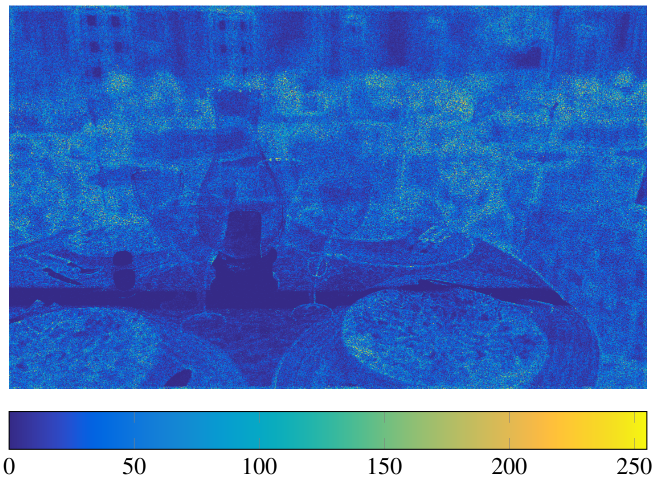

Figure 14: *Absolute differences of the grayscaled images in Fig. 13 with 4 depth-of-field sample times 8 motions samples between our approach and time sampled tracing.*

the Graduate School of Computational Engineering at Technische Universität Darmstadt. We would like to thank Fabian Langguth for helping with the figures in this paper as well as Stepan Konrad for creating the supplemental video. Crytek-sponza scene courtesy of Frank Meinl. San-miguel scene courtesy of Guillermo M. Leal Llaguno.

References

[AL09] Aila T., Laine S.: Understanding the efficiency of ray traversal on gpus. In *HPG* (2009). 5

[AMMH07] Akenine-Möller T., Munkberg J., Hasselgren J.: Stochastic rasterization using time-continuous triangles. In *Graphics Hardware* (2007). 1, 2, 9

[BLF*10] Boulos S., Luong E., Fatahalian K., Moreton H., Hanrahan P.: Space-time hierarchical occlusion culling for micropolygon rendering with motion blur. In *HPG* (2010). 2

[BSS*13] Belcour L., Soler C., Subr K., Holzschuch N., Durand F.: 5D covariance tracing for efficient defocus and motion blur. *ACM Trans. Graph.* (2013). 2

[CM14] Clarberg P., Munkberg J.: Deep shading buffers on commodity GPUs. *ACM Trans. Graph. 33*, 6 (Nov. 2014). 3

[CPC84] Cook R. L., Porter T., Carpenter L.: Distributed ray tracing. In *SIGGRAPH* (1984). 1, 2, 3

[FLB*09] Fatahalian K., Luong E., Boulos S., Akeley K., Mark W. R., Hanrahan P.: Data-parallel rasterization of micropolygons with defocus and motion blur. In *HPG* (2009). 1, 2

[Fri12] Frisvad J. R.: Building an orthonormal basis from a 3D unit vector without normalization. *Journal of Graphics Tools* (2012). 5

[GBAM11] Gribel C. J., Barringer R., Akenine-Möller T.: High-quality spatio-temporal rendering using semi-analytical visibility. In *SIGGRAPH* (2011). 2

[Gla88] Glassner A. S.: Spacetime ray tracing for animation. *IEEE Comput. Graph. Appl.* (1988). 3

[GMN14] Guertin J.-P., McGuire M., Nowrouzezahrai D.: A fast and stable feature-aware motion blur filter. In *HPG* (2014). 1, 2, 8

[GSNK11] Grünschloss L., Stich M., Nawaz S., Keller A.: MSBVH: an efficient acceleration data structure for ray traced motion blur. In *HPG* (2011). 3

[HM99] Hughes J. F., Möller T.: Building an orthonormal basis from a unit vector. *J. Graph. Tools* (1999). 5

[HMV15] Hasselgren J., Munkberg J., Vaidyanathan K.: Practical layered reconstruction for defocus and motion blur. *Journal of Computer Graphics Techniques 4*, 2 (June 2015). 2

[HQL*10] Hou Q., Qin H., Li W., Guo B., Zhou K.: Micropolygon ray tracing with defocus and motion blur. In *SIGGRAPH* (2010). 2

[HW94] Haines E., Wallace J.: Shaft culling for efficient ray-cast radiosity. In *Photorealistic Rendering in Computer Graphics*. 1994. 5

[Kar12] Karras T.: Maximizing parallelism in the construction of bvhs, octrees, and k-d trees. In *HPG* (2012). 3, 5

[LAC*11] Lehtinen J., Aila T., Chen J., Laine S., Durand F.: Temporal light field reconstruction for rendering distribution effects. In *SIGGRAPH* (2011). 2

[LAKL11] Laine S., Aila T., Karras T., Lehtinen J.: Clipless dual-space bounds for faster stochastic rasterization. In *SIGGRAPH* (2011). 2

[LES09] Lee S., Eisemann E., Seidel H.-P.: Depth-of-field rendering with multiview synthesis. In *SIGGRAPH Asia* (2009). 1

[LGS*09] Lauterbach C., Garland M., Sengupta S., Luebke D., Manocha D.: Fast BVH construction on GPUs. In *EG* (2009). 3, 5

[MESL10] McGuire M., Enderton E., Shirley P., Luebke D.: Real-time stochastic rasterization on conventional GPU architectures. In *HPG* (2010). 1, 3

[MVH*14] Munkberg J., Vaidyanathan K., Hasselgren J., Clarberg P., Akenine-Möller T.: Layered reconstruction for defocus and motion blur. *CGF 33*, 4 (July 2014). 2

[NCJ*12] Nilsson J., Clarberg P., Johnsson B., Munkberg J., Hasselgren J., Toth R., Salvi M., Akenine-Möller T.: Design and novel uses of higher-dimensional rasterization. In *HPG* (2012). 1

[NSG11] Navarro F., Serón F. J., Gutierrez D.: Motion Blur Rendering: State of the Art. *CGF* (2011). 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 Trans. Graph.* (2011). 9

[SC97] Shirley P., Chiu K.: A low distortion map between disk and square. *J. Graph. Tools* (1997). 3, 7

[SFD09] Stich M., Friedrich H., Dietrich A.: Spatial splits in bounding volume hierarchies. In *HPG* (2009). 3

[SRP*15] Selgrad K., Reintges C., Penk D., Wagner P., Stamminger M.: Real-time depth of field using multi-layer filtering. In *I3D* (2015). 1, 2

[VTS*12] Vaidyanathan K., Toth R., Salvi M., Boulos S., Lefohn A.: Adaptive image space shading for motion and defocus blur. In *HPG* (2012). 2

[Wal07] Wald I.: On fast construction of sah-based bounding volume hierarchies. In *IEEE IRT* (2007). 2

[WPS*15] Widmer S., Pająk D., Schulz A., Pulli K., Kautz J., Goesele M., Luebke D.: An adaptive acceleration structure for screen-space ray tracing. In *HPG* (2015). 1

[WWTS15] Wu Y.-J., Way D.-L., Tsai Y.-T., Shih Z.-C.: Clip space sample culling for motion blur and defocus blur. *Journal of Information Science and Engineering 31*, 3 (2015). 3

[YWY10] Yu X., Wang R., Yu J.: Real-time depth of field rendering via dynamic light field generation and filtering. In *CGF* (2010). 1