To appear in ACM TOG 30(4) (Proc. ACM SIGGRAPH 2011)

Temporal Light Field Reconstruction for Rendering Distribution Effects

Jaakko Lehtinen, NVIDIA Research
Timo Aila, NVIDIA Research
Jiawen Chen, MIT CSAIL
Samuli Laine, NVIDIA Research
Frédo Durand, MIT CSAIL

Figure 1: *A scene with complex occlusion rendered with depth of field. Left: Images rendered by PBRT [Pharr and Humphreys 2010] using 16 and 256 low-discrepancy samples per pixel (spp) and traditional axis-aligned filtering. Right: Image reconstructed by our algorithm in 10 seconds from the same 16 samples per pixel. We obtain defocus quality similar to the 256 spp result in approximately 1/16th of the time.*

Abstract

Traditionally, effects that require evaluating multidimensional integrals for each pixel, such as motion blur, depth of field, and soft shadows, suffer from noise due to the variance of the high-dimensional integrand. In this paper, we describe a general reconstruction technique that exploits the anisotropy in the temporal light field and permits efficient reuse of samples between pixels, multiplying the effective sampling rate by a large factor. We show that our technique can be applied in situations that are challenging or impossible for previous anisotropic reconstruction methods, and that it can yield good results with very sparse inputs. We demonstrate our method for simultaneous motion blur, depth of field, and soft shadows.

Keywords: depth of field, motion blur, soft shadows, light field, reconstruction

1 Introduction

A number of advanced rendering techniques require the reconstruction and integration of radiance from samples. Recent analysis has emphasized the anisotropic and bandlimited nature of the radiance signal, leading to frequency-based adaptive sampling for glossy highlights [Durand et al. 2005], depth of field [Soler et al. 2009], motion blur [Egan et al. 2009], and soft shadows [Egan et al. 2011]. These techniques focus on sampling, and while they provide dramatic reductions in sampling rate, they rely on fairly simple reconstruction that suffers from a number of limitations. First, because they use linear reconstruction kernels and a simple model of local spectrum, they fail near object boundaries, and need to resort to brute-force sampling and reconstruction there. While this would not be a problem for pinhole images of static scenes, it becomes significant for motion blur and depth of field, where the blur causes boundaries to affect a large fraction of pixels, 70% in the case of Figure 1. Other techniques [Hachisuka et al. 2008] rely on the sampled radiance itself to determine anisotropy, which requires noise-free samples, and a potentially high sampling rate for high-frequency signals such as textured surfaces, defeating the purpose of adaptive sampling.

We concentrate on reconstruction, and seek to improve the images obtained from a relatively sparse stochastic sampling of the high-dimensional domain (screen, lens, time, light source, etc.) of the radiance function. This complements adaptive techniques that drive the sampling process by predictions derived from analyzing light transport. Our algorithm can be applied as a black box, as long as we have collected auxiliary information (motion vectors, depth) about the samples. We demonstrate high-quality rendering results in situations where linear reconstruction or contrast-driven adaptive sampling are ineffective, while using a small fraction of the time required for rendering equal-quality results using traditional methods. We operate strictly in the primal light field domain.

This paper makes the following contributions:

- A non-linear temporal light field reconstruction algorithm that is applicable in the presence of complex occlusion effects,
- A method for determining the visibility consistency of a set of light field samples based on visibility events,
- A method to resolve visibility without explicit surface reconstruction, with support for occlusion boundaries, and
- A hierarchical query structure for efficient pruning of the input light field samples.

We apply our algorithm to simultaneous depth of field, motion blur, and shadows cast by area light sources.

2 Related Work

Distribution ray tracing [Cook et al. 1984] is widely used for simulating blurry phenomena such as motion blur, depth of field, and glossy reflection in offline rendering. Averaging the stochastic samples by a pixel filter corresponds to using axis-aligned nearest neighbor reconstruction in the light field prior to integrating over time/lens/light/incident direction. Even with high-quality sampling patterns, a large number of potentially very expensive samples is often required for reducing noise to an acceptable level, especially with defocus and soft shadows from area lights. This is undesirable due to the high cost of both shading and visibility sampling, even when using techniques that reuse shaded values [Cook et al. 1987; Ragan-Kelley et al. 2011]. There is a clear need for methods that maximize the image quality obtainable from a given set of samples.

Light transport analysis and sheared reconstruction Egan et al. [2009] present a frequency analysis of motion blur. They drive sampling based on predicted bandwidth across time and space, and propose a sheared reconstruction algorithm that extends filter support along the motion. This drastically reduces the number samples required in homogeneously moving regions, because information is shared between multiple pixels. The benefit diminishes in regions that exhibit inhomogeneous movement, for example due to object or occlusion boundaries. Similar arguments apply to sheared reconstruction for soft shadows [Egan et al. 2011]. In contrast to both approaches, we handle complex occlusion effects, and support simultaneous motion, defocus and soft shadows.

Soler et al. [2009] analyze defocus in frequency space, and propose an algorithm for adaptive sampling for depth of field. They vary both the image-space sampling and the lens sampling, but only perform reconstruction in the 2D image. This prevents them from sharing lens samples across the image, and presents challenges at object boundaries.

Contrast-based adaptive sampling Multi-dimensional adaptive sampling (MDAS) [Hachisuka et al. 2008] takes radiance samples in the space of the high-dimensional light transport integrand, and reconstructs the integrand using an anisotropic nearest neighbor filter. Specifically, MDAS strives to concentrate samples around significant contrast thresholds, and tailor the filter to anisotropy measured directly from the samples. Given noise-free samples, this can work well for defocus and motion blur. However, a complex integrand with high variance due to noise, complex visibility boundaries, or even texture, causes the sampler to essentially revert to uniform sampling. Furthermore, the algorithm is sensitive to the choice of input parameters, e.g., the relative scales between the dimensions of the integrand. In addition, MDAS quickly becomes infeasible in higher dimensions due to its reliance on kD-trees, making application to simultaneous motion, defocus and area lights challenging.

Adaptive wavelet rendering [Overbeck et al. 2009] extends frequency ray tracing [Bolin and Meyer 1995] to handle integrals computed at each pixel. It adapts to regions where either the image or the integrand is smooth. However, it cannot adapt as well to the anisotropic case where radiance is smooth along a direction other than an axis of the wavelet basis, such as the motion blur of a high-frequency texture.

Reprojection techniques Chen and Williams [1993] describe a method for view interpolation from pinhole cameras, and apply it to motion blur and soft shadows. This approach has been recently extended and parallelized to run quickly on GPUs [Yu et al. 2010]. Both of these approaches show good results in simple scenarios, but the reconstruction heuristics used for resolving visibility and avoiding holes are geared towards simplicity and speed instead of high quality. In a similar vein, some algorithms reuse samples across frames with the aim of speeding up illumination computations, e.g. [Walter et al. 1999; Nehab et al. 2007]. In contrast, we perform reprojection within the high-dimensional integration domain, not across frames, making use of stochastic samples. In addition, we explicitly deal with visibility based on the similarity of the apparent trajectories of the samples in the temporal light field.

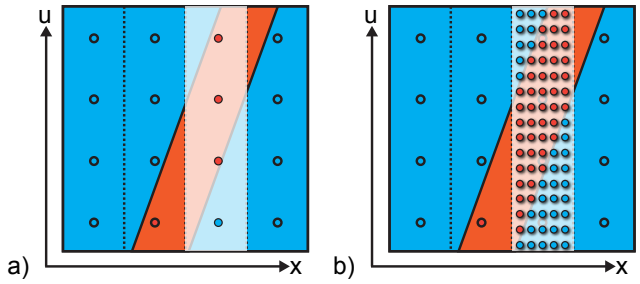

Figure 2: *a) A slice through an XU light field showing an out-of-focus foreground object (orange) on an out-of-focus background (blue). Vertical dashed lines denote pixel boundaries, and disks denote* input samples *over the screen and lens. A traditional distribution ray tracer generates an image by averaging the input samples in a pixel. b) The light field, reconstructed at a large number of* reconstruction locations *by our algorithm. Thanks to the increased resolution, the resulting image is of much higher quality. A regular sampling is shown for clarity only; our algorithm operates on stochastic samples.*

Lee et al. [2010] blur the distinction between sampling and reconstruction. They accelerate defocus effects by first rasterizing the scene into a layered depth image (LDI), and then performing fast ray tracing using the LDI as a scene representation. The initial sampling is done using a pinhole camera, and thus some surfaces may not be represented well. Furthermore, extending the method to motion blur and other distributed effects like area shadows seems difficult.

Fast 2D approximations Some early motion blur algorithms are based on compositing 2D images that have been blurred in the direction of motion in layers that correspond to different objects [Potmesil and Chakravarty 1983; Max and Lerner 1985]. This poses problems for complex scenes. Recent realtime algorithms use image-based blurring driven by the depth buffer. Nguyen [2007] presents a survey. While the techniques are fast and well adapted to GPU pipelines, they typically suffer from a number of noticeable artifacts due to insufficiently captured occlusion.

3 Algorithm

We seek to achieve high-quality distribution effects such as depth of field and motion blur from a sparse set of 5D (screen-lens-time) samples. For this, we present a new reconstruction algorithm that can upsample these sparse input samples using a non-linear process that leverages information about depth and motion, and that accurately resolves visibility. See Figure 2. Final pixel values are obtained by summation over the reconstructed samples, similar to a normal distribution ray tracer, except that reconstruction is much faster than computing radiance using ray tracing.

Our reconstruction builds on the common observation that the high-dimensional radiance field exhibits strong anisotropy [Chai et al. 2000; Durand et al. 2005; Egan et al. 2009], with slope aligned

with the inverse depth [Chai et al. 2000], inverse velocity [Egan et al. 2009], or inverse blocker depth [Egan et al. 2011]. This is particularly true for Lambertian objects, but is also largely the case for glossy highlights due to a rough BRDF. We exploit this anisotropy to perform reconstruction that follows the direction of the signal. The method can be extended to leverage bandwidth and adapt the range of influence of a sample to handle cases such as arbitrary BRDFs, which we demonstrate with a proof-of-concept implementation.

Our reconstruction provides higher quality than previous work because it leverages information about depth and motion of each input sample in two critical ways. First, we use per-sample depth and motion information to accurately reproject the individual samples. This contrasts with previous sheared reconstructions [Hachisuka et al. 2008; Egan et al. 2009] that consider only one direction of anisotropy per reconstruction and have limited ability to handle cases where the reconstruction kernel overlaps samples with different directions, in particular when multiple objects are present (Figure 3). Second, we use depth information together with new algorithms that determine occlusion between samples to accurately treat visibility. This further enables our method to handle reconstruction around object boundaries.

We describe our algorithm in 5D to handle simultaneous motion and defocus blur. We extend the algorithm to soft shadows in Section 5.

Algorithm overview The key challenge is to perform high-quality and efficient reconstruction at a given 5D location with coordinates $\hat{x}\hat{y}$ in the image, $\hat{u}\hat{v}$ on the lens, and time $\hat{t}$. In order to exploit the anisotropy and reason about occlusion, we reduce the problem to a 2D $\hat{x}\hat{y}$ neighborhood in image space by reprojecting the samples to the lens-time coordinates $\hat{u}\hat{v}\hat{t}$ along their individual trajectories. This provides us with a scattered set of point samples in 2D. In the simplest case, the samples' visibility is consistent, i.e., they form a single apparent surface, and they can all be filtered together by averaging their radiance. However, the strength of our technique is its ability to handle occlusion configurations as well. For this, we need to solve two critical problems. First, we need a criterion to detect occlusion in the light field. This is especially difficult because we deal with moving objects. Second, if occlusion occurs, we need to determine which samples correspond to the apparent surface visible at the the reconstruction location.

Finally, only a small number of input samples actually contribute to a given reconstruction location. It is, however, difficult to determine which samples need to be considered because it depends on each sample's individual trajectory (Figure 3). This is why we introduce a hierarchical data structure to accelerate sample retrieval.

The input to our algorithm consists of a set of *input samples*

$$s_i = \{(x_i,\ y_i,\ u_i,\ v_i,\ t_i) \mapsto (z_i,\ \mathbf{v}_i,\ L_i)\}$$

from the temporal light field, where xy are screen coordinates, uv are lens coordinates, t is the time coordinate, z is the depth, $\mathbf{v}$ is the 3D motion vector (including depth change), L_i is the radiance associated with the input sample. We make the common approximation of linear motion in world space, which results in non-uniform screen-space motion due to perspective[1]. Note that even though the motion of a single sample is linear, an object as a whole may undergo non-rigid deformation. For defocus, we employ the standard two-plane thin-lens model [Pharr and Humphreys 2010].

[1] In principle, any motion model could be used, as long as its parameters can be computed and exported at the time of input sampling. In practice, we believe curvilinear motion is best approximated by linear segments.

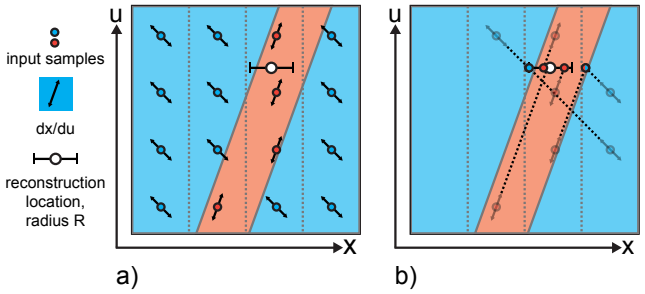

Figure 3: *Reprojecting the input samples. a) The input samples and their dx/du. The reconstruction location $(\hat{x}, \hat{u})$ is denoted by the white disk, and the radius R by the line segment. b) Reprojecting the input samples to the reconstruction location's $\hat{u}$ coordinate.*

To generate the output image, we explicitly integrate over the screen, time, lens dimensions (and light source when handling soft shadows) by Monte Carlo integration. At each Monte Carlo sample, which we call *reconstruction locations*, we reconstruct the temporal light field from the input samples. We typically take $N = 128$ reconstruction locations per pixel, and find the pixel radiance by simply integrating the reconstruction results with pixel filter $P(x, y)$:

$$L(\text{Pixel}) = \frac{1}{N}\sum_{j=1}^{N} \textsc{Reconstruct}(\hat{x}_j, \hat{y}_j, \hat{u}_j, \hat{v}_j, \hat{t}_j)\ P(\hat{x}_j, \hat{y}_j), \tag{1}$$

where $\hat{u}_j, \hat{v}_j, \hat{t}_j$ are samples drawn from the lens $\times$ time cube, and $\hat{x}_j, \hat{y}_j$ are samples drawn from the support of the pixel filter. As reconstruction and integration are combined, we do not need to store the upsampled temporal light field at any point.

Reconstruction algorithm The function RECONSTRUCT processes each 5D reconstruction location $(\hat{x}_j, \hat{y}_j, \hat{u}_j, \hat{v}_j, \hat{t}_j)$ as follows.

1. We first reproject the input samples' screen coordinates to the $(\hat{u}_j, \hat{v}_j, \hat{t}_j)$ coordinates of reconstruction location (Section 3.1), and discard samples that are too far away in xy (Section 3.3). The reprojected samples are efficiently queried from a hierarchy (Section 4), and are returned from the hierarchy as clusters of samples.
2. We proceed to group the returned clusters into apparent surfaces using the algorithm described in Section 3.2. This typically results in 1-2 surfaces per reconstruction location.
3. If more than one apparent surface is found, we sort the surfaces front-to-back, and proceed to determine which one covers the reconstruction location (Section 3.3).
4. We compute the output color by filtering the samples that belong to the covering surface using a circular tent filter. Note that the reconstruction results are still subsequently filtered with the pixel filter $P(x, y)$ according to Equation 1.

3.1 Reprojection

Based on our motion and lens models, the depth and motion vector of an input sample determine how it behaves in the temporal light field under defocus and motion. Concretely, we solve for how the input sample's screen position changes as a function of the $(\hat{u}, \hat{v}, \hat{t})$ coordinates. Figure 3 illustrates this for defocus.

The reprojected screen position (x_i', y_i') for sample s_i as a function of $(\hat{u}, \hat{v}, \hat{t})$ is obtained by first computing the depth according to the

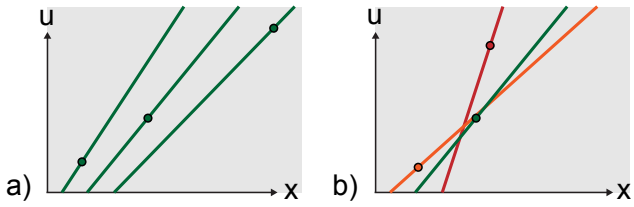

Figure 4: *a) samples whose trajectories do not intersect have no mutual visibility events, and can be treated as a single surface in the reconstruction. b) Visibility changes between surfaces leads to crossings between the trajectories of the samples.*

z component of the motion vector as

$$z_i'(\hat{t}) = z_i + (\hat{t} - t_i)\mathbf{v}_i^z, \tag{2}$$

and then computing the new screen position as

$$\begin{aligned} x_i'(\hat{u},\hat{t}) &= \frac{\big[x_i - u_i C(z_i)\big] z_i + (\hat{t} - t_i)\mathbf{v}_i^x}{z_i'(\hat{t})} + \hat{u}C(z_i') \\ y_i'(\hat{v},\hat{t}) &= \frac{\big[y_i - v_i C(z_i)\big] z_i + (\hat{t} - t_i)\mathbf{v}_i^y}{z_i'(\hat{t})} + \hat{v}C(z_i'). \end{aligned} \tag{3}$$

Conceptually, Equations 2-3 reproject the sample from its original screen location (x_i, y_i), computed at (u_i, v_i, t_i), to the center of the lens, backproject the point to 3D, then move it along the 3D motion vector to the new time $\hat{t}$, project back to screen coordinates, and finally to the new lens position $(\hat{u}, \hat{v})$ according to the circle of confusion at the new depth. As expected, setting $(\hat{u}, \hat{v}, \hat{t}) = (u_i, v_i, t_i)$ leaves the screen position unchanged. The signed circle of confusion $C(z_i) = \partial x/\partial u = \partial y/\partial v$ is computed by

$$C(z) = \frac{C_1}{z} + C_2, \tag{4}$$

where C_1, C_2 are per-frame constants determined from standard thin-lens geometry [Pharr and Humphreys 2010].

The reprojection of a sample through the full lens-time cube might ignore view-dependent effects for non-Lambertian objects. This corresponds to the shade-once assumption often used in production rendering, e.g. [Cook et al. 1987], and most results in this paper are obtained this way. However, our method can be extended to use BRDF bandwidth estimates [Ramamoorthi and Hanrahan 2004; Durand et al. 2005] for determining a range of validity so that such a sample may only contribute to reconstruction locations that are close enough to the reconstruction location along the light field trajectory. We demonstrate a prototype implementation.

3.2 Detecting Visibility Consistency

The ability to detect when a set of reprojected samples should be filtered together is crucial to our algorithm. We could use simple heuristics such as surface IDs or thresholds on the difference between depths and motion vectors of the input samples, but such approaches are notoriously brittle, as confirmed by our early experiments. Instead, we derive a formal criterion based on the crossing of the reprojection trajectories in the temporal light field. Our criterion allows any set of samples that have no detectable visibility changes in the vicinity of a reconstruction location — regardless of their originating primitive or object — to be filtered together.

We start with a simpler case, depth of field in a static scene, where we need to determine if two samples can occlude each other from different lens positions. Our key observation is that *the relative ordering of the screen positions of samples from a non-overlapping surface never changes under reprojections*. In other words, if sample s_1 lies to the right and up from sample s_2, as seen from one point on the lens, this must hold for any other lens coordinates as well[2]. In contrast, visibility changes due to defocus always lead to crossings that can be detected as changing xy orderings between points as $(\hat{u}, \hat{v}, \hat{t})$ change (cf. Figure 4). This holds because our original input sampling only contains points that are visible from some part of the 5D domain, i.e., there are no samples that are always behind another surface. We define the consistency test for samples s_1 and s_2 around the lens coordinates $(\hat{u}, \hat{v})$ as

$$\begin{aligned} &\textsc{SameSurface}(s_1, s_2, \hat{u}, \hat{v}) \stackrel{\text{def}}{=} \\ &\quad x_1'(\hat{u} \pm \delta, \hat{v} \pm \delta) - x_2'(\hat{u} \pm \delta, \hat{v} \pm \delta) \\ &\quad y_1'(\hat{u} \pm \delta, \hat{v} \pm \delta) - y_2'(\hat{u} \pm \delta, \hat{v} \pm \delta) \\ &\quad \text{have consistent sign for all the 4 choices of } (\pm, \pm) \end{aligned} \tag{5}$$

where $\delta = \sqrt{1/N}$, where N is the number of reconstruction locations per pixel, approximates the size of the subdomain that corresponds to one reconstruction location. The square root accounts for the fact that the lens domain is 2D.

We extend the above test to handle motion using the same rationale: screen positions of samples that can be treated as a single apparent surface do not cross on the screen, when the time coordinate is changed. Furthermore, the criterion is easily extended to groups of samples by comparing the relative xy orderings of the vertices of their bounding rectangles. In practice, we allow the trajectories to cross for at most 0.1 pixels.

3.3 Reconstruction with Complex Visibility

Reconstruction from samples that originate from multiple apparent surfaces is hard, because we need to determine which surface is visible at the reconstruction location. The challenge is to distinguish between small holes in the geometry and apparent holes caused by stochastic sampling. Our solution is to locally triangulate the reprojected samples, aided by efficient, conservative early-out tests.

Based on the 5D input sampling pattern, we precompute a radius R of the largest empty circle we expect to see on the $\hat{x}\hat{y}$ plane after reprojection. Consequently, any hole smaller than R should be filled, because it is beyond the resolution of our input sampling. However, surface boundaries need special treatment, because extending their coverage by R leads to visible bloating (Figure 5).

Determining R The *dispersion* of a point set is defined as the radius of the largest circle that does not contain any of the points. For a sampling pattern, this provides a fundamental limit below which we cannot distinguish between apparent and actual holes. We set the radius R to be equal to the dispersion limit measured from the input sampling pattern. However, the dispersion measured in xy changes under reprojections. E.g., when we take a high-quality $xyuv$ sampling pattern that has low dispersion (such as one obtained from a Sobol sequence) and reproject according to Equations 2-3, the point set becomes much less uniformly distributed in xy, i.e., its dispersion increases[3]. When measuring R, we take

[2]The only exception is when the surface changes apparent orientation from backfacing to frontfacing for some lens coordinates. Our algorithm treats samples that originate from different sides as separate surfaces.

[3]Some intuition as to why this happens is offered by the fact that a good sampling pattern has little energy in the center of the Fourier domain, except for the peak at zero. Defocus and motion are shears that effectively

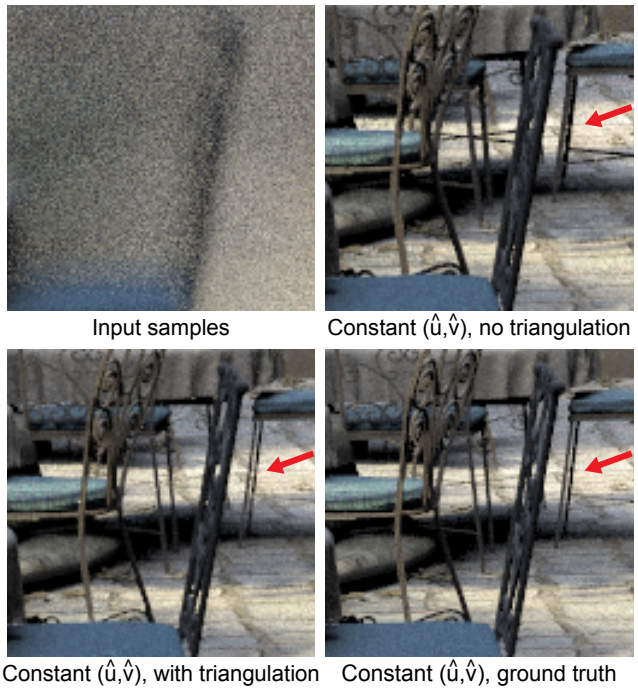

Figure 5: *A constant-$\hat{u}\hat{v}$ slice of the $\hat{x}\hat{y}\hat{u}\hat{v}$ light field shows the importance of triangulation. Top-left: Axis-aligned filtering of the 16 input samples per pixel. Top-right: Surface boundaries bloat without triangulation. Bottom-left: Triangulation avoids the bloat. Bottom-right: Ground truth from PBRT using a pinhole camera.*

the maximum dispersion under a large number of random reprojections. The measured values for R are approximately $2/\sqrt{N}$ for our Sobol sequences, and other high-quality sampling patterns seem to give similar values. Note that the dispersion limit is a property of the sampling pattern and thus independent of the scene.

Determining visibility We say that a reconstruction location is *covered* if it is possible to find three reprojected input samples that form a triangle that 1) covers the reconstruction location, and 2) fits inside a circle of radius R. Let us call such triangles *valid*. By elementary geometry, the edges of a valid triangle cannot be longer than $2R$ because otherwise it cannot possibly fit inside a circle of radius R. Since the reconstruction location can be close to one its vertices, we may consequently need input samples that reproject as far as $2R$ of the reconstruction location to find all valid triangles.

We explain the algorithm using two surfaces; the extension to any number of surfaces is straightforward. Figure 6 illustrates the three possible scenarios between two surfaces:

1. Samples belong to one surface.
2. Samples belong to two surfaces, and the nearest surface extends in all directions from the reconstruction location.
3. Samples belong to two surfaces, and we are at a boundary of the nearest surface.

If samples from only one surface are found within R of the reconstruction location, we have trivial coverage. Most occurrences of the second case can be detected quickly by classifying the input

contract the empty area at the center, because the subsequent xy projection corresponds to a slice through the zero frequency axes Ω_u and Ω_v (Fourier slice theorem). This is visible in the primal domain as larger empty areas.

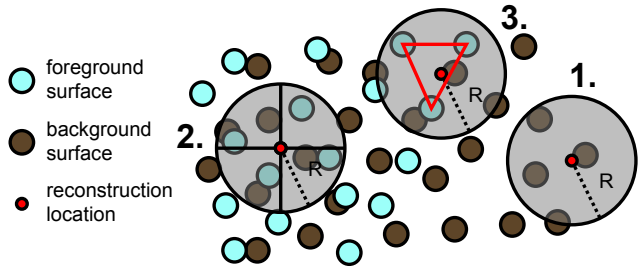

Figure 6: *Determining visibility. 1) Samples from only one surface fall within radius R of the reconstruction location. All dark brown samples within R are linearly filtered. 2) Samples come from two surfaces, but samples from the foreground surface fall in all four quadrants around the reconstruction location. Hence, a covering quadrilateral necessarily exists. The xy filter is applied to the light green samples only. 3) Border. The actual covering surface is found by locally triangulating the samples.*

samples into 90-degree quadrants relative to the reconstruction location's $(\hat{x}, \hat{y})$. If all quadrants contain a sample from the nearest surface within R, we know without further work that we can form a valid triangle that covers the reconstruction location. Cases 1 and 2 handle over 90% of the reconstruction locations in our test scenes.

In the third case, we search for *any* three input samples that span a valid triangle. This is implemented as a $O(n^3)$ loop over the input samples that belong to each surface. Despite the high algorithmic complexity, the sample counts were low enough in our tests that the triangulation was not a bottleneck. Should the need ever arise, asymptotically faster $O(N \log N)$ triangulation algorithms could be used instead. Section 6.2 discusses the timing breakdown.

If the reconstruction location is covered, we filter the radiance from all samples of the covering surface using a tent filter of radius R centered at the reconstruction location.

Sometimes we have only one or two samples from a surface, and therefore cannot determine the coverage using triangulation. In these cases we merge the surface to the next one, effectively blending the two surfaces together. In our scenes, this happens in $0.3 - 0.5\%$ of the reconstruction locations.

4 Acceleration

The reconstruction algorithm needs to quickly find the input samples that reproject to the vicinity of reconstruction location's $(\hat{x}, \hat{y})$, given $(\hat{u}, \hat{v}, \hat{t})$. The primary challenge is that an input sample can move a considerable distance on the screen during the shutter interval, or as a function of lens coordinates in out-of-focus areas[4]. Our solution is a 5D hierarchical acceleration structure. Glassner [1988] and other authors have proposed acceleration structures where the nodes' spatial extents are parameterized with time in order to efficiently ray trace animated scenes. We employ the same general principle and organize the input samples into a bounding volume hierarchy (BVH), where the nodes' xy extents are parameterized using $\hat{u}$, $\hat{v}$ and $\hat{t}$. At query time, we use this parameterized bounding volume to test if the reconstruction location is inside the bounds.

Node bounds We compute four plane equations, $X_{\min}, X_{\max}$, $Y_{\min}, Y_{\max}$, for each node of the hierarchy. These conservatively

[4]However, the number of samples reprojecting to the vicinity of $(\hat{x}, \hat{y})$ from a single $(\hat{u}, \hat{v}, \hat{t})$ is relatively unaffected by the magnitude of motion and defocus. Variation is caused primarily by changes in visibility.

function ConstructSurfaces(x, y, u, v, t)
1: $N \leftarrow$ QueryTree(x, y, u, v, t)
2: $N \leftarrow$ Sort(N)
3: $i \leftarrow 0$
4: $surfaces \leftarrow$ empty list
5: $S \leftarrow$ empty list
6: **while** $i < N.length$ **and** SameSurface$(S, N[i])$ **do**
7: S.append$(N[i])$
8: $i \leftarrow i + 1$
9: **end while**
10: $surfaces$.append(S)
11: **if** $i < N.length$ **goto** 5
12: **return** $surfaces$

Figure 7: *Pseudocode for grouping nodes to surfaces for a reconstruction location (x, y, u, v, t). List N holds the leaf nodes returned by the tree. List S holds the current surface, and* SameSurface$(S, node)$ *tests node against all nodes in S.*

bound the xy coordinates of all samples contained in the node as a function of lens coordinates and time, so that we can quickly obtain a conservative screen bounding rectangle for a node for any $(\hat{u}, \hat{v}, \hat{t})$. Concretely, $X_{\text{min}} = (A, B, C)$ is constructed such that for all input samples in the node, their reprojected coordinates $x_i'(\hat{u}, \hat{v}, \hat{t})$ obey

$$x_i'(\hat{u}, \hat{v}, \hat{t}) \geq A\hat{u} + B\hat{t} + C \tag{6}$$

for any $(\hat{u}, \hat{v}, \hat{t})$, and analogously for the other planes, flipping comparisons as necessary[5]. The algorithm for constructing the planes is detailed in Appendix A.

If the reconstruction location's $(\hat{x}, \hat{y})$ lie within the specified radius of the node's extents, as evaluated by Equation 6, the node may contain samples that get reprojected within the radius of the reconstruction location. We then say that a reconstruction location *intersects* the node.

Query The task of a query from the acceleration structure is to find the input samples that reproject close to $(\hat{x}, \hat{y})$, group the reprojected samples into apparent surfaces, and order the surfaces front-to-back. The query starts from the BVH's root node, recursively proceeds to all intersected child nodes, and collects the intersected leaf nodes into a list. Once the leaf nodes are found, we group them into apparent surfaces, sort the surfaces, and reproject the individual samples contained in them. As a critical optimization, we guarantee during hierarchy construction that each leaf node contains samples from exactly one surface, as determined by SAMESURFACE (Equation 5). *This allows us to form the surfaces by grouping leaf nodes instead of individual samples*, greatly reducing the complexity of this stage. In practice, we first sort the leaf nodes according to the reprojected z' of the first sample in each leaf, and then walk the sorted list and group the adjacent nodes if they satisfy SAMESURFACE. Figure 7 provides pseudocode. Finally, the individual samples of the leaf nodes are reprojected and checked for inclusion within the radius.

Construction The construction follows a push-pull procedure, where we first create the leaf nodes using a top-down quadtree subdivision, and then construct a BVH using bottom-up grouping of nodes.

[5]Note that X_{min} and X_{max} are independent of v because defocus is separable between xu and yv. Similarly Y_{min} and Y_{max} are independent of u.

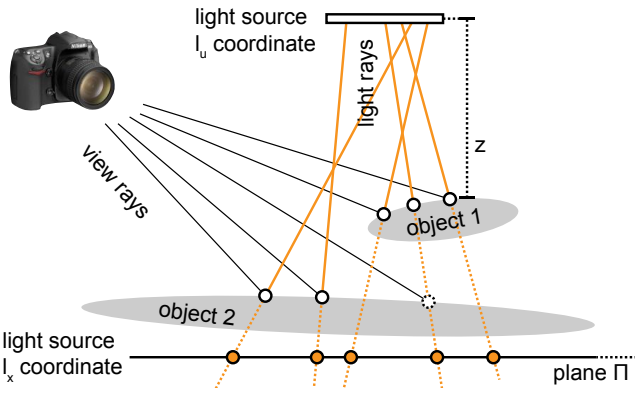

Figure 8: *To reconstruct shadows from area light sources, we first determine the points that are visible from camera (black rays). We shoot a ray from a point on the light source (parameterized by l_u, l_v) towards these points, and compute light-space depths z for the nearest hits as seen from the light. For shadowed view samples (dashed disk), the depth denotes the first occluder along the ray. The direction of light rays is parameterized by the the l_x, l_y coordinates of their intersection with the Π plane.*

We start by reprojecting all input samples to the center of the $(\hat{u}, \hat{v}, \hat{t})$ cube. We bucket the reprojected samples into regular bins on the xy plane, and construct an initial quadtree of the bins, so that the subdivision stops whenever there are fewer than a specified number of samples (we use 32) inside the node, or the node consists of only one bin. Additionally, we split leaf nodes that contain samples from more than one surface. Once the leaf nodes are finished, we construct a BVH using an agglomerative bottom-up algorithm [Walter et al. 2008].

The acceleration structure construction takes approximately one second for Figure 1 on a quad-core Intel Core i7 CPU.

5 Soft Shadows

The reconstruction of soft shadows from planar area lights is closely related to defocus. Depth of field is produced by integrating radiance over the 2D lens for each pixel; soft shadows are produced by integrating shadowed radiance from the 2D light source to a receiver point in the scene. However, two main differences must be accounted for. First, the density of light rays depends on the receiver points, and is therefore non-uniform. Second, the radiance contributed by a light ray changes along the ray, because the ray terminates at the nearest occluder.

Parameterization To leverage our defocus and motion blur machinery, we parameterize the light rays emitted from the light source using two parallel planes: the plane of the light source, and a plane Π placed at an arbitrary non-zero distance from the light (see Figure 8). The coordinates (l_u, l_v) specify points on the light source, and (l_x, l_y) encode the ray's direction by specifying the point where the ray hits plane Π. This is similar to the usual lens-focal plane parameterization for defocus. Each *light sample* $\{(l_x, l_y, l_u, l_v) \mapsto (z, L)\}$ records the depth z of the first hit from the direction of the light, and the radiance L carried by the ray onto the blocking surface. This 4D set of rays defines an *incident radiance light field*. Motion blurred shadows arising from moving lights, shadow casters, or camera are easily supported by incorporating time as a parameter of the light samples, and storing a motion vector in addition to z and L.

Algorithm The input samples to the shadow reconstruction algorithm are drawn from a 7D screen-time-lens-light domain. First, we render the screen-lens-time input samples from the camera's point of view, as before, and construct an acceleration hierarchy. For each scene point hit by a camera ray, we choose a point on the light source, and generate a light sample (Figure 8). A separate acceleration hierarchy is built from the light samples.

To reconstruct the radiance from a point $(\hat{l}_u, \hat{l}_v)$ on the light source to a point $\mathbf{p}$ in the scene, we first determine the $(\hat{l}_x, \hat{l}_y)$ coordinates of the ray from $(\hat{l}_u, \hat{l}_v)$ to $\mathbf{p}$ (Figure 8). These coordinates change linearly as a function of $(\hat{l}_u, \hat{l}_v)$, with slope determined by $\mathbf{p}$'s light-space depth [Egan et al. 2011]. The situation is now similar to defocus, with $(\hat{l}_x, \hat{l}_y)$ taking the place of the reconstruction location $(\hat{x}, \hat{y})$: reprojection and coverage testing using the light samples proceed as described earlier.

A crucial difference from Section 3 is that we must account for the change in visibility along the light ray. That is, a ray carries no light after it has hit the blocker. We handle this by comparing the light-space depths of the receiver $\mathbf{p}$ and those of the light samples, and setting $L = 0$ for light samples that terminate before $\mathbf{p}$. Filtering of the samples is otherwise unchanged. This means that a single light sample may contribute light to one surface and shadow to another. This is in contrast to [Egan et al. 2011], who render shadows due to distant occluders and must carefully decide where their filter can be applied.

To enable simultaneous motion blur, depth of field, and area shadows, we employ the above single-ray reconstruction in a fashion similar to path tracing. Specifically, we first reconstruct the scene point $\mathbf{p}$ visible at the camera reconstruction location $(\hat{x}, \hat{y}, \hat{u}, \hat{v}, \hat{t})$ using the camera samples. We then perform a lighting reconstruction for one point $(\hat{l}_u, \hat{l}_v)$ on the light using point $\mathbf{p}$ and the light samples. This path tracing formulation prevents the combinatorial explosion in the number of reconstructions. We reconstruct shadowed lighting using the same t as the camera sample to ensure that scene geometry is consistent between the camera and light.

Density Because the light samples are cast towards the hit points of the camera samples, their density in the (l_x, l_y) coordinate system is determined by the relative orientations of the light, camera, and the surface on which the camera sample lies. To account for the non-uniform density, we derive an analytic formula for the 2×2 Jacobian $J = \partial(l_x, l_y)/\partial(x, y)$ of the change l_x, l_y as a function of screen coordinates x, y, and store this density $\rho = \sqrt{|\det J|}$ in each light sample. The reconstruction algorithm uses this density for scaling R by ρ to account for the non-uniform density. The square root is necessary because the Jacobian measures area scaling, while R is a radius. To facilitate the computation of the view-light Jacobian, we augment the view samples with the depth derivatives $\partial z/\partial x$ and $\partial z/\partial y$ computed when taking the camera samples. Appendix B gives detailed formulae.

When computing J, we currently do not account for the additional changes in density caused by a finite lens and light size, i.e., our Jacobian is based on a pinhole camera and a pointlight. Also, the depth gradients are computed at $t = 0.5$. While Figure 11 shows that good results can be achieved with these approximations, validating and extending the density model remains future work.

6 Results and Discussion

We evaluate or algorithm on one scene that features defocus (BALCONY, Figure 1), two scenes that feature simultaneous defocus and motion blur (BUTTERFLIES and CHAIRS, Figure 9), and the EXCAVATOR scene (Figure 11) features simultaneous defocus, motion, and area lighting. The supplementary material contains videos that demonstrate our algorithm's temporal stability.

Figure 10: *Reconstruction from extremely limited information. Some discrepancies can be seen between our reconstruction and ground truth on the close-to-focus leaves, but overall the quality is high, especially for only one sample per pixel.*

Implementation details and test setup We have produced both a multithreaded CPU implementation and a CUDA GPU implementation of our algorithm. Currently, the GPU implementation can render defocus and motion blur, while soft shadows from area lights are still rendered on the CPU. Our test setup consists of an Intel Core i7 930 CPU at 2.8GHz with 12GB RAM, and an NVIDIA GeForce 480GTX GPU with 1.5GB RAM. All results, unless otherwise noted, use 128 reconstruction locations per pixel.

The input samples used for generating the results come from three sources: PBRT [Pharr and Humphreys 2010] (Figures 1, 15, 9), Pixie [Arikan 2009] (Figure 17), both modified to output the samples, and our own ray tracer (Figure 11). In addition, PBRT was modified to support linear world-space motion. This demonstrates the plug-and-play nature of our approach. The input samples are stored in a common, easy-to-parse file format, and no source-dependent processing was used for generating the results, with the exception of the shadow pipeline that requires additional information about the relative placement of the camera and light.

Defocus and motion Figure 1 features a balcony scene with heavily defocused foreground objects (foliage and balcony railing). The inputs consists of 16 low-discrepancy samples per pixel, rendered at 800×550 in 403 seconds using PBRT. Given these samples, our GPU implementation produced in 10 seconds an image of equal quality, subjectively as well as in terms of PSNR, to an 256 spp rendering from PBRT, which took 6426 seconds ($16\times$ longer).

Figure 9 features two scenes, BUTTERFLIES and CHAIRS. PBRT was used for generating the samples and the ground truth images. Images were rendered at 960×540. BUTTERFLIES features complex visibility due to the foliage and changing occlusions due to the motion of the butterfly on the right. Again, given the initial 16 samples per pixel, our reconstruction algorithm produced in about 10 seconds images comparable to 256 spp renderings from PBRT, corresponding to speedup of $16\times$.

The number of reconstruction locations in our algorithm is not directly comparable to the number of samples per pixel taken by PBRT, because each of our reconstruction locations uses multiple input samples for filtering its result. To verify this, we replaced our tent filtering of input samples with a nearest neighbor filter, which resulted in quality comparable to PBRT with the same number samples per pixel. Additionally, we excluded the possibility that differences in sampling patterns might affect the image quality noticeably by using PBRT's patterns in our implementation; the effect was negligible.

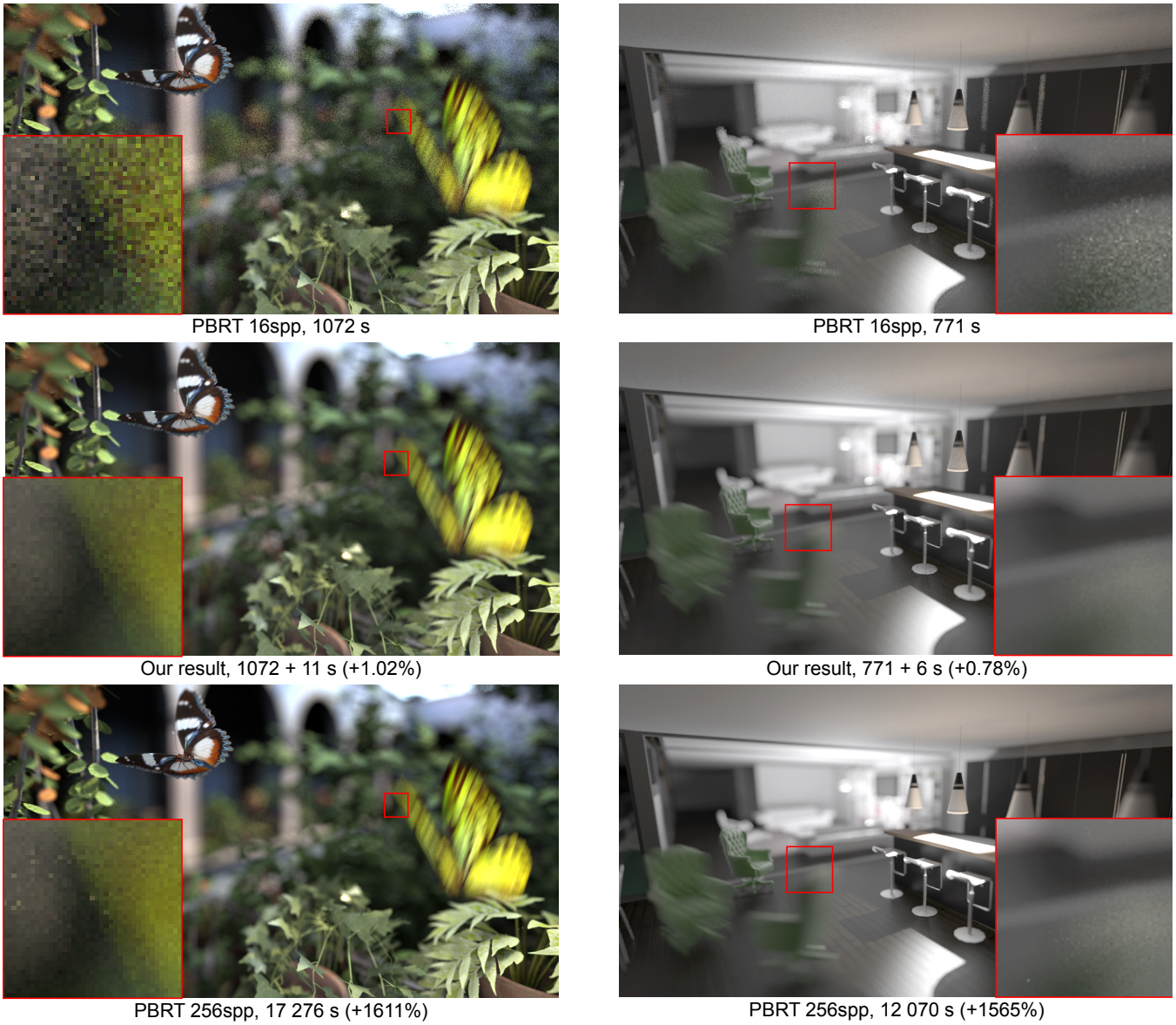

Figure 9: *Simultaneous depth of field and motion blur. Top row: Image rendered by PBRT using 16 samples per pixel. Middle row: Our reconstruction from the* same *16 samples per pixel. Bottom row: Equal quality PBRT rendering.*

Simultaneous area lights, defocus, motion In the EXCAVATOR scene (Figure 11) we reconstruct soft shadows, motion blurred soft shadows, and motion blurred soft shadows with depth of field from four 7D $(\hat{x}, \hat{y}, \hat{u}, \hat{v}, \hat{t}, \hat{l_x}, \hat{l_y})$ input samples per pixel. The top row shows images generated by axis-aligned filtering of the input samples, and the next row shows our reconstruction result.

This scene shows that our 5D $(\hat{x}, \hat{y}, \hat{u}, \hat{v}, \hat{t})$ reconstruction algorithm can be robustly extended to new integration domains (light), and that it performs surprisingly well even with very sparse input samplings. In particular, note how the left-hand excavator's shovel blends on top of the cockpit in the middle and right images. The accompanying videos demonstrate temporal stability of the reconstructed shadows, with and without motion and defocus. In these animations, each frame is sampled and reconstructed independently. Some geometric aliasing is visible in the high-frequency geometry, like the excavators' tracks, but that is a property of the input sampling, not our reconstruction algorithm. We stress that these results are deliberately computed with very sparse input samplings.

Numerical verification Figure 12 contains signal-to-noise ratios (PSNR) measured from our reconstruction as a function of both the number of input samples and the number of reconstruction locations, computed for the BUTTERFLIES scene. The ground truth was computed using 1024 samples per pixel using PBRT. We computed the PSNRs after applying gamma and clamping the results to the 0-255 range, but before quantization. It can be observed that our algorithm yields a considerable improvement over axis-aligned filtering, even when using a very low number of input samples, and matches ground truth well, as demonstrated by the PSNRs in the high 30s. As demonstrated by Figure 10, reasonable results can be obtained even from very limited samplings (1 input sample/pixel).

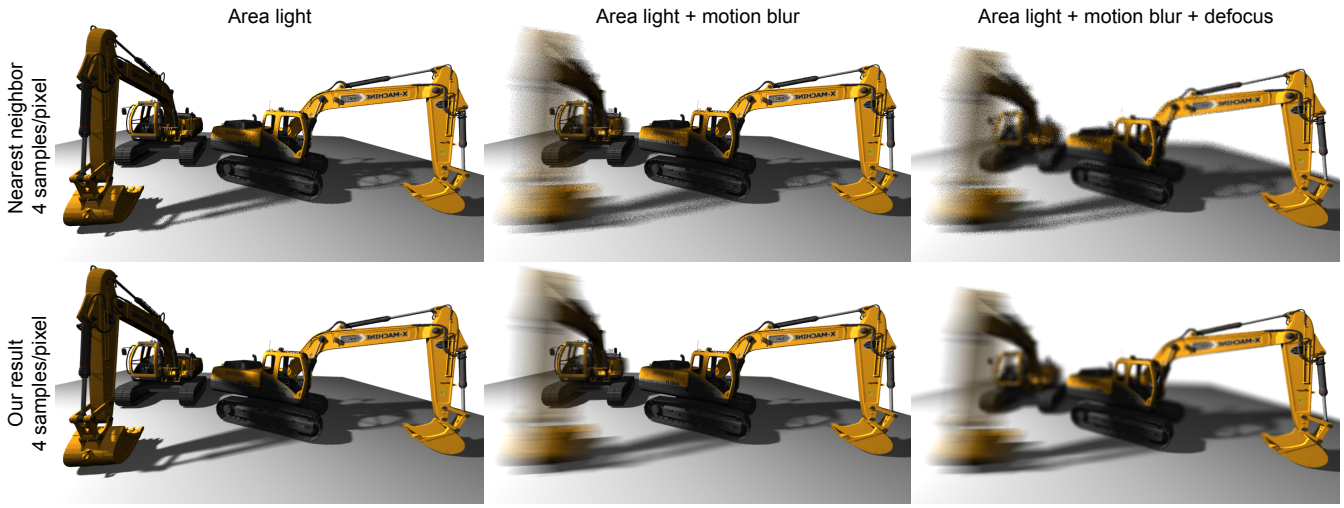

Figure 11: *Physically-based soft shadows with motion blur and depth of field. Top row: Nearest neighbor filtering, four samples per pixel. Bottom row: Our reconstruction, computed from the same four samples per pixel as the top images.*

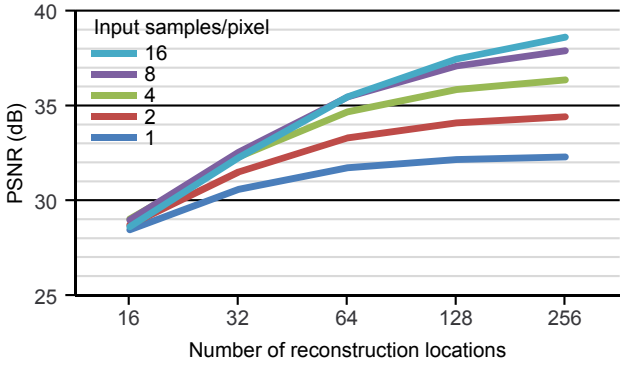

Figure 12: *PSNR measurements vs. ground truth for* BUTTERFLIES *as a function of #reconstruction locations and #input samples. Axis-aligned filtering gave 13.6dB, 16.3dB, 19.5dB, 23.1dB and 27.6dB for 1, 2, 4, 8 and 16 input samples per pixel.*

Figure 13: *Left: Reconstruction using the original focus distance of input samples. Right: Reconstruction from the same samples using a modified focusing distance. See the supplementary material for a video showing a continuous rack focus.*

Fast refocusing It is easy to refocus images during the reconstruction stage using the same input samples. This can be done by changing the lens model parameters C_1, C_2 after the samples have been reprojected to the center of the (u, v, t) domain using the original lens parameters, but before building the acceleration structure. This modifies the input samples' slopes $\partial(x, y)/\partial(u, v)$, or equivalently, shears the lens integration kernel [Ng 2005]. Changing the aperture is achieved the same way, by scaling C_1 and C_2. As demonstrated in Figure 13, our algorithm enables high-quality refocusing in a matter of seconds. The supplementary material includes a video demonstrating refocusing on the BALCONY scene of Figure 1 from 16 spp input, demonstrating the stability of the reconstruction under parameter changes. All frames of the animation are reconstructed from the same input samples. Our method scales to a large number of reconstruction locations, and we use 2048 per pixel in this video because the upper right of the scene contains extremely high contrast due the HDR environment map.

Discussion of failure cases Our visibility heuristic detects sets of samples that have no significant relative parallax over time and the lens. Even if visibility is undersampled due to high-frequency geometry, e.g., distant foliage, the samples can be correctly filtered together, unlike in object space groupings. For this to fail, the samples would have to originate from very high-frequency geometry that has significant relative parallax due to defocus or motion. To alleviate this, potentially undersampled surfaces are merged with the next one, effectively blending them together, as described in Sec. 3.3. This happens rarely, and due to the parallax, only on surfaces that are blurred due to motion or defocus anyway.

Our algorithm is based on the assumption that the input sampling captures the frequency content of the integrand. Consequently, if the input samples are aliased, we introduce a low-frequency error, just like traditional sampling and reconstruction. Away from visibility boundaries, this is avoided by usual prefiltering methods (MIP-maps, ray differentials).

Bandwidth information and view-dependent effects Figure 14 demonstrates a proof-of-concept implementation which utilizes additional information about shading bandwidth caused by view-dependent effects. Based on an estimate of the bandwidth of the reflectance function [Durand et al. 2005], we weight each reconstruction sample by its distance from the reconstruction location on the uv plane by a Gaussian matched to the BRDF bandwidth to

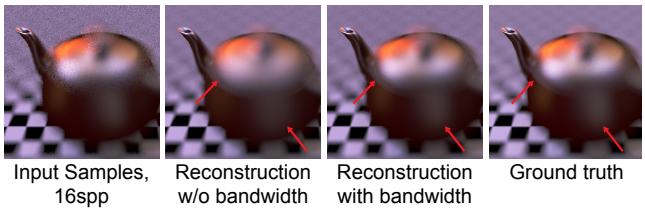

Figure 14: *The effect of additional bandwidth information illustrated on a defocused glossy teapot under environment lighting.*

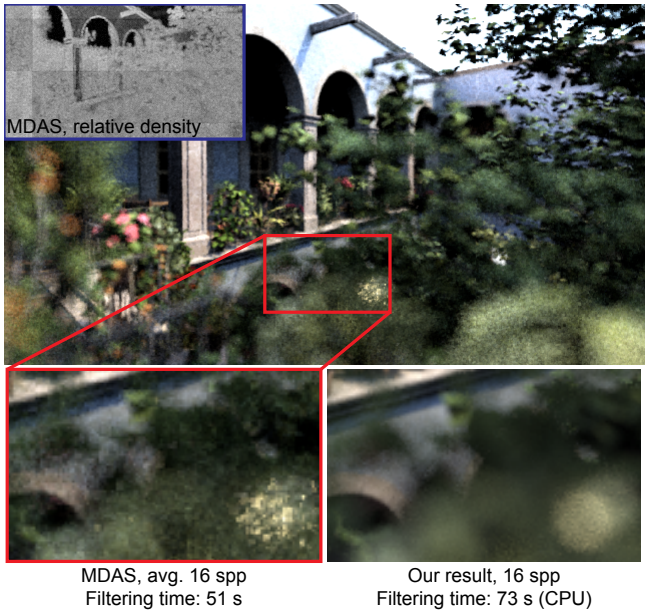

Figure 15: *MDAS applied to the scene from Figure 1. Average sampling rate is 16 samples per pixel. Top: Image rendered by MDAS, with the inset showing the relative density of samples taken. Lower left: Detail of the image reconstructed by MDAS. Lower right: Our result. Reported timings include only filtering. These numbers refer to the CPU implementations; on GPU our filtering took 10 seconds.*

account for view-dependent variation along the 5D trajectories. To match the effective drop in the number of input samples that affect each reconstruction location, we sample the glossy object proportionally more, leading to a non-uniform distribution of the input samples. Combining our filter with predictions of the shape of the reflected spectrum [Durand et al. 2005; Soler et al. 2009; Egan et al. 2009] remains an interesting avenue of future work.

6.1 Comparisons

MDAS Figure 15 shows a comparison between our depth of field reconstruction and multidimensional adaptive sampling [Hachisuka et al. 2008]. Because we have only a CPU implementation of MDAS, we quote timings from our CPU implementation to ensure fairness. The scene is difficult for the contrast-based algorithm because it contains complex visibility and lighting, which cause problems for the anisotropy estimator that works based on radiance samples alone. As can be seen, this results in a noisy final image in comparison to our result. As discussed by Hachisuka et al. [2008], MDAS could likely be extended to utilize additional anisotropy information.

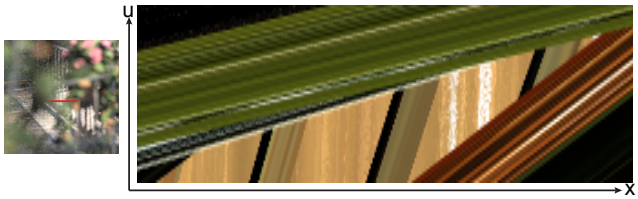

Figure 16: *An xu slice through the screen-lens light field in an 8-pixel-wide region in the San Miguel scene (slice highlighted in red in the left image). The complex visibility over the lens and the samples' varying slopes prohibit sheared linear filtering.*

Sheared reconstruction Egan et al. [2009; 2011] share our goal of reconstructing an image from a relatively sparse sampling, aided by auxiliary information about the integrand. Their spectrum model predicts crucial phenomena such as image bandwidth reduction resulting from motion (resp. circle of confusion, blocker depth), and can be used for driving adaptive sampling. However, some issues arise when reconstruction is performed using a linear filter derived from the same model.

First, the reconstruction falls back to axis-aligned (traditional) filtering near occlusion boundaries or regions of inhomogeneous motion. This happens because in such areas the local spectrum of the light field is too complex to allow a wider spatial filter. The effect is visible in the BALLERINA scene (Figure 17) near the boundaries and around the central fold of the dress. Furthermore, depth of field can exacerbate the issue. Figure 16 shows an xu slice of a complex screen-lens light field with many objects at different depths, resulting in a complex function with occlusion and multiple slopes. While Egan et al. [2009] introduced their algorithm for motion blur, it can be also be applied to defocus without significant changes. When we did this, only about 30% of the pixels in Figure 1 could be filtered using a sheared filter; in other areas the algorithm reverted to axis-aligned filtering. In contrast, our algorithm handles these difficult situations thanks to the per-sample slopes and explicit visibility determination.

Second, the model assumes that the integration filters, i.e., shutter response, lens aperture, and light intensity, are bandlimited (in practice, Gaussians). This prohibits the use of box filters, which is crucial for obtaining faithful depth of field reconstruction, as camera apertures are not Gaussian. In our experiments with sheared reconstruction filters, using a box filter against the theory leads to noticeable aliasing artifacts.

Third, the model is derived in an infinite screen-time (resp. screen-lens, screen-light) domain. This forces one to draw samples *outside* the effective width of the shutter interval (resp. lens, light source) to avoid aliasing.

The sheared filter of Egan et al. [2009] and our reconstruction agree when the samples come from a single fronto-parallel planar object moving uniformly in the direction of the image plane (no depth changes). In this case the light field slopes of all samples are the same. If we furthermore apply a Gaussian weighting on the time (or lens) axes, our filter reduces to exactly the same sheared linear filter as predicted by Egan et al.'s analysis [2009].

6.2 Timing breakdown and memory consumption

In our tests, the time taken by different parts of the reconstruction algorithm is distributed as follows. The tree construction, which is currently performed on the CPU, accounts for around 10% for the GPU variant and around one percent when filtering on the CPU in BUTTERFLIES. Querying the tree, including testing the leaves

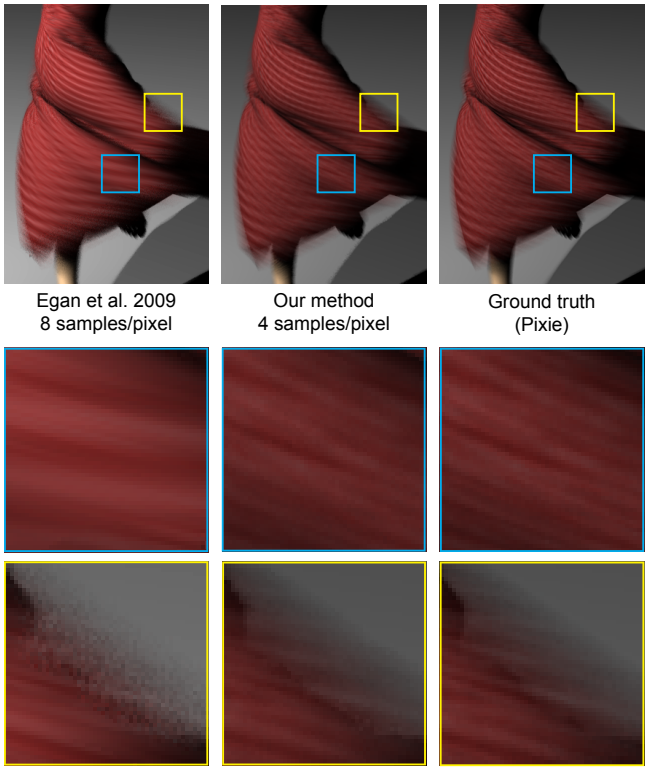

Figure 17: *Comparison of motion blurred results between sheared reconstruction, our method, and ground truth. Even though sheared reconstruction gets the large scale movement right, some deviations from ground truth remain even in the homogeneously moving regions. The insets show that our algorithm yields results closer to ground truth, also in regions where sheared reconstruction must fall back to axis-aligned filtering due to occlusion boundaries or inconsistent motion. Differences in moving shadows result from a content pipeline issue that prevents us from running our shadow filter on this image. As a result, surface shadows are treated like texture. The scene was rendered in a resolution of* 512×512*. Our reconstruction took* 2.3 *seconds (GPU).*

using SAMESURFACE, reprojecting the individual samples, and filtering when triangulation is not necessary, takes roughly 80% of the time on the GPU (resp. 90% on CPU), with the rest of the time used for triangulation. The statistics are similar for the other scenes. In practice, execution time scales linearly with the number of reconstruction locations.

Each input sample consumes 64 bytes of memory. The most significant contributors to the overall memory consumption of our algorithm in BUTTERFLIES are the input samples (510MB) and and the acceleration structure (116MB), i.e., the memory consumption is almost entirely determined by the number of input samples.

7 Conclusion

Our technique enables accurate reconstruction of the temporal light field, making it possible to create high-quality renderings featuring simultaneous motion blur, depth of field, and area shadows from relatively sparse input samplings. We improve on previous work on two crucial aspects. First, we are not limited to a single anisotropic direction per reconstruction location, but instead use the accurate trajectory for each individual input sample. Second, we introduced novel methods for determining the visibility consistency of stochastic light field samples and for resolving their visibility, enabling reconstruction in presence of object boundaries. The algorithm has no scene-dependent parameters, and has been successfully applied to several scenes obtained from multiple rendering systems at varying sampling densities. Our method is fast, especially when implemented on the GPU, and additionally enables effects such as refocusing.

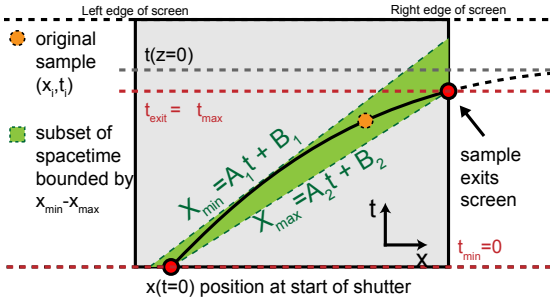

Figure 18: *Bounding the reprojection of a sample. (See text.)*

Our approach to reprojection and visibility determination could also be applied to other integration domains, such as incident radiance for glossy reflection. While we have demonstrated that bandwidth information may be used in our reconstruction algorithms, we feel that combining our technique more closely with sophisticated, non-uniform, adaptive sampling based on predictions of the anisotropic spectra and bandwidth resulting from e.g. gloss, curvature and defocus [Durand et al. 2005; Soler et al. 2009; Egan et al. 2009] should enable further savings.

A Bounding Planes

We construct the planes used for bounding the input samples' motion under reprojections as follows. We first construct planes for each individual sample, and then merge the planes as the samples are grouped first to leaves and then internal nodes. We first perform fits for X and Y bounding lines as two separate 2D (xt and yt) problems, and account for defocus, which is separable from motion, afterwards. Note that in contrast to motion BVHs used in ray tracing, we bound projected screen coordinates. See Figure 18.

We first determine the time interval when the sample is in front of the camera: $[0, t_{z=0}]$ if the sample starts in front, and $[t_{z=0}, 1]$ if the sample is behind the camera at $t = 0$. We subsequently compute the time instants t_{enter} and t_{exit} when the sample potentially enters and exits the screen — the sample can enter the screen if its $t = 0$ position is outside the screen and vice versa — and use these to further trim the time interval to only those times $[t_{\text{min}}, t_{\text{max}}]$ when the sample is potentially on-screen. We then fit two lines that bound $x(t)$ from right and left. Note that these lines are not coincident, because the trajectory $x(t)$ is typically curved due to perspective. After repeating for y, we have four lines that satisfy $A_1t + B_1 \leq x(t) \leq A_2t + B_2$ and $C_1t + D_1 \leq y(t) \leq C_2t + D_2$. The additional effect of defocus is subsequently handled by evaluating $\partial(x, y)/\partial(u, v)$ at t_{min} and t_{max}, and lifting the lines into 3D planes, potentially adjusting the constants B and D, such that the changes in x and y due to defocus are bounded by the result.

When constructing tree nodes, we need to merge two sets of bounding planes. We first conservatively extend the $[t_{\text{min}}, t_{\text{max}}]$ range to enclose the ranges from both inputs. Subsequently, we evaluate both sets of plane equations at the ends of the new interval, and at the $[-1, 1]$ extremities of the lens coordinates, and fit new bounding planes to these points.

B View-light Jacobian

We seek the Jacobian $J = \partial(l_x, l_y)/\partial(x, y)$, where (x, y) are screen coordinates and (l_x, l_y) are coordinates on the light's image plane. Given the screen coordinates x, y in clip units $[-1, 1]^2$, the depth z, and the partials $\partial z/\partial(x, y)$ of the surface where the camera sample hits, we locally linearize the surface being viewed. Let $X(x, y) = x\, z(x, y)$ and $Y(x, y) = y\, z(x, y)$ be 3D clip space points that project to (x, y). Note that z varies with (x, y) because the surface is in general orientation. Now, by the product rule,

$$\frac{\partial X}{\partial x} = z + x\frac{\partial z}{\partial x},\ \frac{\partial X}{\partial y} = x\frac{\partial z}{\partial y},\ \frac{\partial Y}{\partial x} = y\frac{\partial z}{\partial x},\ \frac{\partial Y}{\partial y} = z + y\frac{\partial z}{\partial y},$$

and the vectors $(\frac{\partial X}{\partial x}, \frac{\partial Y}{\partial x}, \frac{\partial z}{\partial x})^{\mathrm{T}}$, $(\frac{\partial X}{\partial y}, \frac{\partial Y}{\partial y}, \frac{\partial z}{\partial y})^{\mathrm{T}}$ form a basis for the locally linearized 3D surface. In other words, points on the plane can be written as

$$\begin{pmatrix} X(\alpha,\beta) \\ Y(\alpha,\beta) \\ z(\alpha,\beta) \end{pmatrix} = \begin{pmatrix} \frac{\partial X}{\partial x} & \frac{\partial Y}{\partial x} & xz \\ \frac{\partial Y}{\partial x} & \frac{\partial Y}{\partial y} & yz \\ \frac{\partial z}{\partial x} & \frac{\partial Y}{\partial z} & z \end{pmatrix} \begin{pmatrix} \alpha \\ \beta \\ 1 \end{pmatrix}$$

for some (α, β), where we denote the matrix by $\mathbf{M}$. Therefore, $(\alpha, \beta, 1)^{\mathrm{T}} = \mathbf{M}^{-1}(X, Y, z)^{\mathrm{T}}$, and $1/z = \mathbf{M}^{-1}(3, :)(x, y, 1)^{\mathrm{T}}$, where $\mathbf{M}^{-1}(3, :)$ denotes the third row of the inverse[6]. Now, the camera-space homogeneous point $(X, Y, z, 1)$ is projectively equivalent to $(x, y, 1, 1/z)$, which is obtained from (x, y) by

$$\mathbf{A}\,(x, y, 1)^{\mathrm{T}}, \quad \text{with } \mathbf{A} = \begin{pmatrix} \mathbf{I}_{3\times 3} \\ \mathbf{M}^{-1}(3, :) \end{pmatrix}.$$

Finally, we apply the projective camera-to-light transformation $\mathbf{L}$ to yield $(l_x, l_y, l_z, 1)^{\mathrm{T}} \sim \mathbf{L}\mathbf{A}\,(x, y, 1)^{\mathrm{T}}$, where $\sim$ denotes usual projective equivalence. We have a rational linear expression for (l_x, l_y) in terms of (x, y), which is easy to differentiate to get J.

Acknowledgments

We thank Kevin Egan for help with the Ballerina scene and fruitful discussions on sheared reconstruction; Jonathan Ragan-Kelley for early brainstorming; George Drettakis for helpful comments; Guillermo M. Leal Llaguno for the Balcony scene on which Butterflies is based; Florent Boyer for the scene on which Chairs is based; Daniel Genrich for the Ballerina scene. This work was partially funded by the MIT-Singapore Gambit lab and a grant from Intel.

References

ARIKAN, O., 2009. Pixie – Open source RenderMan. http://www.renderpixie.com.

BOLIN, M. R., AND MEYER, G. W. 1995. A frequency based ray tracer. In *Proc. ACM SIGGRAPH 95*, 409–418.

CHAI, J.-X., TONG, X., CHAN, S.-C., AND SHUM, H.-Y. 2000. Plenoptic sampling. In *Proc. ACM SIGGRAPH 2000*, 307–318.

CHEN, S. E., AND WILLIAMS, L. 1993. View interpolation for image synthesis. In *Proc. ACM SIGGRAPH 93*, 279–288.

COOK, R. L., PORTER, T., AND CARPENTER, L. 1984. Distributed ray tracing. In *Computer Graphics (Proc. ACM SIGGRAPH 84)*, vol. 18, 137–145.

COOK, R. L., CARPENTER, L., AND CATMULL, E. 1987. The Reyes image rendering architecture. In *Computer Graphics (Proc. ACM SIGGRAPH 87)*, vol. 21, 95–102.

[6]We've shown the well known result that $1/z$ is linear in screen space, but derived it using the partials of depth computed at one point.

DURAND, F., HOLZSCHUCH, N., SOLER, C., CHAN, E., AND SILLION, F. X. 2005. A frequency analysis of light transport. *ACM Trans. Graph. 24*, 3, 1115–1126.

EGAN, K., TSENG, Y., HOLZSCHUCH, N., DURAND, F., AND RAMAMOORTHI, R. 2009. Frequency analysis and sheared reconstruction for rendering motion blur. *ACM Trans. Graph. 28*, 3, 93:1–93:13.

EGAN, K., HECHT, F., DURAND, F., AND RAMAMOORTHI, R. 2011. Frequency analysis and sheared filtering for shadow light fields of complex occluders. *ACM Trans. Graph. 30*, 2, 9:1–9:13.

GLASSNER, A. 1988. Spacetime ray tracing for animation. *IEEE Computer Graphics and Applications 8*, 2, 60–70.

HACHISUKA, T., JAROSZ, W., WEISTROFFER, R. P., DALE, K., HUMPHREYS, G., ZWICKER, M., AND JENSEN, H. W. 2008. Multidimensional adaptive sampling and reconstruction for ray tracing. *ACM Trans. Graph. 27*, 3, 33:1–33:10.

LEE, S., EISEMANN, E., AND SEIDEL, H.-P. 2010. Real-time lens blur effects and focus control. *ACM Trans. Graph. 29*, 4, 65:1–65:7.

MAX, N., AND LERNER, D. 1985. A two-and-a-half-d motion-blur algorithm. In *Computer Graphics (Proc. ACM SIGGRAPH 85)*, vol. 19, 85–93.

NEHAB, D., SANDER, P., LAWRENCE, J., TATARCHUK, N., AND ISIDORO, J. 2007. Accelerating real-time shading with reverse reprojection caching. In *Proc. Graphics hardware*, 25–35.

NG, R. 2005. Fourier slice photography. *ACM Trans. Graph. 24*, 3, 735–744.

NGUYEN, H. 2007. *GPU Gems 3*. Addison-Wesley Professional.

OVERBECK, R., DONNER, C., AND RAMAMOORTHI, R. 2009. Adaptive wavelet rendering. *ACM Trans. Graph. 28*, 5, 140:1–140:12.

PHARR, M., AND HUMPHREYS, G. 2010. *Physically Based Rendering, 2nd ed.* Morgan Kauffmann.

POTMESIL, M., AND CHAKRAVARTY, I. 1983. Modeling motion blur in computer-generated images. In *Computer Graphics (Proc. ACM SIGGRAPH 83)*, vol. 17, 389–399.

RAGAN-KELLEY, J., LEHTINEN, J., CHEN, J., DOGGETT, M., AND DURAND, F. 2011. Decoupled sampling for real-time graphics pipelines. *ACM Trans. Graph.*. To appear.

RAMAMOORTHI, R., AND HANRAHAN, P. 2004. A signal-processing framework for reflection. *ACM Trans. Graph. 23*, 1004–1042.

SOLER, C., SUBR, K., DURAND, F., HOLZSCHUCH, N., AND SILLION, F. 2009. Fourier depth of field. *ACM Trans. Graph. 28*, 2, 18:1–18:12.

WALTER, B., DRETTAKIS, G., AND PARKER, S. 1999. Interactive rendering using the render cache. In *Proc. Eurographics Workshop on Rendering*, 235–246.

WALTER, B., BALA, K., KULKARNI, M., AND PINGALI, K. 2008. Fast agglomerative clustering for rendering. In *Proc. Symposium on Interactive Ray Tracing*, 81–86.

YU, X., WANG, R., AND YU, J. 2010. Real-time depth of field rendering via dynamic light field generation and filtering. *Comput. Graph. Forum 29*, 7, 2099–2107.