Filtered Stochastic Shadow Mapping using a Layered Approach

M. Andersson[1,2], J. Hasselgren[1], J. Munkberg[1], and T. Akenine-Möller[1,2]

[1]Intel Corporation [2]Lund University

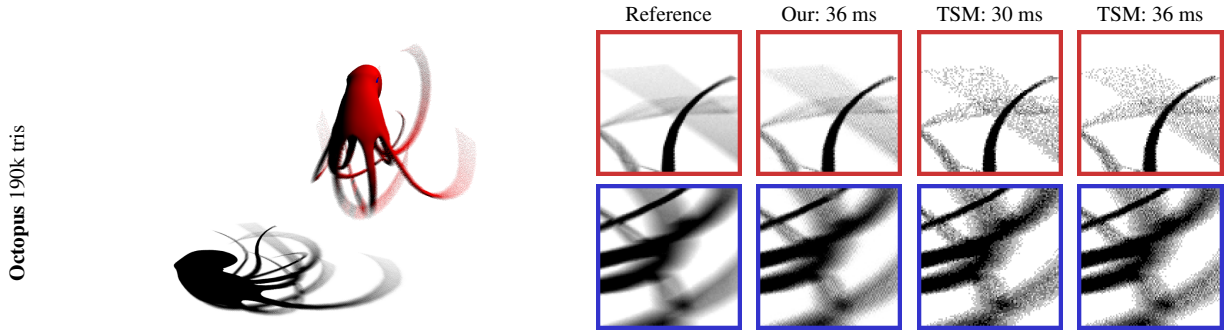

Figure 1: *An octopus in motion casting a complex motion blurred shadow rendered by our algorithm. With the same input samples, our algorithm has significantly less noise compared to time-dependent shadow maps (TSM). At equal time, the noise level is still largely reduced. The animated octopus mesh is taken from the Alembic source distribution.*

Abstract

Given a stochastic shadow map rendered with motion blur, our goal is to render an image from the eye with motion blurred shadows with as little noise as possible. We use a layered approach in the shadow map, and reproject samples along the average motion vector, and then perform lookups in this representation. Our results include substantially improved shadow quality compared to previous work and a fast GPU implementation. In addition, we devise a set of scenes that are designed to bring out and show problematic cases for motion blurred shadows. These scenes have difficult occlusion characteristics, and may be used in future research on this topic.

Categories and Subject Descriptors (according to ACM CCS): I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Color, shading, shadowing, and texture

1. Introduction

Motion blur in photographed images, offline rendering, and in real-time graphics provides the viewer with a sense of motion direction and also reduces temporal aliasing [NSG11]. When motion blur is present, the shadows of moving objects should be motion blurred as well. However, while shadow rendering has received a lot of attention in the research community for static scenes [ESAW11], rendering of motion blurred shadows has remained relatively unexplored.

Accumulation buffering can be used to generate motion blurred shadows [HA90], but the algorithm only supports using identical sample times for all pixels in the shadow map [Wil78]. This often shows up as banding artifacts unless many full-screen passes are used, and that is often not possible within the time budget for real-time rendering. Instead, one can use stochastic sampling when generating the shadow map [AMMH07]. This removes banding artifacts by allowing samples to have unique times, but the resulting image is often too noisy at affordable sample rates. Deep shadow mapping [LV00] is a technique originally intended for rendering shadows for hair, smoke, fur, etc. Motion blurred

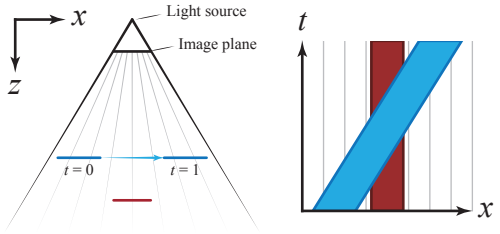

Figure 2: *Left: a blue line segment is moving parallel to the image plane of the light, in front of a static red line segment. Right: the light space* epipolar image*, i.e., a spatio-temporal plot in x and t. The blue segment partially occludes the red line for a certain time interval.*

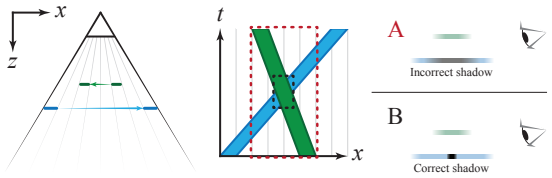

Figure 3: *Left: two objects are moving in opposite directions at different distances from the light source. Middle: the light space epipolar image reveals that the green segment only covers the blue segment for a brief moment, as indicated by the black, dotted outline. By omitting time information, the green occluder appears to cast a shadow on the blue receiver throughout the entire exposure time, as indicated by the red dotted outline. Right: collapsing the time dimension produces a smeared shadow* (A)*. The correct shadow is generated by taking temporal occlusion into account* (B)*.*

shadows can also be generated using deep shadow maps, however, they are only correct when the receiving object is static. A thorough review of previous work can be found in Section 2.

Since the receiver, the shadow caster, and the light source can all move at the same time, rendering motion blurred shadows is a notoriously difficult problem, and largely unsolved in the domain of real-time rendering. Our approach to motion blurred shadows is to generate a time-dependent stochastic shadow map (TSM) [AMMH07] from the light source, and then develop a novel filtering algorithm in order to reduce shadow noise. In our algorithm description, we make extensive use of epipolar images (see Figure 2). Examples of our results can be seen in Figure 1. We also implement our algorithms on a graphics processor and provide interactive or near real-time rendering performance with substantially reduced noise levels compared to TSM, and in addition, our algorithm handles more cases correctly than previous methods.

2. Previous Work

The number of publications for shadow rendering is huge, and for a comprehensive overview, we refer to the book by Eisemann et al. [ESAW11]. For an overview of motion blur in graphics, we refer to the state-of-the-art report by Navarro et al. [NSG11]. Most of the shadow rendering algorithms have been developed for static scenes, while the number of methods that incorporate motion blur is sparse.

Akenine-Möller et al. introduce time-dependent shadow mapping (TSM) [AMMH07], where multiple shadow maps are created using stochastic rasterization. Each shadow map represents a time slice of the full exposure and stratified sampling is used to ensure that exactly one sample falls into each time slice. In a second pass, the scene is rendered using stochastic rasterization from the camera's point of view. Again, stratified sampling is used to draw one sample from each time slice. The shadow test is performed using the shadow map sample at the time slice corresponding to the camera sample. This means that they are close in time and, given a static or slowly moving light source, in space, though they do not match exactly.

The idea behind deep shadow mapping (DSM) [LV00] is to store multiple semi-transparent occluders in each pixel of a shadow map. The visibility of a point, $\mathbf{p}$, can then be computed as $\prod_{p_z<z_i}(1-\alpha_i)$, where z_i and α_i are the depth and transparency of the i:th occluder. The authors show that DSM can be used for motion blurred shadows by treating each moving occluder as a semi-transparent layer. The scene is rendered from the light's point of view using stochastic rasterization. For each pixel in the shadow map, the samples are clustered into distinct depth layers and each layer is given an opacity proportional to the number of samples drawn from that layer. The layers are then encoded into the DSM and are treated like transparent occluders. Since the time dimension is collapsed, i.e., motion blur is treated as transparency, certain problems can occur, e.g., for moving receivers. An example of this problem is illustrated in Figure 3.

McGuire and Enderton [ME11] present an alternative to DSM called colored stochastic shadow mapping (CSSM), where a transparent layer is encoded stochastically by allowing a proportional amount of shadow map samples to pass through the transparent layer, and instead get the depth value of the layer behind, similar to stochastic transparency. The shadow map lookup is filtered by drawing a number of spatially coherent samples and using the averaged shadow term. Stochastic motion blur rasterization and stochastic transparency is analogous, and therefore the samples generated by a stochastic rasterizer could be used directly as input to the CSSM algorithm to extend it to handle motion blurred shadows. However, similar to DSM, the time dimension is collapsed in CSSM, and hence inherits the same problems.

While the results of multiple depth tests can be filtered

together [RSC87], filtering the depth values prior to the depth test, however, will not yield the correct result. The problem of creating a filterable shadow map representation has been given a lot of attention, most notably variance shadow mapping (VSM) [DL06], convolution shadow mapping (CSM) [AMB*07], and exponential shadow mapping (ESM) [AMS*08]. VSMs have also been extended to render plausible soft shadows [YDF*10], where the filter size is computed based on the average occluder distance to the receiver and the light source [Fer05]. We rely on variance shadow maps to create a representation where the time dimension can be efficiently filtered. We also experimented with using ESM, but saw no convincing benefit in neither quality nor performance. As our shadow map representation is split into a set of depth layers, we avoid most of the shine-through artifacts from VSMs, similar to layered variance shadow maps [LM08].

Lehtinen et al. [LAC*11] present a motion and defocus blur reconstruction algorithm, which also supports the case of motion blurred and soft shadows by taking the entire 7D light-field into account during reconstruction. Their algorithm uses a dual acceleration structure, one for the samples visible from the camera, and a second for the depth as seen from the light source. When reconstructing a camera space point $\mathbf{p} = (x, y, t)$, the corresponding light space sample at (l_x, l_y, t) is reconstructed and a binary shadow test is performed. While their algorithm produces very high quality images, results rely on reconstructing and filtering many ($\sim$128) locations per pixel. The reconstruction time is over 10 seconds even for high-end discrete GPUs on 5D examples (excluding motion blur shadows).

Egan et al. [ETH*09] use frequency analysis [DHS*05] of the motion blurred light field to derive sheared filters for reconstruction and to drive adaptive sampling. Frequency analysis can also be used to derive filters and adaptive sampling techniques for shadows from complex occluders [EHDR11], directional occlusion [EDR11], soft shadows filtering [MWR12], diffuse indirect lighting [MWRD13], and multiple distribution effects [MYRD14].

Our shadowing approach has similarities with recent reconstruction algorithms, most notably the work by Munkberg et al. [MVH*14] and Hasselgren et al. [HMV14], who use a similar layered representation and sheared $5D$-filter to reconstruct *primary visibility* for motion blur and depth of field. Their work does not handle reconstruction of motion blurred shadows and as such, our work can be seen as an important complement. In addition, we extend on their work by taking into account the correlation between time of samples in the shadow map and samples relating to primary visibility. We also propose a novel filtering approach.

3. Theory

In this section, we briefly present the theory behind our setup for motion blurred shadow rendering. The outgoing radiance from a point x in a direction ω_o is given by:

$$l(x, \omega_o) = \int_{\Omega_i} v(x, \omega_i) f(x, \omega_i, \omega_o) l(\omega_i) d\omega_i, \qquad (1)$$

where v is the visibility function, f is the BRDF (including the cosine term), and $l(\omega_i)$ is the incoming light. We only consider direct lighting from a set of discrete point or directional light sources, $\{l_i\}$, and the expression for the outgoing radiance can therefore be simplified to a sum over the light sources:

$$l(x, \omega_o) = \sum_i v(x, \omega_i) f(x, \omega_i, \omega_o) l_i. \qquad (2)$$

Now, we assume that the scene is dynamic, where the objects, lights, and the camera can move. The corresponding expression for the outgoing radiance at a certain time, t, can be expressed in a coordinate system following x as:

$$l(x, \omega_o(t), t) = \sum_i v(x, \omega_i(t), t) f(x, \omega_i(t), \omega_o(t)) l_i. \qquad (3)$$

This expression shows the outgoing radiance at x, but does not take occlusions into account between x and the camera.

To render motion blurred shadows, we want to evaluate the occlusion term $v(x, \omega_i(t), t)$. In a ray tracer, one can simply answer this query with a shadow ray through the dynamic scene. In a stochastic rasterizer, one can instead query a time-dependent shadow map [AMMH07], which stores a light space depth value (z) for each spatio-temporal coordinate (x_l, t). To do this, the query coordinate x and direction $\omega_i(t)$ is remapped into the moving coordinate system of the light (denoted with subscript l): $(x, \omega_i(t), t) \mapsto (x_l(t), t)$. If the shadow map depth is smaller than the light space depth of x, the light source, l_i, is occluded from x at time t.

To determine the color for each pixel, we want to integrate over t to compute a blurred value over the open interval of the camera shutter. Hence, due to motion and the spatial pixel filter, many points will contribute to the blurred radiance value of each pixel. The shading evaluation may include multiple shadow map lookups within a spatio-temporal footprint, as illustrated in Figure 4.

Furthermore, there may be discrete changes in primary visibility, as different primitives move over the pixel's view frustum over the temporal interval. In the general case, the camera, all objects, *and* all lights may move in time. To approximate this result, one often take a large number of spatio-temporal Monte Carlo (MC) samples. However, with an estimate of the footprint in the spatio-temporal shadow map of hit points on visible primitives, one can apply filtered lookups in order to reduce shadow noise. This is the main goal in this paper.

Due to perspective motion, $x_l(t)$ may be a rational polynomial in t. However, similar to recent motion blur filters [ETH*09, MHBO12, MVH*14], we make a linear motion assumption in our shadow map representation.

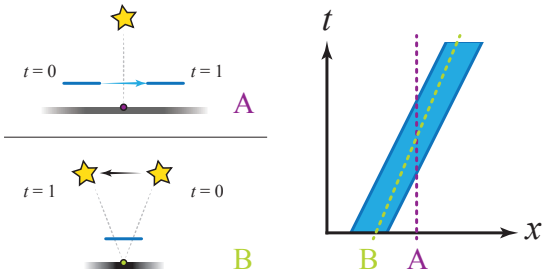

Figure 4: *Left: scene* A *shows a static light source and a moving occluder, while scene* B *instead shows a moving light and a static occluder. Right: the two different scenes produce the same light space epipolar image, yet the observed shadows are very different. This is a consequence of the receiver being static in light space in scene* A*, while the receiver point move in scene* B *(due to the moving light source). The footprint for the receiver point is shown by the dashed lines in the epipolar image.*

4. Algorithm

We propose a layered, filtered shadow mapping algorithm for motion blurred shadows. The algorithm is divided into two passes, namely a *shadow pass* and a *lighting pass*. The shadow pass renders the scene using stochastic rasterization [AMMH07] and generates a time-dependent shadow map augmented with per-sample motion vectors. The subsequent lighting pass renders the scene from the camera's point of view, and performs a shadow query for each sample seen from the camera.

In contrast to time-dependent shadow mapping (TSM), wherein the shadow query gives a binary result, i.e., if the sample is in shadow or not, our algorithm estimates the temporal integral of the visibility term (discussed in Section 3), which results in smoother motion blurred shadows.

Our algorithm is based on the assumption that the motion in a small spatial region of a depth layer is slowly varying, which has been a successful approximation in previous work [MVH*14]. During the shadow pass, we divide the stochastic shadow map into texture space tiles, and split samples of each tile into depth layers. We then process each such tile and depth layer individually.

First, we compute an average motion vector $\mathbf{d}$ for each depth layer in each tile. If all spatio-temporal samples, $\{(\mathbf{x}_i, t_i)\}$, in the depth layer move with the same motion vector $\mathbf{d}$, then each sample's movement is described by the following equation:

$$\mathbf{x}_i(t) = \mathbf{x}_i + \mathbf{d}(t - t_i). \tag{4}$$

At $t = 0.5$, a sample has a spatial coordinate:

$$\mathbf{x}'_\mathbf{i} = \mathbf{x}_i + \mathbf{d}(0.5 - t_i). \tag{5}$$

With this observation, we create a compact time-dependent shadow map by *reprojecting* all samples along the depth layer's average motion vector, $\mathbf{d}$, to $t = 0.5$ using Equation 5, and storing this layered, reprojected, shadow map in memory for use in the subsequent lighting pass. To perform a shadow lookup in the lighting pass, we offset this representation along the layer's motion vector to get the depth layer represented at a particular time.

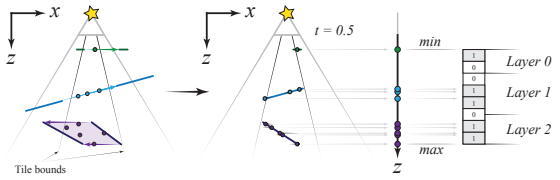

Figure 5: *A simple scene illustrating the clustering approach that we use. The minimum and maximum depths, z, as seen from the middle of the shutter interval, $t = 0.5$, are found, and this interval is split uniformly. Each bin that contains at least one sample is marked with a 1. Finally, the resulting bit mask for the bins is used to find a small set of depth layers.*

4.1. Shadow Pass

Creating the shadow map representation involves a number of steps, which are covered in detail in this section.

Visibility sampling First, the scene is stochastically rasterized with N samples per pixel in (x, y, t) light space. For each sample, we store depth and motion vectors. The motion vectors are comprised of the shadow map texture space motion in xy and depth motion in z.

Depth clustering Next, the samples in a tile in shadow map texture space are clustered in depth to obtain a set of depth layers of samples. To find suitable depth layers, we perform a simple depth clustering [AHAM11] step over all samples within a search window centered around each tile. We perform the layer split at the middle of the exposure interval, by offsetting the sample depths to $t = 0.5$ with their respective motion vector. Next, the depth range $[z_{min}, z_{max}]$ of the relocated samples is computed, which is then subdivided into uniform intervals. Intervals containing samples are flagged as *occupied*. Layer delimiters are then introduced where the largest stretches of *unoccupied* intervals are found. This process is illustrated in Figure 5. In our current GPU implementation we use 64 uniform intervals, which are clustered into (up to) four depth layers.

Per-layer motion We assume that the motion is slowly varying within each depth layer of the tile. We find a common representable motion vector, $\mathbf{d}$, for the layer by averaging the motion vectors of the samples in the layer.

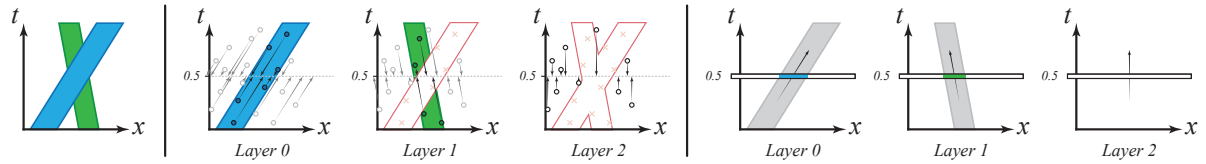

Figure 7: *Left: an epipolar image of a simple scene with two objects and a static background layer. Middle: the samples are reprojected to $t = 0.5$ for each layer, using the layer's motion vector. Samples within a layer (black outlines) increase the opacity. Samples behind the layer (gray outlines) (farther away from the light source) decrease the opacity. Samples in front of the layer (red crosses) are discarded. Right: the shadow map can be queried at different times using the reprojected samples at $t = 0.5$ for each layer along with the layer's motion vector*

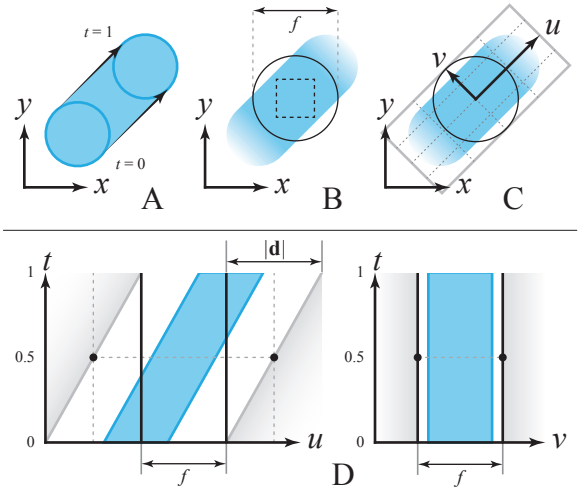

Figure 6: A: *a moving disc, viewed from the light source.* B: *we consider a single tile in the middle of the image. Samples from within the guard band will be used for this tile.* C: *the average motion direction is found, and a new coordinate system, (u,v), is constructed. The u-axis is scaled and aligned with the motion vector.* D: *the epipolar images for $v = 0$ (left) and $u = 0$ (right). The final, reprojected layer representation contains samples within the gray borders, which are derived from the guard band size, f, and the motion vector,* **d**.

Grid setup The reprojection step builds upon previous work for motion blur filtering [MVH*14], with the difference that we work with depth values instead of color. However, unlike previous work, we reproject onto a stretched grid, which is aligned with the average motion direction, **d**. For clarity of presentation, we let the (x,y) coordinates have origin at the center of the tile. We parameterize the stretched grid with coordinates, (u,v), where the u-axis is aligned with the layer's motion vector, **d**, and the v-axis is perpendicular. As illustrated in the epipolar images in Figure 6, any sample originating from within the *guard band*, f, of a tile may reproject anywhere within a region that is $f + ||\mathbf{d}||$ wide. Therefore, we use a grid scaling factor of $\frac{f}{f+||\mathbf{d}||}$ along the u-direction, which ensures that no samples will be reprojected outside our scaled grid. We now define a rotation and scaling transform, M, for each layer, such that $M\mathbf{d} = \left(\frac{f||\mathbf{d}||}{f+||\mathbf{d}||}, 0\right)$. If we apply this transform to a moving sample: $\mathbf{x}_i(t) = \mathbf{x}_i + \mathbf{d}(t - t_i)$, cf. Equation 4, we obtain the corresponding sample in the stretched grid as:

$$(u_i(t), v_i) = M\mathbf{x}_i + \left(\frac{f||\mathbf{d}||}{f+||\mathbf{d}||}, 0\right)(t - t_i). \qquad (6)$$

In presence of motion, the grid stretches outside the tile bounds, as shown in Figure 6D. The scaling factors of M used in our implementation are discussed further in Section 5.

Sample reprojection For each layer, each sample is transformed to the local coordinate system and is moved along the layer's motion vector to the middle of the shutter interval, by using Equation 6 with $t = 0.5$. Additionally, the depth of the sample at $t = 0.5$, $z_i^{\text{reproj}} = z_i + (0.5 - t_i)d_z$ is computed. The sample's position in uv-space maps to a texel location.

The next step is to compute the coverage and depth contribution of the sample to the texel. We track four quantities which are used for filtering later in the lighting pass described in Section 4.2. First, we need the depth value for the shadow test. We use the filterable variance shadow map (VSM) representation with the first and second depth moments (z and z^2) [DL06]. The next quantity is the opacity, α, which tells us how much each texel in each layer should contribute to the final shadow result. The remaining quantity is the weight, w, of the filter kernel used in the reprojection. Each sample will contribute with different α_i and w_i values for each layer. For each texel, (u,v), in the shadow map, these quantities are accumulated into a tuple with four elements on the form:

$$T(u,v) = \left(\sum w_i\alpha_i z_i, \sum w_i\alpha_i z_i^2, \sum w_i\alpha_i, \sum w_i\right), \qquad (7)$$

i.e., a weighted sum of the first and second depth moments, a weighted opacity, and the total weight.

The values of w_i and α_i for the current layer are calculated as follows. If a sample lies in or behind the current layer (i.e., farther away from the light source), then w_i has a non-zero value based on the filter used. Otherwise, $w_i = 0$

(i.e., the sample does not contribute to this layer). In our implementation we use a box filter, and thus w_i corresponds to the number of samples falling in a pixel. The opacity value is one ($\alpha_i = 1$) if the sample lies within the layer, and is zero otherwise. The idea behind this is that if a sample that belongs to a background layer is visible through the foreground layer, then the foreground layer must be transparent for that sample. Since fewer samples affect layers farther back, this implies that the opacity estimate is better for foreground layers. This accumulation strategy is discussed in further detail in Vaidyanathan et al.'s [VMCS14] reconstruction work. Figure 7 illustrates a simple example of the reprojection process.

4.2. Lighting pass

In the lighting pass, the scene is rendered from the camera using stochastic rasterization [AMMH07], and we search for the amount of light that reaches a *receiver* sample, $(\mathbf{x}_r, z_r, t_r)$. For every receiver sample, the corresponding tile in the shadow map is found, and the visibility contribution of its layers are combined to a final shadow term.

Our shadow map is compactly represented as a set of tiles with a set of layers at $t = 0.5$ with the accompanying coordinate transforms, M. To retrieve a shadow map value for a particular layer at receiver time t_r, the receiver sample is reprojected using Equation 6 with $t = 0.5$. The reprojected coordinate maps to a location in the shadow map.

Furthermore, we account for the camera filter footprint when performing a shadow map lookup. Since the camera, light, and receiver point may move, this is an anisotropic footprint in xyt. We make the assumption that the receiver point is static in camera space for a short duration around the receiver sample time t_r. The duration is inversely proportional to the number of samples per pixel, N, used in the lighting pass. The camera filter footprint is approximated by transforming the receiver point to light space at times $t_r - \frac{1}{2N}$ and $t_r + \frac{1}{2N}$. Figure 8 shows two examples of how footprints are computed. Should the footprint stretch outside the layer's allotted region in the shadow map, it is clamped to avoid fetching invalid data from a neighboring tile. The size of the guard band, f, used in the coordinate transform M, determines how far outside the original tile the footprint may stretch, as illustrated in Figure 6.

The shadow map location, along with the footprint axes as gradient vectors, are used in the hardware anisotropic filtering to retrieve a tuple on the form given in Equation 7. From this, we derive:

$$\bar{z} = \frac{\sum w_i \alpha_i z_i}{\sum w_i \alpha_i}, \quad \overline{z^2} = \frac{\sum w_i \alpha_i z_i^2}{\sum w_i \alpha_i}, \quad \bar{\alpha} = \frac{\sum w_i \alpha_i}{\sum w_i}. \tag{8}$$

With $\bar{z}$ and $\overline{z^2}$, we compute a *visibility term*, V, using a standard variance shadow map (VSM) test [DL06] with two moments. We base the test on the receiver point depth moved

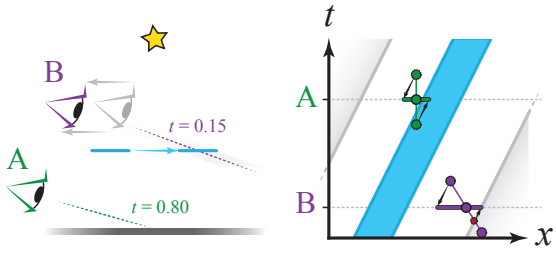

Figure 8: *Two examples of how we calculate the filter footprint for a moving layer. In case* A*, the camera and the light are static, and thus the filter in the light space epipolar image is a vertical line. Reprojecting the filter end points along the motion vector gives the final filter footprint. In case* B*, the camera is moving in the opposite direction to the layer motion and thus produces a slanted trail in the light space epipolar image. The footprint stretches outside the region that can be reconstructed for this layer, and is clamped. In these examples, we use* 4 *samples per pixel, and thus have filter footprints stretching over* $\frac{1}{4}$ *of the time interval.*

to the reprojected shadow map time $z_r^{\text{reproj}}(t_r) = z_r + (0.5 - t_r)d_z - b$, where t_r is the receiver sample time and b is a VSM shadow bias term. It should be noted that moving the receiver sample to $t = 0.5$ is equivalent to moving the depth of the shadow casting sample to the time of the receiver sample, t_r. Given $\bar{z}$ and $\sigma^2 = \overline{z^2} - \bar{z}^2$, the variance shadow map visibility is computed as follows [DL06]:

$$V = \frac{\sigma^2}{\sigma^2 + (z_r^{\text{reproj}} - \bar{z})^2}. \tag{9}$$

Combined with the opacity of the layer at the point of lookup, $\bar{\alpha}$, we can approximate the visibility of the receiver point through this particular layer as:

$$V_l = 1 - \bar{\alpha}(1 - V). \tag{10}$$

The visibility through all layers is accumulated using $V_{total} = \prod_l V_l$ to get a final visibility approximation.

5. Implementation

We implemented our algorithm and TSM in a GPU software stochastic rasterizer, similar to McGuire et al. [MESL10], which we have extended with time-dependent shadow maps (TSMs) [AMMH07] and faster coverage tests [LK11].

Apart from the tile size and guard band used to derive the scaling factors in the coordinate transform, M, in Equation 6, an additional parameter, o, is used, such that:

$$(s_u, s_v) = \frac{o}{f}\left(\frac{f}{f + ||\mathbf{d}||}, 1\right), \tag{11}$$

where s_u and s_v are the scaling factors in the matrix, M. Here, o controls the resolution of the output grid, and thereby also

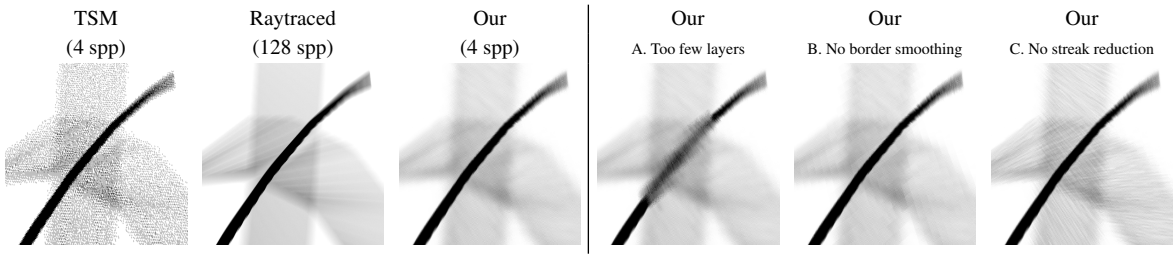

Figure 9: *This figure shows the need for the various parameters and additional smoothing used in our algorithm. The camera is positioned so that the shadowed region in the red (top) inset from Figure 1 is viewed closely from above (instead of from an angle). Three tentacles are moving in different directions and create complex shadows on the receiver floor. From the raytraced reference, the detailed geometry from the suction cups are visible as they produce a striped pattern in one of the shadow layers. We use the same sample set for* TSM *and our algorithm to create the shadow. In inset* (A)*, we have reduced the number of maximum layers from* 4 *to* 3*, and subsequently a single depth layer is created from two of the tentacles, with an incorrect shadow as the result. Similar artifacts are visible when the clustering step fails to produce proper splits between layers with vastly different motion and/or depth characteristics. Inset* (B) *shows our algorithm without the stochastic neighbor selection enabled, leading to visible tile boundaries. Finally, in inset* (C)*, the v-direction filter for large motion is disabled, with some streaking artifacts as a consequence. In this particular case, note that there are some streaks in the reference image, which we slightly over-blur with the v-filter enabled. Also note that, in general, one might want to increase the shadow map resolution and the number of samples per pixel to increase the quality further.*

the resolution of the shadow map. In our implementation, for each 8×8 pixel tile in the input depth and motion maps, we use an output shadow map tile size $o = 16$, and a search region $f = 16$. Or more intuitively, in absence of motion, for each input pixel in an $f \times f$ neighborhood around each tile, $\frac{o}{f}$ output texels are produced. As motion increases, the output grid grows proportionally, while its resolution in the shadow map is retained, essentially trading spatial detail for covering a larger volume in xyt, as illustrated in Figure 6.

During the lighting pass, described in Section 4.2, we may sometimes end up with a total weight of zero, $\sum w_i = 0$. This happens when the shadow map density is too sparse compared to the filter (or size of motion), and is more likely to happen for layers further back as most of the samples will be caught by the layers in front. We alleviate this problem by expanding the filter size until a valid sample ($w_i \neq 0$) is included. This is similar to how the problem is solved in previous work [MVH*14]. They use an exponential filter and therefore samples further away can be used if there are no local samples, and giving them an insignificant weight if local samples exist.

Furthermore, we note that Munkberg et al. [MVH*14] exhibit some streaking artifacts in regions with large motion and low circle of confusion. In order to reduce such streaking artifacts, we apply a small filter aligned with the v-axis in such tiles. In our current implementation and selection of scenes, we found that a small tent filter with pixel width $w_v = \max(0, 0.05\,(||\mathbf{d}|| - 2))$, where $\mathbf{d}$ is the average motion vector, works well in practice.

In areas with varying motion vectors, we may get tile artifacts due to the rotated grids not aligning at tile borders. We remedy this using an approach similar to the one proposed by Guertin et al. [GMN14]. When performing a shadow map lookup, we compute a probability based on the distance between the lookup position and the tile border. We then use that probability to stochastically perform the lookup in either the current tile or the neighboring tile. This means that the region in which a valid shadow test can be performed for a tile in somewhat increased, i.e., it depends on that a sufficiently large f-parameter is selected. In practice, we found that using a linear ramp starting at $1/3$ from the tile center and going up to 50% probability for selecting a neighbor at the tile border produces visually pleasing results, and works well with the selected $f = 16$.

In Figure 9, the benefits of border smoothing and streak reduction are shown. In addition, we show the quality impact of using too few layers.

Memory Consumption The memory requirement for the shadow map depends on the number of layers, the output tile sizes, and the input depth and motion map resolution. In addition, for each tile and layer, we store the transform M using two values (scale and direction of major axis), the motion in z and the layer split position. We use 32 bit floats for the four tuples in Equation 7, and 16 bit floats for the tile data. For a 1024^2 pixel shadow map at 4 spp, the total cost amounts to 90 MB per depth layer. TSM stores one depth value for each sample, and the same input consumes 16 MB. It should be noted that we have optimized our algorithm for speed rather than size. There are, however, several avenues for reducing the memory usage. For example, using ESM instead of VSM, reducing the precision of the opacity and weight in Equation 7, or dynamically allocating output tiles.

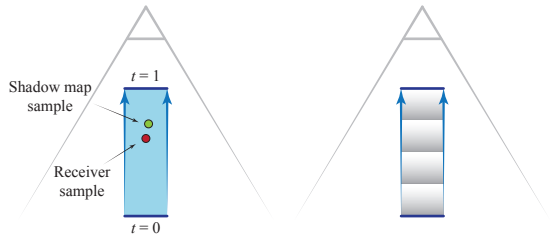

Figure 10: *An illustration of a self shadowing issue with TSM, where a surface is moving towards the light source. The surface should be fully lit since it is visible from the light source throughout the animation. Left: the shadow map lookup for the receiver sample (red) returns the closest shadow sample (green) in (x,y). Since the shadow sample is closer to the light source than the receiver sample, it will* falsely *report that the sample is in shadow. Right: within each time slice, the probability of encountering an occluder sample increases with the distance to the light source. In the resulting image, the time slices are visible as striped self-shadows. Our dynamic bias decreases the likelihood of such events.*

We leave this for future work since it would require further investigation on how these changes would affect the shadow quality.

5.1. Dynamic bias for TSM

In TSM, the exposure time is partitioned into N slices, each containing a subset of the samples. When a shadow test is performed for a sample at a particular time, the corresponding enclosing pixel and time slice in the shadow map is found. However, due to the discretization, there is a discrepancy between the receiver sample time and the shadow map sample time. This difference may be as large as $\frac{1}{N}$ of the exposure time, and can lead to self shadowing artifacts, as illustrated in Figure 10. In some cases, a shadow map bias alleviates the problem, where the bias incorporates the distance in depth that any object travels. This quickly becomes problematic, since contact shadows will disappear when the bias is increased.

As discussed in Section 4, in our algorithm, we compensate for the discrepancy in time between the receiver sample and the shadow map sample. Depth motion is accounted for by moving the samples along the average motion vector's depth component based on the sample time difference. We improve TSM in a similar way, but use the samples' own motion vectors. The receiver point is translated to the shadow map sample's time to get a new receiver depth:

$$\dot{z}_i = z_i + d_{zi}(t_r - t_i), \tag{12}$$

In Section 6, we will show how this adjustment improves image quality of TSM and makes motion blurred shadow map rendering more robust.

6. Results

We have constructed a set of scenes, shown in Figures 11, 12 and 13, to test difficult cases, where both the receiving and shadow casting geometry move relative to the light source in various ways. The pillar scene illustrates the difficult case with a moving light source. Although the pillars are static as seen from the camera, they move a large distance in light space. The other two scenes show variations of moving receivers and shadow casters.

To evaluate the quality and robustness of our algorithm, we compare the quality against time-dependent shadow mapping (TSM) and deep shadow mapping (DSM), as well as against the reference images rendered using the Embree [WWB*14] ray tracing kernels with 128 stratified rays per pixel. For the quality comparison, we use four samples per pixel for our algorithm and TSM. To accentuate the quality differences, we use fairly low resolution shadow maps of 512^2 pixels. An input tile size of 4×4 pixels was used for our algorithm, with the parameter settings $f = 12$ and $o = 8$. We implemented DSM by sampling the scene with 128 samples per pixel (spp) from the light source. This way, we can build a high quality visibility function for each shadow map pixel. Lokovic and Veach [LV00] suggested using a lower sampling rate and computing the visibility function by filtering over a larger footprint in the shadow map. However, the size of the filter is not well described, and therefore we chose to use a high quality (but inefficient) DSM implementation, which illustrates the algorithm's asymptotic behavior at high sampling rates.

The results of our quality evaluation can be seen in Figures 11, 12 and 13. DSM has problems with self-shadowing due to the assumption of a static receiver. Furthermore, the shadows from the pillars become smeared in Figure 11. The same behavior can be seen in the depth motion example in Figure 12, where a moving object incorrectly causes self shadowing. The scene has two moving objects, where the orange object cast a shadow on the moving green object, and both objects shadow a static ground plane. As can be seen, the shadow on the moving receiver is washed out for DSM. The correlation between the receiver and shadow caster must be captured to faithfully recreate the shadow for these cases. In all scenes, our algorithm and TSM produce results similar the reference. However, at equal samples per pixel and shadow resolution, our algorithm has considerably less noise.

In addition to the stress test scenes, we use another set of test scenes, shown in Figure 14, which have substantially richer geometrical detail and much higher occlusion complexity. We focus on evaluating performance of our shadowing algorithm compared to TSM, since the other competing

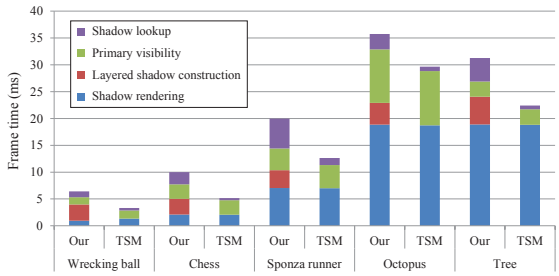

Figure 15: *The number of milliseconds spent in each of the algorithms steps for our approach and TSM. All results are measured on NVIDIA GTX 980.*

algorithms collapsed samples over time, which generated severe artifacts as shown in Figures 11, 12 and 13. All results were measured on an NVIDIA GTX 980. We include both an *equal input* and an *equal time* comparison. For *equal input*, we use 4 spp both for the primary visibility and the shadow map. The resolution of the shadow maps for both algorithms is 1024^2 for all test scenes. For the *equal time* comparison, when constructing the shadow map for TSM we used 8 spp and increased the resolution to fit within the same frame budget. For low complexity scenes, such as Wrecking ball (8k tris), TSM achieves slightly better quality at equal time. However, at higher polygon counts, the cost of stochastic rasterization is non-negligible, and our algorithm produces better quality shadows at equal time. This is visible in the Sponza runner scene and in Figure 1. The Tree scene has a moving light source, which makes it difficult to get an accurate result with TSM, although having more time slices in the shadow map alleviates the problem. The image resolution is 1280×720 for all scenes.

Figure 15 shows timing for the different steps in our algorithm and TSM. Although we store motion vectors when rendering from the light (visibility sampling), the cost for that step is very similar for both algorithms. Our algorithm then converts the incoming depth and motion buffers to its layered shadow map representation, which takes 3 – 5 ms. The cost of stochastic rasterization from the camera is roughly the same for both algorithms. However, the shadow lookup and shading pass are more expensive with our algorithm as we perform filtering in this step.

Finally, in the accompanying video, we show the temporal behavior of our algorithm.

7. Conclusions

We have presented a novel time-dependent shadow mapping algorithm, which supports high-quality filtering and accurately handles time dependencies between shadow casters and receivers. Our algorithm has real-time performance for reasonably complex scenes and scales with the number of samples, rather than geometrical complexity. This can be seen in that our best results were obtained with the most complex scenes.

For future work, we note that the weakness of our algorithm is when the depth complexity is too high or when there is large local motion variation within a small depth range. The clustering algorithm is crucial for both performance and quality, and we would like to explore clustering using additional attributes, apart from depth, such as the direction and magnitude of the motion vector. However, this is a non-trivial extension, which requires a solution to intra-layer visibility if layers have overlapping depth ranges.

Acknowledgements

We thank the Advanced Rendering Technology (ART) team at Intel. We also thank David Blythe and Chuck Lingle for supporting this research.

References

[AHAM11] ANDERSSON M., HASSELGREN J., AKENINE-MÖLLER T.: Depth Buffer Compression for Stochastic Motion Blur Rasterization. In *High Performance Graphics* (2011), pp. 127–134. 4

[AMB*07] ANNEN T., MERTENS T., BEKAERT P., SEIDEL H.-P., KAUTZ J.: Convolution Shadow Maps. In *Proceedings of EGSR* (2007), pp. 51–60. 3

[AMMH07] AKENINE-MÖLLER T., MUNKBERG J., HASSELGREN J.: Stochastic Rasterization using Time-Continuous Triangles. In *Graphics Hardware* (2007), pp. 7–16. 1, 2, 3, 4, 6

[AMS*08] ANNEN T., MERTENS T., SEIDEL H.-P., FLERACKERS E., KAUTZ J.: Exponential Shadow Maps. In *Proceedings of Graphics Interface* (2008), pp. 155–161. 3

[DHS*05] DURAND F., HOLZSCHUCH N., SOLER C., CHAN E., SILLION F. X.: A Frequency Analysis of Light Transport. *ACM Transactions on Graphics, 24*, 3 (2005), 1115–1126. 3

[DL06] DONNELLY W., LAURITZEN A.: Variance Shadow Maps. In *Symposium on Interactive 3D Graphics and Games* (2006), pp. 161–165. 3, 5, 6

[EDR11] EGAN K., DURAND F., RAMAMOORTHI R.: Practical Filtering for Efficient Ray-Traced Directional Occlusion. *ACM Transactions on Graphics, 30*, 6 (2011), 180:1–180:10. 3

[EHDR11] EGAN K., HECHT F., DURAND F., RAMAMOORTHI R.: Frequency Analysis and Sheared Filtering for Shadow Light Fields of Complex Occluders. *ACM Transactions on Graphics, 30*, 2 (2011), 9:1–9:13. 3

[ESAW11] EISEMANN E., SCHWARZ M., ASSARSSON U., WIMMER M.: *Real-Time Shadows*. AK Peters Ltd./CRC Press, 2011. 1, 2

[ETH*09] EGAN K., TSENG Y.-T., HOLZSCHUCH N., DURAND F., RAMAMOORTHI R.: Frequency Analysis and Sheared Reconstruction for Rendering Motion Blur. *ACM Transactions on Graphics, 28*, 3 (2009), 93:1–93:13. 3

[Fer05] FERNANDO R.: Percentage Closer Soft Shadows. In *ACM SIGGRAPH 2005 Sketches* (2005), p. 35. 3

[GMN14] GUERTIN J.-P., MCGUIRE M., NOWROUZEZAHRAI D.: A fast and stable feature-aware motion blur filter. In *HPG* (2014), ACM/Eurographics, p. 10. 7

[HA90] HAEBERLI P., AKELEY K.: The Accumulation Buffer: Hardware Support for High-Quality Rendering. In *Computer Graphics (Proceedings of SIGGRAPH 90)* (1990), vol. 24, ACM, pp. 309–318. 1

[HMV14] HASSELGREN J., MUNKBERG J., VAIDYANATHAN K.: *Practical Layered Reconstruction for Defocus and Motion Blur*. Tech. rep., Advanced Rendering Technology, Intel, December 2014. 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. 3

[LK11] LAINE S., KARRAS T.: *Efficient Triangle Coverage Tests for Stochastic Rasterization*. Tech. Rep. NVR-2011-003, NVIDIA Corporation, Sep 2011. 6

[LM08] LAURITZEN A., MCCOOL M.: Layered Variance Shadow Maps. In *Graphics Interface* (2008), pp. 139–146. 3

[LV00] LOKOVIC T., VEACH E.: Deep Shadow Maps. In *Proceedings of SIGGRAPH 2000* (2000), ACM, pp. 385–392. 1, 2, 8

[ME11] MCGUIRE M., ENDERTON E.: Colored Stochastic Shadow Maps. In *Symposium on Interactive 3D Graphics and Games* (2011), ACM, pp. 89–96. 2

[MESL10] MCGUIRE M., ENDERTON E., SHIRLEY P., LUEBKE D.: Real-Time Stochastic Rasterization on Conventional GPU Architectures. In *High Performance Graphics* (2010), pp. 173–182. 6

[MHBO12] MCGUIRE M., HENNESSY P., BUKOWSKI M., OSMAN B.: A Reconstruction Filter for Plausible Motion Blur. In *Symposium on Interactive 3D Graphics and Games* (2012), pp. 135–142. 3

[MVH*14] MUNKBERG J., VAIDYANATHAN K., HASSELGREN J., CLARBERG P., AKENINE-MÖLLER T.: Layered Light Field Reconstruction for Defocus and Motion Blur. *Computer Graphics Forum, 33*, 4 (2014), 81–92. 3, 4, 5, 7

[MWR12] MEHTA S., WANG B., RAMAMOORTHI R.: Axis-Aligned Filtering for Interactive Sampled Soft Shadows. *ACM Transactions on Graphics, 31*, 6 (2012), 163:1–163:10. 3

[MWRD13] MEHTA S. U., WANG B., RAMAMOORTHI R., DURAND F.: Axis-Aligned Filtering for Interactive Physically-based Diffuse Indirect Lighting. *ACM Transactions on Graphics, 32*, 4 (2013), 96:1–96:12. 3

[MYRD14] MEHTA S. U., YAO J., RAMAMOORTHI R., DURAND F.: Factored Axis-aligned Filtering for Rendering Multiple Distribution Effects. *ACM Transactions on Graphics, 33*, 4 (2014), 57:1–57:12. 3

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

[RSC87] REEVES W. T., SALESIN D. H., COOK R. L.: Rendering Antialiased Shadows with Depth Maps. In *Computer Graphics (Proceedings of SIGGRAPH 87)* (1987), vol. 21, ACM, pp. 283–291. 3

[VMCS14] VAIDYANATHAN K., MUNKBERG J., CLARBERG P., SALVI M.: Layered Light Field Reconstruction for Defocus Blur. *ACM Transactions on Graphics (to appear)* (2014). 6

[Wil78] WILLIAMS L.: Casting curved shadows on curved surfaces. In *Computer Graphics (Proceedings of SIGGRAPH 78)* (1978), vol. 12, pp. 270–274. 1

[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. 8

[YDF*10] YANG B., DONG Z., FENG J., SEIDEL H.-P., KAUTZ J.: Variance Soft Shadow Mapping. *Computer Graphics Forum, 29*, 7 (2010), 2127–2134. 3

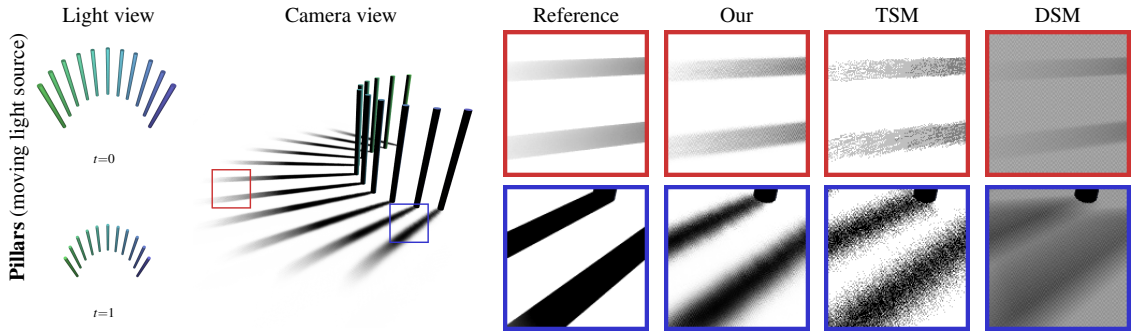

Figure 11: *Shadows are produced by a light source, moving upwards, away from the pillars. This scene is difficult since points on the receiver plane that remain static viewed from the camera travel long distances in light space. Due to perspective, the sample paths through light space are non-linear. Some over-blurring is apparent, but our linear motion approximation works reasonably well.* TSM *also exhibits some additional blurring. When a shadow map lookup is performed, the camera sample is transformed to light space at the camera time t_c to obtain the lookup coordinates. The time t_s for the sample at this location is (almost) always different from t_c. During this time discrepancy, the light source may have moved arbitrarily, potentially resulting in a poor match between the camera and shadow map samples in xyt-space.* DSM *does not account for moving light sources either, and get severe self-shadowing artifacts.*

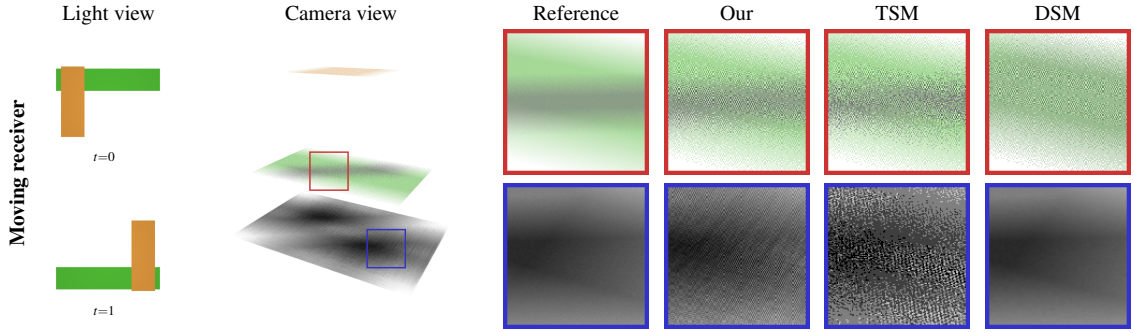

Figure 12: *Two rapidly moving quads at different depths with motion directions orthogonal to each other. Both our algorithm and* TSM *are able to capture the diagonal shadow stripe that appears on the green, moving receiver, whereas* DSM *does not.*

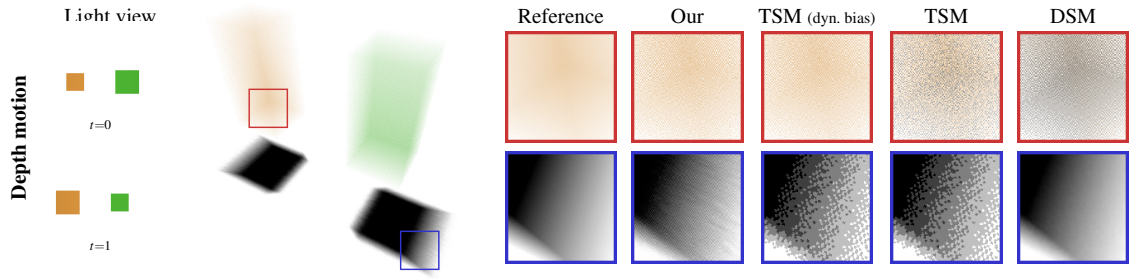

Figure 13: *Two quads with perspective motion in light space, one moving towards and the other moving away from the light source. Our algorithm performs well, even though there is a high degree of motion along the z-axis, which violates our assumption of linear sample paths. With our dynamic bias from Section 5.1,* TSM *does not exhibit self-shadowing artifacts. The bias in* DSM *cannot be fixed with an increased bias value, since the movement of the quads is larger than the gap between them and the ground plane.*

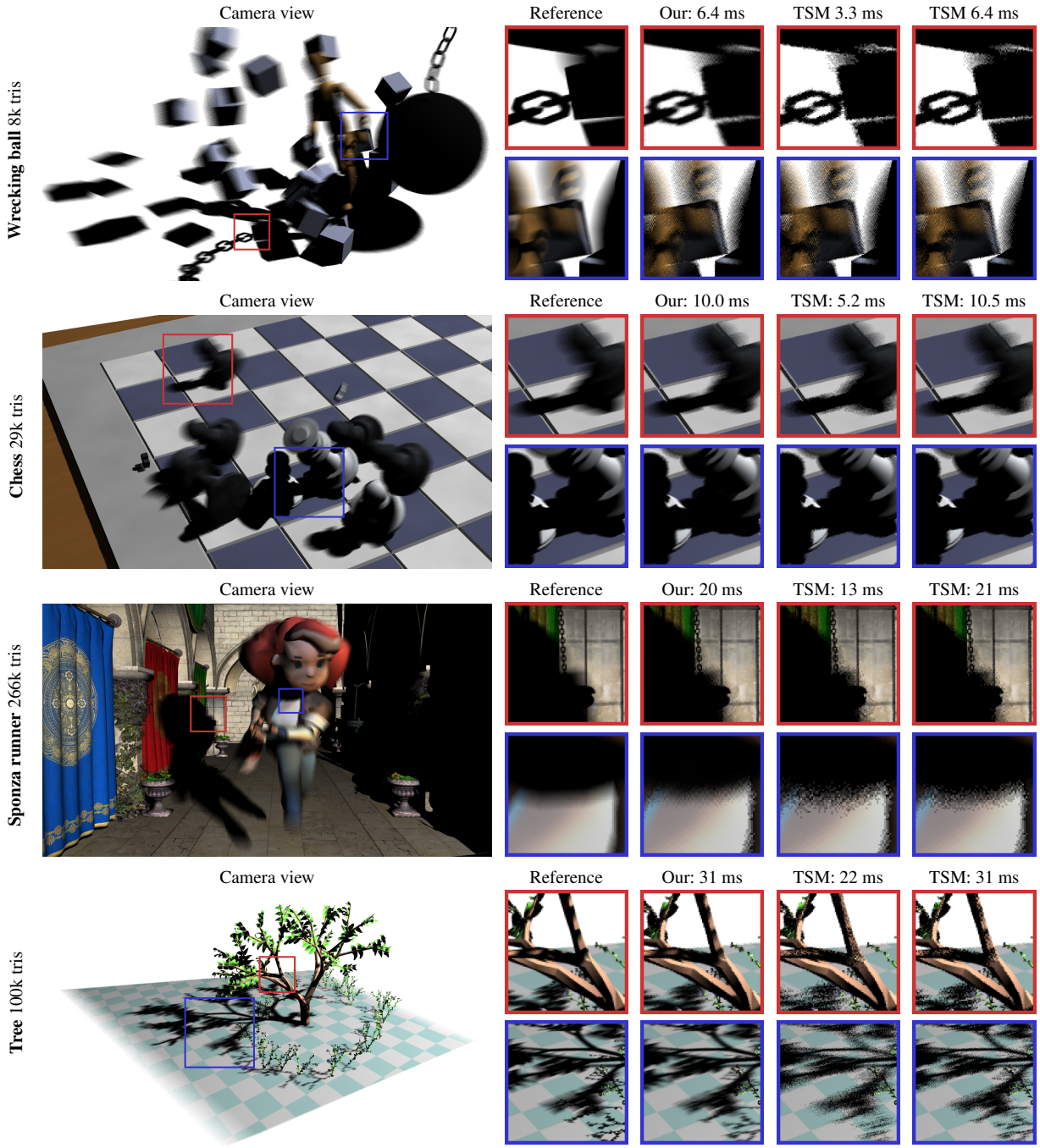

Figure 14: *Our algorithm has significantly less noise compared to time-dependent shadow maps (TSM) and handles complex occlusion. The animated wooden doll model in the wrecking ball scene is from the Utah 3D Animation Repository. The chess scene was created by Rasmus Barringer. The Sponza model was made by Frank Meinl at Crytek, and the skinned runner model by Björn Sörensen. The tree scene with a moving light source was created using Autodesk Maya.*