Fast 4D Sheared Filtering for Interactive Rendering of Distribution Effects

LING-QI YAN and SOHAM UDAY MEHTA
University of California, Berkeley
RAVI RAMAMOORTHI
University of California, San Diego
and
FREDO DURAND
MIT CSAIL

Soft shadows, depth of field, and diffuse global illumination are common distribution effects, usually rendered by Monte Carlo ray tracing. Physically correct, noise-free images can require hundreds or thousands of ray samples per pixel, and take a long time to compute. Recent approaches have exploited sparse sampling and filtering; the filtering is either fast (axis-aligned), but requires more input samples, or needs fewer input samples but is very slow (sheared). We present a new approach for fast sheared filtering on the GPU. Our algorithm factors the 4D sheared filter into four 1D filters. We derive complexity bounds for our method, showing that the per-pixel complexity is reduced from $O(n^2l^2)$ to $O(nl)$, where n is the linear filter width (filter size is $O(n^2)$) and l is the (usually very small) number of samples for each dimension of the light or lens per pixel (spp is l^2). We thus reduce sheared filtering overhead dramatically. We demonstrate rendering of depth of field, soft shadows and diffuse global illumination at interactive speeds. We reduce the number of samples needed by $5-8\times$, compared to axis-aligned filtering, and framerates are $4\times$ faster for equal quality.

Categories and Subject Descriptors: I.3.7 [**Computer Graphics**]: Three-Dimensional Graphics and Realism—*Color, shading, shadowing, and texture*

General Terms: Algorithms, Performance, Design

Additional Key Words and Phrases: Fourier analysis, sampling, filtering, soft shadows, depth of field, diffuse global illumination, sheared filtering

ACM Reference Format:

Ling-Qi Yan, Soham Uday Mehta, Ravi Ramamoorthi, and Fredo Durand. 2015. Fast 4D sheared filtering for interactive rendering of distribution effects. ACM Trans. Graph. 35, 1, Article 7 (December 2015), 13 pages. DOI: http://dx.doi.org/10.1145/2816814

This work was supported in part by NSF grants 1115242, 1451830, the Intel Science and Technology Center for Visual Computing, and equipment and fellowships from NVIDIA.

Authors' Email Addresses: L.-Q. Yan, S. U. Mehta: {lingqi, sohamum}@cs.berkeley.edu; R. Ramamoorthi: ravir@cs.ucsd.edu; F. Durand: fredo@mit.edu.

1. INTRODUCTION

Monte Carlo distribution raytracing is an accurate way to render effects such as depth-of-field, soft shadows and indirect illumination. But convergence to a noise-free image is slow, often requiring over a thousand rays per pixel. Thus, there is a considerable interest in fast adaptive sampling and filtering approaches.

Fortunately, there is significant coherence in the intensity between pixels. Our work is closest to the 4D sheared filtering methods, pioneered by Egan et al. [2009]. These methods perform a careful frequency analysis to determine near-optimal sampling rates for a number of different effects, such as motion blur, soft shadows and spherical harmonic/ambient occlusion [Egan et al. 2011a; Egan et al. 2011b]. While the sample count reductions are dramatic, with very few additional assumptions, these sheared filtering techniques are usually memory intensive and have high reconstruction overheads. One of the key challenges is the irregular search for samples. Even if the initial samples are stratified, they are distributed irregularly once one considers the footprint of the 4D sheared filter for each pixel. Therefore, inspite of numerous efforts to accelerate the basic sheared filtering algorithm, it remained a slow process taking several minutes per frame for reconstruction, often dwarfing the cost of even offline raytracing.

Thus, sheared reconstruction was established as a theoretically sound technique that reduced sample counts by one to two orders of magnitude. But it was not practical for fast or interactive raytracing systems, since irregular sampling and high memory usage made reconstruction too expensive. Methods based on axis-aligned filtering [Mehta et al. 2012, 2013, 2014] were developed in the past three years in response to this, to bring sampling and filtering into the real-time domain. There is a significant tradeoff in sample count, with axis-aligned filtering requiring an order of magnitude more samples than 4D sheared filtering. Nevertheless, the simplicity of the filter and its natural separability in pixel-light, pixel-time or equivalent space can be exploited to minimize filtering time, and enable inclusion in interactive raytracing systems. However, one needs many more input ray samples, since the simple filter doesn't bound the frequency spectrum tightly.

In this article, we describe a solution to the now long-standing problem of fast 4D sheared filtering, showing that the sample count tradeoff in axis-aligned filtering methods is no longer needed, for the common visual effects of soft shadows, depth of field, and diffuse global illumination. Indeed, we achieve the best of both worlds—the low sampling rates of sheared filtering, and reconstruction times comparable with axis-aligned filtering.

We start from the 4D pixel-light sheared filter [Egan et al. 2011b] in the primal domain for soft shadows (a similar analysis applies

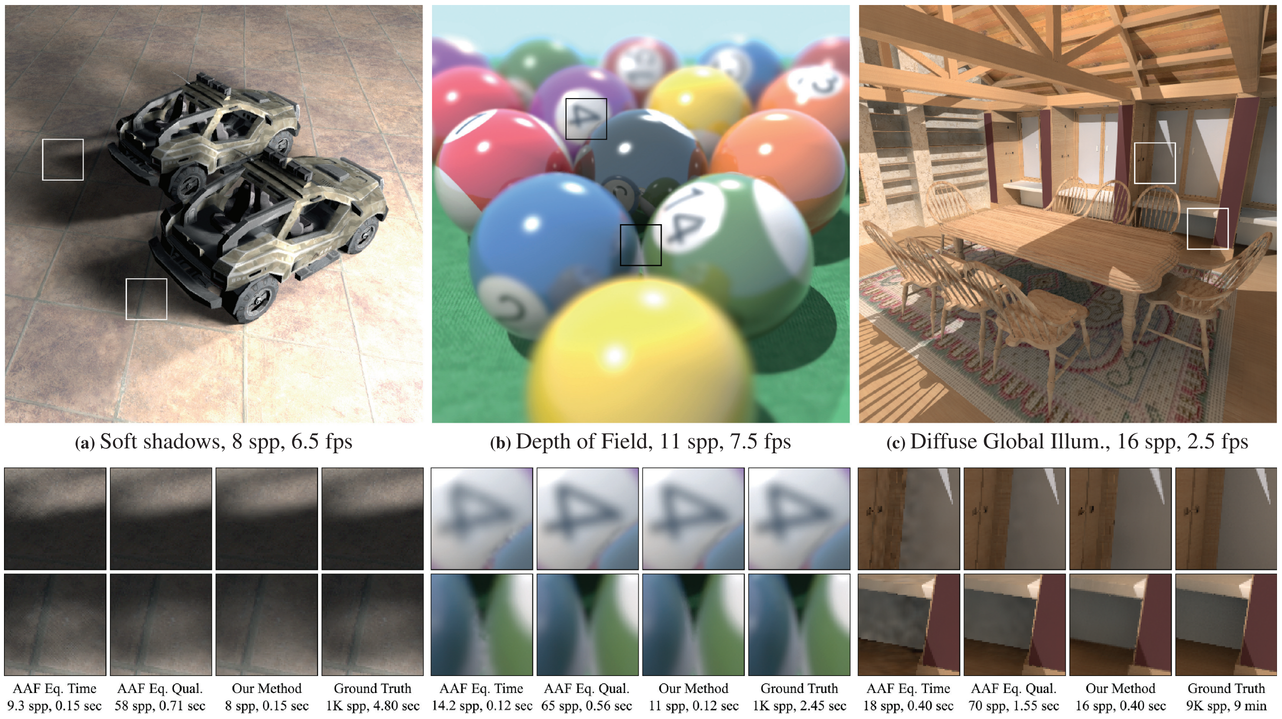

(a) Soft shadows, 8 spp, 6.5 fps
(b) Depth of Field, 11 spp, 7.5 fps
(c) Diffuse Global Illum., 16 spp, 2.5 fps
AAF Eq. Time 9.3 spp, 0.15 sec
AAF Eq. Qual. 58 spp, 0.71 sec
Our Method 8 spp, 0.15 sec
Ground Truth 1K spp, 4.80 sec
AAF Eq. Time 14.2 spp, 0.12 sec
AAF Eq. Qual. 65 spp, 0.56 sec
Our Method 11 spp, 0.12 sec
Ground Truth 1K spp, 2.45 sec
AAF Eq. Time 18 spp, 0.40 sec
AAF Eq. Qual. 70 spp, 1.55 sec
Our Method 16 spp, 0.40 sec
Ground Truth 9K spp, 9 min

Fig. 1. We can render soft shadows (CARS, two area lights), defocus blur (POOL, two point lights, modified from NVIDIA OptiX SDK) and diffuse global illumination (ROOM, one point light) at interactive speeds by fast 4D sheared filtering on a sparsely sampled Monte Carlo (MC) input, which is very noisy as seen in the insets. We require very low sampling rates, often under 16 samples per pixel (spp). Compared to axis-aligned filtering (AAF) with adaptive sampling [Mehta et al. 2012, 2013], we perform 4× faster, and reduce the sampling rate required by 5-8×.

to depth of field and indirect illumination). Inspired by the natural separability of axis-aligned filtering, we come up with a solution that handles the high-dimensional sheared filtering by factorizing it into lower-dimensional forms. This overcomes the problem of expensive irregular search for samples, caused by the shearing that couples pixel and light dimensions [Egan et al. 2011b]. Besides the theoretical contribution of fast high dimensional filtering, that bridges sheared and axis-aligned filtering algorithms, we dramatically reduce the practical computational cost, achieving a 4x faster implementation compared to Mehta et al. [2012, 2013], and orders of magnitude faster than Egan et al. [2009, 2011b]. Specifically, we make the following contributions:

Factoring 4D sheared filter into four 1D filters. We first observe that the 4D sheared filter is a product of two 2D sheared filters along orthogonal pixel-light planes, and develop a two step factored algorithm. We then derive a further factorization into four 1D integrals, that separate the 2D sheared shape into a pre-convolution and a collection. The computational complexity[1] per pixel is reduced from $O(n^2l^2)$ to $O(nl)$, where n is the linear filter size (along one dimension) and l^2 is the number of samples per pixel (so l is the number of samples along each dimension of the lens or light). In sheared filtering, l can be very small, typically $l \leq 4$ and the samples per pixel (spp) $l^2 \leq 16$. Thus, the complexity is comparable to the $O(n)$ cost of (fully factored) axis-aligned filtering.[2]

Efficient GPU Implementation of Sheared Filter. With an efficient GPU (CUDA) implementation of the factored sheared filter, we reduce filtering time per frame to about 70 msec. In comparison, a direct implementation of the 4D sheared filter takes one to two orders of magnitude longer. This is the first general implementation of fast 4D sheared filtering that gives interactive performance.

Interactive Rendering of Distribution Effects. We demonstrate accurate results for soft shadows, depth of field, and diffuse global illumination with only 6-16 samples per pixel (spp), as shown in Figure 1. (In the main body of this article, we consider only single effects at a time; handling multiple effects simultaneously as in Mehta et al. [2014] is discussed briefly in the Appendix, with example images). Even though the input data is very noisy, we are able to perform high quality reconstruction. Our results match ground truth closely, which is typically obtained with 100× the number of samples per pixel (see Figures 6–8). We implement our filtering algorithm on the GPU-based real-time Optix raytracer, and demonstrate a 4× speedup in framerate over equal quality axis-aligned filtering, while reducing sample counts by $5 - 8\times$.

2. PREVIOUS WORK

Our work builds on a recent history of methods for adaptive image filtering to remove noise in ray traced solutions, but most of these methods were not intended for real-time use. Our approach also relates to Fourier and light field reconstruction techniques, as well

[1]The full complexity is $O(nl + l^3)$ but the $O(nl)$ term is dominant, as explained later.

[2]The fast sheared filter is still somewhat more expensive, both from the l factor, and because of the larger size n of the sheared filter. This is more than made up for, by the much smaller number of ray samples that are needed by our method, as compared to axis-aligned filtering.

as initial approaches for fast sheared filtering for depth of field and motion blur.

Image and Adaptive Filtering. Image filtering has a long history, including [Rushmeier and Ward 1994; McCool 1999]. Adaptive image sampling also has a long history, with seminal work by Mitchell [1991]. Recently, Hachisuka et al. [2008] presented multi-dimensional adaptive sampling and anisotropic reconstruction, that has inspired much follow-on work. Recent work also includes adaptive wavelet rendering [Overbeck et al. 2009], the A-Trous wavelet transform [Dammertz et al. 2010], cross bilateral filters [Petschnigg et al. 2004; Paris and Durand 2006] and filtering of stochastic buffers [Shirley et al. 2011]. A significant advance is random parameter filtering [Sen and Darabi 2012] which seeks to separate variation from random parameters and geometric signals. Other recent works are based on statistical theories like SURE [Li et al. 2012] and nonlocal means filtering [Rouselle et al. 2012]. Recently, Kalantari and Sen [2013] developed a method to locally identify noise in different parts of the image, followed by standard adaptive sampling and denoising, while Delbracio et al. [2014] use ray color histograms. However, these methods do not exploit the Fourier structure of the higher-dimensional light field, and typically require high sampling rates with offline reconstruction; they are not interactive.

Real-time Distribution Effects. Real-time soft shadows are commonly produced using soft shadow maps that consider occlusion from the entire area source [Guennebaud et al. 2007; Annen et al. 2008]. As noted in Johnson et al. [2009], these methods make various tradeoffs of speed and accuracy. Soler and Sillion [1998] provide an analytic solution, but only for geometry in parallel planes. Shadow volumes [Crow 1977] can also be extended to soft shadows using geometric ideas like penumbra wedges [Assarsson and Möller 2003] and shadow volumes [Laine et al. 2005]. Another body of work is precomputed relighting [Sloan et al. 2002], but it is usually limited to static scenes lit by environment maps. Analogously, for real-time depth of field, the general approach is to rasterize layers using a pinhole camera [Lee et al. 2010; Lei and Hughes 2013], and then splat and gather the samples on the image plane to approximate defocus blur for a particular focus depth. There are also simpler post-processing algorithms [Potmesil and Chakravarty 1981; Yu et al. 2010] that use a single pinhole rendering and depth buffer to simulate defocus blur.

Real-time approximate global illumination techniques (a survey can be found in Ritschel et al. [2012]) include voxel-based cone tracing [Crassin et al. 2011] on the GPU. Point-based approaches include micro-rendering such as Ritschel et al. [2009], which raytraces shading points and partitions them by k-means, and then does a final gather using GPU-based photon mapping.

Although these approaches are commonly used for their high performance, they make approximations that can produce aliasing and other artifacts. Our method is based on unbiased Monte-Carlo sampling, and can offer high-quality results with nearly the same speed.

Fourier and Light Field Analysis. Our goal is to obtain low sample counts from sheared filtering [Egan et al. 2009, 2011a, 2011b], while achieving interactive filter times comparable to axis-aligned filtering methods [Mehta et al. 2012, 2013]. Our method applies to any sheared filtering approach; we demonstrate soft shadows, depth of field, and diffuse global illumination, but it could be easily extended to motion blur [Egan et al. 2009].[3] We also support multiple distribution effects [Mehta et al. 2014], as we briefly discuss in the appendix.

[3]We do not include motion blur, since the real-time GPU Optix framework does not natively support raytracing with motion. However, it can be easily implemented within a CPU renderer such as PBRT or Intel's Embree.

Both sheared and axis-aligned filtering are based on a frequency analysis of the light field [Chai et al. 2000; Ramamoorthi and Hanrahan 2001; Durand et al. 2005]. Other recent work in the area includes Fourier depth of field [Soler et al. 2009] and covariance tracing [Belcour et al. 2013] that uses a covariance representation of the 5D space-angle-time light field. In terms of light field reconstruction, Lehtinen et al. [2011, 2012] proposed a reconstruction method for motion and defocus blur from sparse sampling of the 3D/5D (spatial position, lens and time) light field, but with a high memory and computation overhead. In general, these methods are not intended for interactive use, except for axis-aligned filtering that requires higher sample counts. In contrast, we provide accurate results with very low sample counts and interactive frame rates using fast sheared filtering.

Fast Sheared Filtering. We are inspired by Vaidyanathan et al. [2015], who demonstrate a fast sheared filtering approach for defocus blur. This method was later extended by Munkberg et al. [2014] to handle both defocus blur and motion blur at the same time, for which Clarberg and Munkberg [2014] proposed an efficient implementation. However, these methods assume *a fixed filter for a small range of depths*, and therefore require separation of the scene into multiple layers. They use a two-step approach—first project all samples through the center of the lens to neighboring pixels, accumulate per-layer color and alpha, and then do a screen-space convolution (further separated along image axes into two passes). In contrast, we pre-convolve sampled radiance at each pixel individually along the sampling dimension, and then perform a sheared spatially-varying convolution by picking up appropriate pre-convolved samples from neighboring pixels—in a total of 4 steps. Our sheared filter implementation works for multiple distribution effects (soft shadows, defocus blur, diffuse global illumination), with no need for separating the scene into multiple depth planes. Visual comparisons are made in Figure 10. We also analyze the computational complexity of our method, showing how it improves on the basic sheared filtering algorithm.

3. BACKGROUND AND MOTIVATION

In this section, we introduce our notation for the sheared filter, and describe the basic motivation and challenges involving factorization as a solution for fast sheared filtering.

3.1 Basic Notation

We introduce the flatland 2D sheared filter. The next section develops the concept in 3D, and introduces the full 4D filter. We follow notation in previous work [Egan et al. 2011b; Mehta et al. 2012] and discuss our filtering approach for soft shadows first. Let x denote receiver (surface visible at a pixel) coordinate and $y \in [-L, L]$ denote the light coordinate. Very similar parameterization and notation can be used for the lens coordinate for defocus, or incident direction parameterization for global illumination, and important details are mentioned below.

Soft Shadows. Following Mehta et al. [2012], we assume the light has a Gaussian intensity with standard deviation σ_y, and a side length $2L = 4\sigma_y$. For each pixel, we want to simultaneously filter and integrate light visibility and intensity, to compute the overall pixel irradiance. Let $f(x, y)$ be the visibility function and $I(y)$ be the Gaussian light intensity. Then the pixel irradiance is

$$h(x) = \int_{-L}^{L} f(x, y)I(y)dy. \qquad (1)$$

It is shown in [Egan et al. 2011b] that a single occluder plane at distance d_2 from the light, produces a single line of slope given by $s = d_1/d_2 - 1$ in the Fourier spectrum of f, when the receiver pixel is at a distance d_1 from the light source. With multiple occluders, most of the Fourier energy lies between lines of slopes $s_{\min}$ and $s_{\max}$, as shown in Figure 2(a). These bounds can be estimated during the ray-tracing phase. The double-wedge spectrum of f is filtered by the light intensity spectrum on the Ω_y axis, and this bandwidth is $\Omega_y^{\max} = 4/L$. The computation of soft shadows theoretically requires that the receiver's material be diffuse, but in practice moderately glossy receivers also work, as shown by Mehta et al. [2012] and most algorithms based on shadow maps [Hasenfratz et al. 2003; Guennebaud et al. 2006; Annen et al. 2008].

Depth of Field. For rendering depth of field, $x \in [-W, W]$ is measured in pixel space, where W is the width of the image, and $u \in [-A, A]$ is on the lens, where $2A$ is the lens aperture.[4] The light field incident on the camera sensor in (x, u) space has a Fourier transform similar to the area-light visibility f. As shown in Mehta et al. [2014] and Vaidyanathan et al. [2015], a plane at a single depth z produces a line of slope $s = W(F/z - 1)/S$ in the fourier spectrum of the light field, which corresponds to the circle of confusion at that depth. Here F is the focal distance, and S is the size (meters) of the focal plane. Hence, most of the spectrum is bounded between the minimum and maximum circles of confusion, $s_{\min}$ and $s_{\max}$. The bandlimit due to the integration with the Gaussian lens aperture is $\Omega_u^{\max} = 4/A$.

Diffuse Indirect Illumination. To get the double-wedge spectrum for the indirect light field, it must be parameterized in coordinates x along the receiver and v on a plane parallel to the receiver at unit distance. Then a single parallel reflecting surface at distance z from the receiver produces a line of slope $s = z$ in the light field spectrum in the (Ω_x, Ω_v) space. With multiple sloped reflectors, as shown in Mehta et al. [2013], we get a double wedge between slopes $s_{\min}, s_{\max}$. Finally, the double wedge is band-limited by the transfer function of the diffuse BRDF, given by

$$\gamma(v_1, v_2) = \frac{1}{(1 + v_1^2 + v_2^2)^2}. \tag{2}$$

As derived in [Mehta et al. 2013] the bandlimit $\Omega_v^{\max} \approx 2.8$. Note that the bandlimiting function in this case is not a Gaussian, unlike the lens and light functions before. Hence, we do not use Gaussian weights for filtering. Instead, we importance sample along the (v_1, v_2) plane and apply appropriate weights to make it equivalent to cosine-hemisphere sampling, and then use a box filter to filter the samples. This is explained further in Sec. 5.

Sheared Filtering. We now introduce the sheared filter for soft shadows. A similar formalism applies to depth of field and diffuse global illumination, except for a different choice of variables. As indicated in Figure 2(a), the resulting Fourier-domain sheared filter has a shear slope given by the harmonic average of the min and max slopes, and the filter scale is proportional to the difference in the slopes (See [Egan et al. 2011b] for details). We are only concerned with the primal domain filter, as shown in Figure 2(b). The final filtered pixel irradiance $h(x)$ can be obtained as follows, using a 2D

[4]This normalization is chosen to be analogous to the soft shadow example, and is slightly different from Mehta et al. [2014] who normalize the lens coordinate in $[-1, 1]$, and therefore have an extra aperture factor in their formula for circle of confusion.

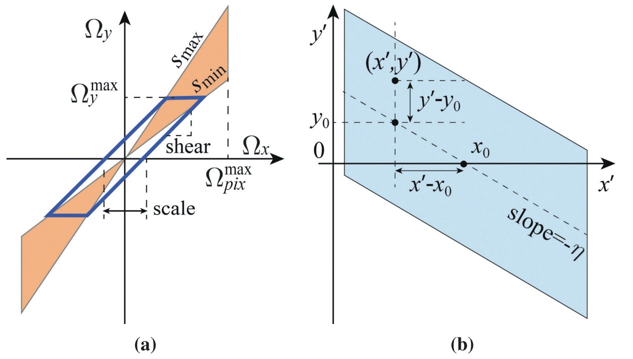


Fig. 2. Illustration of the 2D sheared filter in flatland in (a) Fourier domain and (b) primal domain. The sheared filter in flatland gives the weight of a sample at (x', y') for a pixel of interest x. The filter can be split into two Gaussians: The x-axis Gaussian is fixed with center at x; the y-axis Gaussian has a varying center given by $y = \eta_x(x - x')$.

sheared filter w in flatland:

$$\begin{aligned} h(x) &= \iint f(x', y')w(x', y'; x, y)\,\mathrm{d}x'\,\mathrm{d}y' \\ &= \iint f(x', y')w_x(x' - x; \sigma_x)w_y(y' - y(x, x'); \sigma_y)\,\mathrm{d}x'\,\mathrm{d}y'. \end{aligned} \tag{3}$$

Both $w_x(\cdot)$, $w_y(\cdot)$ are Gaussian functions, with standard deviations σ_x and σ_y respectively. σ_x depends on the sheared filter scale, and σ_y depends on the light bandlimit with

$$\sigma_y = \frac{2}{\Omega_y^{\max}} \qquad \sigma_x = \frac{2}{\Omega_y^{\max}} \frac{s_{\min} s_{\max}}{s_{\max} - s_{\min}}. \tag{4}$$

The filter is a sheared spatially-varying convolution, with the center of the filter along the x' axis determined by the desired location x. The center of the filter along the y' axis is determined by the shear amount η_x, as in Figure 2(b),

$$\begin{aligned} y(x, x') &= \eta_x(x - x') \\ \eta_x &= -\frac{s_{\min} s_{\max}}{2(s_{\min} + s_{\max})}, \end{aligned} \tag{5}$$

while the standard deviation σ_y remains constant and is related to the maximum bandlimit of the light, lens, or sloped reflectors in different applications. Note that we have introduced the auxiliary variable y for the center of the filter along the y' axis.

As an aside, axis-aligned filtering can be thought of as a special case with a shear of $\eta = 0$, so that $y = 0$ always. Then, the overall filter $w_x(x' - x)w_y(y' - y)$ can be separated into two 1D axis-aligned Gaussians, as described in Mehta et al. [2012]. First, they integrate along the y'-axis and store the result for each x'. Second, they filter the 1D result along the x-axis[5] to get the final noise-free image.

3.2 Motivation

The idea is to speed up the integrals for computing the sheared filter, similar to speed-ups obtained by factoring function transforms—for example when converting an image into basis coefficients such as

[5]In practice, they filter the noisy 2D image by separating the filter further along the image axes; thus their overall filter is 1D.

spherical harmonics, fourier or wavelets. In that canonical case, an image of $N \times N$ pixels is transformed into N^2 function coefficients,

$$h(u, v) = \iint f(x, y) w(x, y; u, v)\, dx\, dy, \tag{6}$$

where $f(x, y)$ is the image or function on a 2D domain, $h(u, v)$ are the basis coefficients, and $w(x, y; u, v)$ are the basis functions. A direct implementation has cost $O(N^4)$. However, if the basis is separable along x and y as $w_x(x; u) w_y(y; v)$, we can write

$$h(u, v) = \int \left(\int f(x, y) w_x(x; u) dx \right) w_y(y; v) dy. \tag{7}$$

A two-stage factored algorithm can reduce complexity:

$$\begin{aligned} g(u, y) &= \int f(x, y) w_x(x; u)\, dx \\ h(u, v) &= \int g(u, y) w_y(y; v)\, dy, \end{aligned} \tag{8}$$

where both steps are now $O(N^3)$.[6]

Axis-aligned filtering methods that first integrate samples and then perform image-space convolutions exploit a similar speedup. However, sheared filtering is a slow algorithm because the filter is not separable. Unlike in Equation (7), y is not an independent variable in Equation (5). Hence, we cannot directly separate the dimensions of the sheared filter. Also note that for different x, we have varying η_x values in Equation (5). This prevents us from separately integrating along the y'-axis, because the filter's center y is uncertain.

4. FAST 4D SHEARED FILTERING

We now describe our fast sheared filtering algorithm in its full four-dimensional form. Our key insight is that with an appropriate factorization, the general 4D sheared filter can be made separable into a two-stage 2D integral. By further factoring these 2D integrals into 1D integrals, greater speedups are obtained. We enable interactive frame rates, with overhead not significantly different from axis-aligned filtering, but with much lower sample counts. Table I gives the computational complexity of the various steps.

4.1 4D Sheared Filtering

Consider the 4D form of Equation (3), with both x and y split into two dimensions each. The sheared filtering integral becomes

$$\begin{aligned} h(x_1, x_2) = \iiiint & f(x_1', x_2', y_1', y_2') w_x(x_1' - x_1) w_x(x_2' - x_2) \\ & w_y(y_1' - y_1(x_1, x_1')) w_y(y_2' - y_2(x_2, x_2'))\, \mathrm{d}x_1' \mathrm{d}y_1' \mathrm{d}x_2' \mathrm{d}y_2', \end{aligned} \tag{9}$$

where we have omitted the standard deviations for clarity.

Following the definitions in flatland, here (x_1, x_2) represent receiver space coordinates (pixel coordinates) and (y_1, y_2) represents the light space coordinate. The $w_x(x_1' - x_1)$, $w_x(x_2' - x_2)$ are (spatially-varying) convolutions for each pixel (x_1, x_2), while $w_y(y_1' - y_1)$, $w_y(y_2' - y_2)$ are integrals which eliminate y_1' and y_2'.

Similar to Egan et al. [2011b], we require that the area light is parameterized with orthogonal basis vectors, guaranteeing that

[6]Further speed-ups may of course be obtained by a Fast Fourier Transform or an in-place wavelet transform, but are not immediately relevant to the sheared Gaussian filters used in this article.

Table I.
Computational complexity (per-pixel) and input/output dimensions of various methods. The integral dim. column is the number of dimensions we integrate over. Bold is the overall complexity (most expensive step for the factored algorithms). Here n is the linear filter size, and l^2 is the number of samples per pixel.

Method	Input dim.	Output dim.	Integral dim.	Complexity
4D sheared filtering	4	2	4	$\mathbf{O(n^2 l^2)}$
Axis-aligned filtering	2	2	2	$\mathbf{O(n)}$
2D factoring, Step 1	4	3	2	$\mathbf{O(nl^2)}$
2D factoring, Step 2	3	2	2	$O(nl)$
Our Method, Step 1a	4	4	1	$O(l^3)$
Our Method, Step 1b	4	3	1	$\mathbf{O(nl)}$
Our Method, Step 2a	3	3	1	$O(l^2)$
Our Method, Step 2b	3	2	1	$O(l)$

(x_1, y_1) and (x_2, y_2) span orthogonal 2D subspaces of the 4D light field. For the simplicity of derivation and implementation, x_1 and y_1 are arranged as parallel, and the same for x_2 and y_2.

Solving this 4D sheared filtering integral efficiently is a long-standing problem. There are two main challenges. First, the convolution center on the y-plane (y_1, y_2) is determined by the relative deviation on the x-plane, or $(x_1' - x_1, x_2' - x_2)$. This indicates that y_1 and y_2 are functions of x_1' and x_2' respectively, making the filter nonseparable between x and y. Second, the visibility function f is sampled over the entire 4D space. Unlike the 2D x-plane, which is regularly divided as a pixel grid over the output image, the 2D y-plane is continuous, over which different pixels (x_1, x_2) could (and should) sample at different locations. This means separating samples on y_1 and y_2 is difficult. Hence, neither the filters nor the samples can be easily separated.

To analyze the computational complexity, we define the image resolution in (x_1, x_2) as $N \times N = O(N^2)$, where a typical $N \sim 1000$. We define the extent of the sheared filter in $w(x - x')$, corresponding to the integrals in x_1' and x_2' as n, where we use $n \leq 32$. The number of light samples along y_1' or y_2' is small and can almost be taken as a constant, since sheared filtering works with very low sample counts. We define this as l, where typically $l \leq 4$.

As shown in Table I, the input dimensionality (of f) is 4D, and the output is a 2D image. The computational complexity is $O(N^2)$ for the output, and $O(n^2 l^2)$ for the integral for each pixel. The effective complexity is thus $O(N^2 n^2 l^2)$, or $O(n^2 l^2)$ per pixel. In contrast, the spatially-varying image-space convolutions in axis-aligned filtering can be performed in $O(N^2 n)$ time or $O(n)$ per pixel, using image-space separable Gaussian filters. We will show that our final separated sheared filtering algorithm has only slightly higher $O(N^2 nl)$ complexity or $O(nl)$ per pixel instead of $O(n^2 l^2)$.

4.2 Separating into Two 2D Integrals

The last section showed that neither the filters nor the samples are easily separable, which would suggest that accelerating the 4D sheared filter is very difficult. Our key insight is that while separating (x_1, x_2) and (y_1, y_2) dimensions directly is not possible, we can try to separate the (x_1, y_1) and (x_2, y_2) dimensions as shown in Figure 3. In this section, we show how the 4D sheared filter becomes a product of two independent 2D sheared filters applied over the (x_1, y_1) and (x_2, y_2) planes respectively, and we develop a two step factored algorithm. We keep the samples and reduce their dimension at each

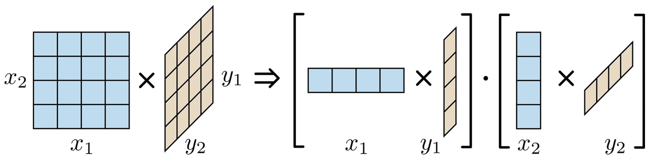


Fig. 3. Separating a 4D sheared filter into a product of two independent 2D sheared filters in (x_1, y_1) and (x_2, y_2). Note that, as discussed in Section 4.1, separating a 4D sheared filter into 2D filters over x-plane and y-plane respectively is more intuitive, but not feasible in theory.

filtering step, eliminating the dependency between y_1 and y_2 by integrating them one by one.

Step 1. We first apply one sheared filter in the $x_1 y_1$ plane, effectively evaluating the inner two integrals in Equation (9). Since y_1 and x_1' are related by Equation (5), we also remove y_1. We denote the three dimensional filtered integral of f as g:

$$g(x_1, x_2', y_2') = \iint f(x_1', x_2', y_1', y_2') w_x(x_1' - x_1) w_y(y_1' - y_1(x_1, x_1'))\, \mathrm{d}x_1' \mathrm{d}y_1'. \quad (10)$$

Once again, we omit the standard deviations on the Gaussians for clarity. The complexity of this step is $O(N^2 n l^2)$, since g needs to be evaluated at $O(N^2 l)$ points, and the integral has complexity $O(nl)$. The per-pixel cost is thus reduced from $O(n^2 l^2)$ to $O(nl^2)$.

Step 2. Once g is computed, we apply the second sheared filter. We integrate along x_2' and y_2' to determine the final pixel irradiance $h(x_1, x_2)$. Similarly, y_2 is eliminated since it is determined by the value of $x_2' - x_2$,:

$$h(x_1, x_2) = \iint g(x_1, x_2', y_2') w_x(x_2' - x_2) w_y(y_2' - y_2(x_2, x_2'))\, \mathrm{d}x_2' \mathrm{d}y_2' \quad (11)$$

The complexity of this step is $O(N^2 nl)$ or $O(nl)$ per pixel. It is less than step 1, since we only need to produce a 2D output, and the dimensionality of g is already less than that of f.

In theory, the separation of the 4D filter into a product of two 2D filters in Equations (10) and (11) is exact only when the filter kernels remain constant over the image plane. However, similar to separating a 2D box filter over an image into a two-pass orthogonal linear filter, the inaccuracy when this approximation is violated is usually negligible in practice. We will evaluate our approximation against brute force 4D sheared filtering in Section 6, and limitations are shown in Figure 11.

4.3 Separating into 1D Integrals

While the separation into two 2D integrals provides savings, the first step in Equation (10) is still expensive, with complexity $O(nl^2)$ per pixel. We derive a further factorization into 1D integrals, with a two-step computation of each 2D step. Both 2D filters have a sheared shape in the xy plane. The basic idea is to separate the sheared filter's shape into a pre-convolution and a collection as in Figure 4.

Step 1a. To compute g in Step 1 efficiently, we first perform a pre-convolution for each y_1 (outer integral in Equation (10)), to produce an intermediate result p:

$$p(x_1', x_2', y_1, y_2') = \int f(x_1', x_2', y_1', y_2') w_y(y_1' - y_1)\, \mathrm{d}y_1'. \quad (12)$$

There are several important points to note here. Unlike elsewhere in this section, y_1 is an independent variable, and p is precomputed

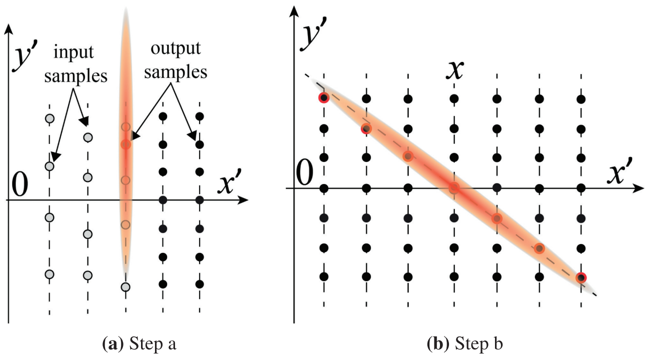


Fig. 4. We first separate the 4D sheared filter into two 2D sheared filters, and then evaluate each 2D filter in two 1D integration steps. As shown in (a), we first convolve along the y-axis and compute a y-dependent function (we show the visibility samples with open circles, and reconstruction locations with filled black circles, and the red circle shows an example convolution). Then in (b), we convolve along the x-axis to remove the y-dependence, effectively collecting pre-convolved samples along the shear direction, using the nearest neighbor pre-convolved values (shown in red).

(pre-convolved) for each y_1 in preparation for step 1b, where y_1 will be expressed as usual in terms of x_1 and x_1'. p is calculated for each pixel (x_1', x_2') and each value of y_2'.

Since y_1 is a continuous parameter, we discretize (stratify) the range of y_1 into $O(l)$ bins.[7] In practice, accurate reconstruction requires about $4l$ bins. Since $l \leq 4$ in our case, we use 16 bins.

Note that p needs to be stored at $O(N^2 l^2)$ points, and the cost of the integral is $O(l)$, so that the total complexity is $O(N^2 l^3)$ or $O(l^3)$ per pixel. While this is still cubic, note that $l \ll n$ is a small constant (typically $l \leq 4$), and this cost is generally less than the $O(nl^2)$ per-pixel complexity of Equation (10).[8] In practice, step 1a is not even the most expensive in our implementation.

Step 1b. After this pre-convolution step, we can finally compute g by applying the filter in the x-dimension (inner integral in Equation (10) as follows:

$$g(x_1, x_2', y_2') = \int p(x_1', x_2', y_1(x_1, x_1'), y_2') w_x(x_1' - x_1)\, \mathrm{d}x_1'. \quad (13)$$

This is a 1D integral that does a "gather" around x_1. Since p is already computed, it can be quickly queried. Note that the parameter y_1 on right, is a function of $x_1' - x_1$ as usual, and is also integrated out in this step. Since we still need to filter and integrate g along y_2', we also discretize y_2' into $O(l)$ bins, similar to y_1.[9]

The complexity of this step is $O(N^2 nl)$ since we need to store g at $O(N^2 l)$ values, and the integral has complexity $O(n)$. The per-pixel cost is thus $O(l^3 + nl)$ from combining steps 1a and 1b. In practice, the sheared filter size n is about 32 pixels, while the number of samples on the light l^2 is usually about 16. Hence we have $n > l^2$ and step 1b dominates, with the net per-pixel complexity

[7]In practice, we use uniform jittered sampling so the samples for a given pixel are offset the same way in each stratum, but this is not critical for our method as long as stratified sampling is used.

[8]Just as in practice we must use $\approx 4l$ bins, we need a similar number of bins for storing y_2' in Equation (10), since the jitter offsets for different pixels are different for the next step.

[9]Instead of enumerating each possible y_2' value and searching for feasible p samples, we instead use an inverse method by projecting different y_2' values of the p samples onto the discretized y_2' space.

being $O(nl)$. Note that this is the square root of the original $O(n^2l^2)$ complexity in Equation (9).

Step 2a. As for step 1, we separate the sheared filter in step 2 into two 1D filters. Similar to step 1a, we first pre-filter the result of step 1b, and determine the result q for every possible y_2:

$$q(x_1, x_2', y_2) = \int g(x_1, x_2', y_2')w_y(y_2' - y_2)\,\mathrm{d}y_2'. \tag{14}$$

Step 2b. Finally, we integrate on x_2' while quickly querying q,

$$h(x_1, x_2) = \int q(x_1, x_2', y_2(x_2, x_2'))w_x(x_2' - x_2)\,\mathrm{d}x_2'. \tag{15}$$

This last step also integrates out the y_2 dependence, because y_2 depends in the usual way on $x_2' - x_2$. Finally, we have evaluated the 4D integral of h in Equation (9) using four 1D filters.

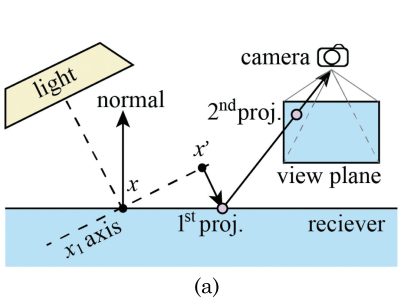
The complexity of step 2a is $O(N^2l^2)$ and of step 2b is $O(N^2n)$. Since $n > l^2$, the last step dominates and the net per-pixel cost is $O(n)$ per pixel, as compared to $O(nl)$ for Equation (11). Step 1b is the overall dominant cost, which is $O(nl)$ per pixel. Since l is a small constant, this is effectively $O(n)$ per pixel, and the computational complexity is comparable to (but with higher constants than) axis-aligned filtering.

5. IMPLEMENTATION

Our implementation involves two basic components, as in most previous work: Sampling by ray or path tracing to obtain the original noisy samples of $f(\cdot)$, and filtering or reconstruction by fast sheared filtering. We implement our algorithm using OptiX 3.0 and CUDA 5.0 inter-operation. We use OptiX to do the sampling step, and we store the result in OptiX buffers. Then we use four sequential CUDA pixel shader passes to perform our four-pass filtering. We will release the source code online upon publication. Note that while we have described filtering using integrals above, these map almost directly to discrete summations over a grid of points. We now discuss a few important details of the implementation.

Ray Tracing and Sampling. To reduce memory footprint while avoiding banding artifacts, we use uniform jittered sampling [Ramamoorthi et al. 2012], so that a given pixel's samples have the same random offset on a regular stratifed sampling grid. This makes it easier for the filtering steps, that can now operate on a regular grid, as well as in reducing memory in storing samples for intermediate stages. For soft shadows and depth of field, we sample uniformly on the light and lens respectively (and use Gaussian filter weights which account for the Gaussian intensity and aperture respectively). For a given pixel, we store the jitter value ($\in [0, 1]^2$), and each sample's visibility or radiance, that is, discrete samples of the function $f(\cdot)$ for filtering. These samples also determine the pixel's frequency information, and ultimately the slope bounds for the sheared filter, $s_{\min}$ and $s_{\max}$. In contrast to previous work on axis-aligned filtering, we use only a single (nonadaptive) sampling pass, since our sampling rates (4 to 16 samples per pixel) are so low; these samples determine both $f(\cdot)$ and the sheared filter.

Filtering. Our filtering algorithm was described in the previous section. We clamp the maximum filtering range to a diameter of 32 pixels; this prevents rapid changes in rendering speed when the view is changed. The number of discrete bins used in filtering steps 1a and 1b for storing y values is $16 \approx 4l$. In steps 1b and 2b, it is required that we filter along x_1, x_2 directions exactly—these are given by the projection of the light's axes y_1, y_2 on the receiver for soft shadows. So, we first compute each pixel's x_1 and x_2 in the world coordinate frame, sample along each direction, then project the sampled point back to the screen as shown in Figure 5(a). For indirect illumination, the filtering directions x_1, x_2 are orthogonal in world space to the receiver normal, and locally aligned. For depth of field x_1, x_2 are exactly along the screen's row and column axes, so the filtering algorithm can be applied directly.



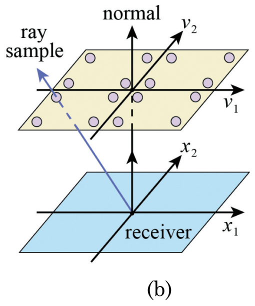


Fig. 5. (a) Since soft shadows must be filtered along x_1, x_2 axes as defined by the light source, we first determine these axes in screen space, by projection from the light to the receiver and then to image space. (b) We sample indirect illumination along the v-plane instead of the usual cosine-hemisphere sampling.

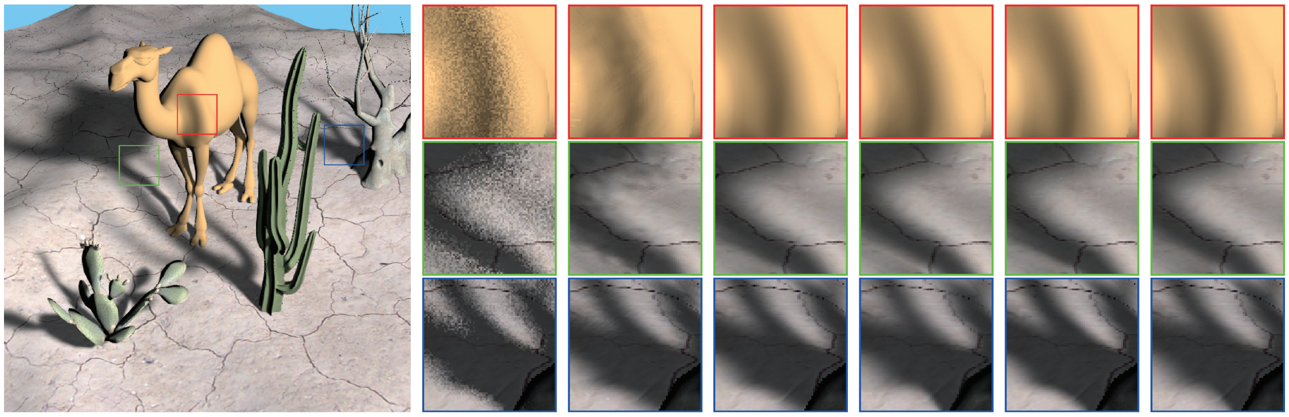
Slope Smoothing for Soft Shadows. After the initial sampling, many pixels on the edges of shadows do not have valid $s_{\min}, s_{\max}$ values (if none of the samples hit occluders), which causes edge artifacts. Hence, similar to Mehta et al. [2012], we obtain the slope range for unoccluded pixels by smoothing over a 5×5 window. The (small) time for this operation is shown as pre-filtering in Table II, and included in the total overhead of our algorithm.

Adaptive Sampling for Depth of Field. For depth of field, for some pixels, the slope bounds $s_{\max}$ and $s_{\min}$ could be of opposite signs. In such cases, the shear value can be close to zero, and the filtering is inaccurate. As in [Vaidyanathan et al. 2015], we find that using axis-aligned filtering gives better results for pixels with $s_{\max} \cdot s_{\min} < 0$. These pixels usually need more samples even with filtering, so we trace a fixed 36 more samples for such pixels in a second sampling pass (and then do standard axis-aligned filtering—this small post-filtering time is reported as part of the overhead in Table II). In most cases, the fraction of the image that requires further sampling is very small. For the scene of Figure 7, the first pass requires 9 spp and the average from the second pass is 2.2 spp.

Sampling/Filtering for Indirect Illumination. As described in Section 3.1, indirect illumination must be filtered in v-space, and hence we also sample on the v plane. For a single pixel, we map a uniform jittered sample with a cubic function and a scaling, to approximate the nonuniform PDF of the diffuse transfer function $\gamma(v_1, v_2)$ on $(v_1, v_2) \in [-5, 5]^2$, and then compute the ray direction, as shown in Figure 5(b). Theoretically, the range of (v_1, v_2) is the infinite plane, but our truncation contains over 99% of the total energy of $\gamma(\cdot)$ and only introduces a very small bias. Note that we apply the BRDF weight to each sample (v_1, v_2) and also account for the sampling PDF. Finally while filtering in steps 1a and 2a, we use a box function instead of a Gaussian as for soft shadows and depth of field, since the BRDF transfer function is no longer a Gaussian. Steps 1b and 2b (spatial filtering) can still use Gaussian weights.

6. RESULTS

Our results are produced on an Intel 6-core 3.6GHz i7-4960X CPU, with a NVIDIA GTX Titan video card. We show results for interactive soft shadows in Figures 1(a) and 6; depth of field in



(a) Soft Shadows (Ours), 9 spp, 12.2 fps **(b)** MC Input 9 spp, 0.05s **(c)** AAF (ET) 13 spp, 0.08s **(d)** AAF (EQ) 46 spp, 0.30s **(e)** Ours 9 spp, 0.08s **(f)** Gr. truth 1K spp, 3.94s **(g)** 4D SHF 9 spp, 2.92s

Fig. 6. The CAMEL scene with soft shadows, rendered at 12.2 fps with 9 samples per pixel (spp), demonstrates our ability to accurately reconstruct overlapping and thin-occluder shadows. Comparisons show (b) noisy unfiltered MC input to our method (note that we store individual samples for sheared filtering), (c) Equal time (ET) Axis-aligned filtering (AAF) retains some low-frequency noise, and overblurs sharp shadow edges (since we use $\mu < 1$) (d) Equal time (EQ) AAF is 4× slower, while (g) simple 4D sheared filtering is 50× slower.

Table II.
Detailed timings of our scenes (in milliseconds) rendered at 720 × 720. Cars and Camel show soft shadows, Pool and Still Life are depth of field, Room and Sibenik are diffuse global illumination. We list triangles and samples per pixel for all six scenes (Pool uses spheres rather than triangles). We also list the per-frame sampling time for raytracing in Optix, followed by timings for various stages or our algorithm, and the total overhead for fast sheared filtering. For comparison, we also list the total overhead for axis-aligned filtering. Finally, we list the total time and frame rates. We achieve interactive frame rates of 3-12 fps on a variety of complex scenes.

			Sampling	Our fast GPU sheared filtering algorithm						AAF	Total	
Scene	Tris	spp	Optix (ms)	Pre/Post filt.(ms)	Step 1a (ms)	Step 1b (ms)	Step 2a (ms)	Step 2b (ms)	overhead (ms)	AAF (ms)	time (ms)	fps
CARS	4 K	8	85.2	8.2	4.0	28.6	11.4	16.2	**68.4**	32.0	153.6	**6.5**
CAMEL	43 K	9	48.0	4.3	2.0	14.4	5.7	7.6	**34.0**	16.0	82.0	**12.2**
POOL	-	11.0	69.5	1.2	4.8	25.8	11.3	20.6	**63.7**	13.0	133.2	**7.5**
STILL LIFE	233 K	11.2	146.0	2.0	6.4	44.0	11.1	22.4	**85.9**	18.0	231.9	**4.3**
ROOM	163 K	16	310.0	-	7.5	47.9	11.5	23.8	**90.7**	75.0	400.7	**2.5**
SIBENIK	75 K	16	225.2	-	7.5	40.8	11.5	23.0	**82.8**	72.5	310.0	**3.2**

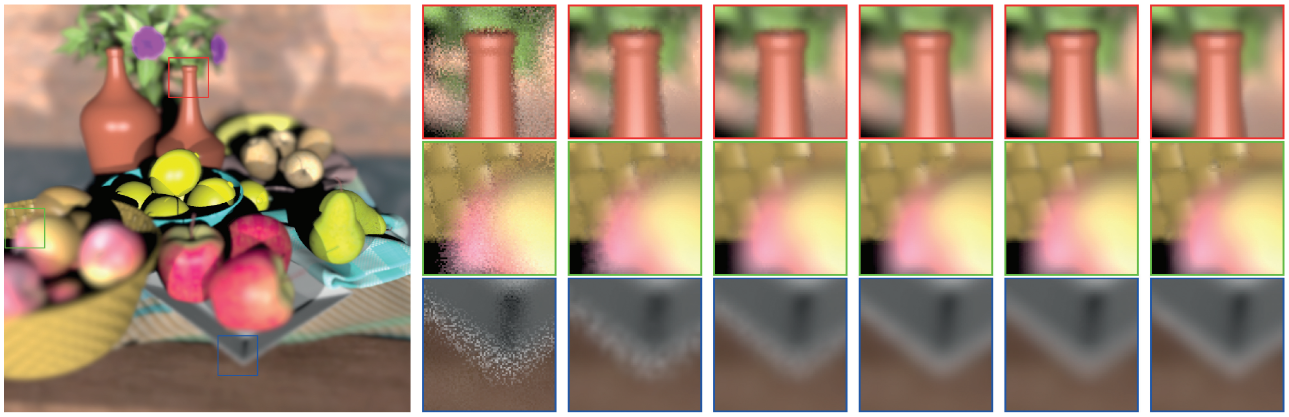
Figures 1(b) and 7; and diffuse indirect illumination in Figures 1(c) and 8. We compare to stratified Monte Carlo sampling without filtering, unaccelerated 4D sheared filtering [Egan et al. 2011b], and to axis-aligned filtering [Mehta et al. 2012, 2013]. The accompanying video shows animations and screen captures with moving light source, viewpoint and some examples of dynamic geometry. We require no precomputation except the ray-tracer BVH, and each frame is rendered independently.

6.1 Accuracy and Speedup over Monte Carlo

The accuracy of our method, and the benefit of filtering over stratified Monte Carlo is evident from the figures, for all three visual effects (soft shadows, defocus, diffuse global illumination) in a number of different situations. We take as input a Monte Carlo result with 4-9 average samples per pixel for depth of field and soft shadows, and 16 samples per pixel for indirect illumination. As shown in the insets of Figure 1, and Figures 6(b)–8(b), this input is very noisy, but our fast sheared filtering technique produces visually accurate results compared to ground truth Monte Carlo with 1024-4096 samples, which is 100-200× slower. A quantitative comparison is in the graph of Figure 9. While the quantitative errors are somewhat higher than from a visual comparison, our method converges with more samples, and for equal RMS error, reduces the number of samples needed by over an order of magnitude compared to Monte Carlo, and about 6× relative to axis-aligned filtering.

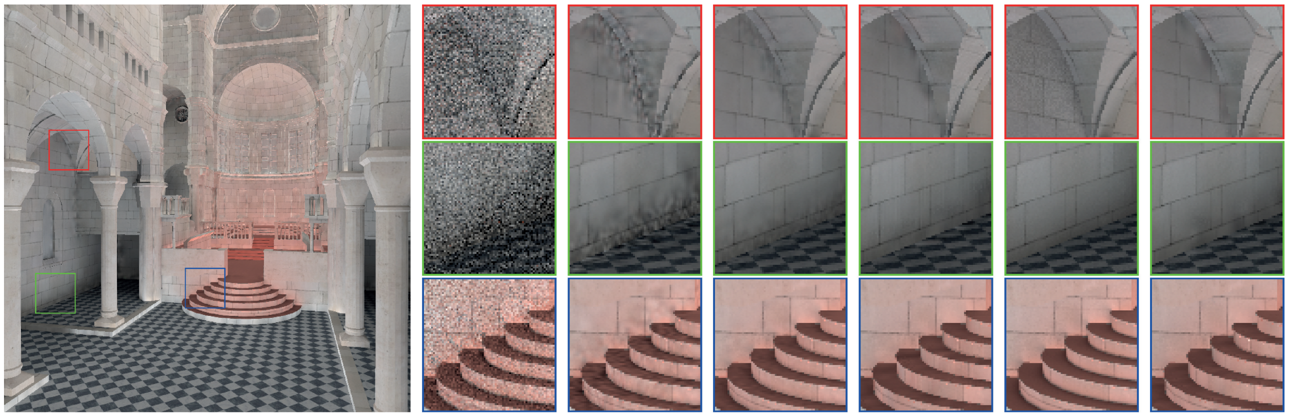
6.2 Timings

In Table II, we show timings for steps of our algorithm on different scenes, rendered at a resolution of 720 × 720. The CARS scene in Figure 1 has 4K triangles with soft shadows from two area lights, each sampled with 4 samples per pixel (total 8 spp). The CAMEL is a more complex example of soft shadows, rendered with 9 spp for the light. The POOL (which uses spheres, rather than triangles as primitives) and STILL LIFE show depth of field effects, while ROOM and SIBENIK are complex scenes with over 10^5 triangles, that demonstrate diffuse indirect illumination with 16 spp. The ray-tracing time using OptiX varies with scene complexity, from 39ms for POOL to 310ms for ROOM. The total overhead of our method is about 40–60ms for soft shadows and depth of field, and about 80ms for indirect illumination. This is less than the cost of raytrac-



(a) Defocus Blur (Ours), 11.2 spp, 4.3 fps (b) MC Input 11.2 spp, 0.16s (c) AAF (ET) 15 spp, 0.22s (d) AAF (EQ) 94 spp, 1.43s (e) Ours 11.2 spp, 0.23s (f) Gr. truth 1K spp, 19.2s (g) 4D SHF 15 spp, 12.2s

Fig. 7. The STILL LIFE with depth of field is illuminated by two point lights and rendered at 4.3 fps with only 11.2 average samples per pixel (spp). Comparisons show (b) noisy unfiltered MC input to our method, (c) Equal time (ET) Axis-aligned filtering (AAF) retains some low-frequency noise, (d) Equal quality (EQ) AAF is 6× slower, while (g) simple 4D sheared filtering is 45× slower. Our method (and 4D SHF) produces slight overblur for background regions and underblur for foreground ones, and transition artifacts near the focal plane where AAF and our method switch. However, our method reduces most noise and requires the least number of samples.



(a) Diffuse Global Illum. (Ours), 16 spp, 3.2 fps (b) MC Input 16 spp, 0.23s (c) AAF (ET) 19 spp, 0.32s (d) AAF (EQ) 94 spp, 1.39s (e) Ours 16 spp, 0.31s (f) Gr. truth 4K spp, 128.2s (g) 4D SHF 16 spp, 7.1s

Fig. 8. The SIBENIK scene showing only 1-bounce diffuse indirect lighting with one point light, rendered with only 16 samples per pixel at 3.2 fps. Monte Carlo input in (b) is noisy, while equal time axis-aligned filtering in (c) has artifacts at this low sample count. Equal quality AAF in (d) requires 6× as many samples as our method in (e), and is 4× slower. While there are a few artifacts remaining, our method significantly reduces noise to the level of ground truth in (f) and simple 4D sheared filtering in (g), but is more than an order of magnitude faster.

ing in most cases, and about 25% of the total cost for the more complex ROOM and SIBENIK scenes, resulting in only a modest decrease in the overall performance of the real-time raytracer. Step 1b, involving the gather operation is the most expensive step, as discussed in the text, accounting for about half the total overhead.

Compared to axis-aligned filtering (AAF), our overhead is about 2× as much for soft shadows, 4× for depth of field, and only 20% more for diffuse indirect illumination. While equal time AAF can use slightly more samples (and adaptively sample), the many fewer samples needed by our method provides a net win of 4× in wall clock time and about $5 - 6\times$ sample count reduction for equal quality, as shown in the figures. Our algorithm does require more memory compared to the axis-aligned approach, since we need to store intermediate results. However, since we usually need low sampling rates, GPU memory is usually adequate. Even for visually indistinguishable convergence, we usually need no more than 25 samples per pixel for soft shadows and depth of field effects, and 49 samples per pixel for global illumination. We achieve interactive frame rates of 3-12fps for a wide range of scenes.

6.3 Comparisons

Axis-aligned filtering. We use the authors' code for soft shadows and indirect illumination, and implement depth of field analogously. Their user-specified parameter μ is used to control filter size and adaptive sampling rate. We use $\mu = 1$ for equal quality comparisons in all six of our scene figures, and this requires about $5 - 6\times$ the sampling rate of our method. For equal-time comparisons in

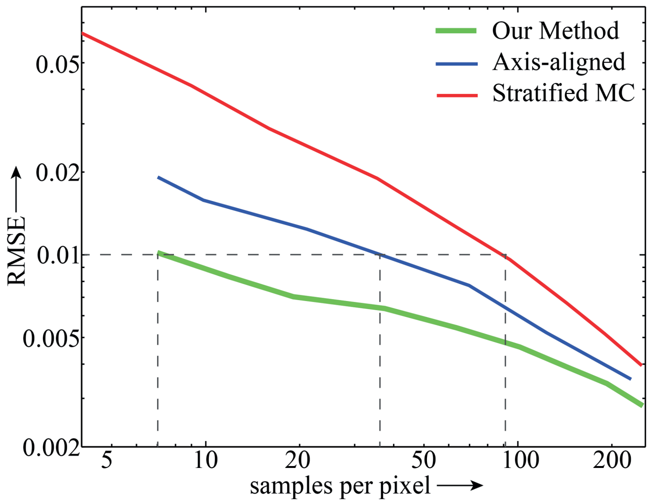


Fig. 9. RMS error of the CAMEL scene as a function of sampling rate for our method, unfiltered stratified Monte Carlo and axis-aligned filtering with adaptive sampling. At very low sample counts, we obtain an overall benefit of more than an order of magnitude over Monte Carlo (even better visually), and around 4× over axis-aligned filtering. Moreover, our method converges to ground truth, and is always more accurate than axis-aligned filtering.

Figures 6(c), 7(c), 8(c), we need to use $\mu < 1$ to reduce the sampling rate. This increases the filter sizes, and slightly overblurs the image. Low-frequency noise is also retained in high-variance regions. For diffuse global illumination, much higher sample counts are needed, and the equal time comparison in Figure 8(c) shows artifacts. Our method performs better; While inaccuracies exist in out-of-focus regions and artifacts can be seen around discontinuous geometry in soft shadows and diffuse global illumination situations, our method filters out most visible noise with significantly fewer samples.

Layered light field reconstruction. We use the source code of LLFR [Vaidyanathan et al. 2015] to make comparisons with our method both in terms of quality and speed for filtering depth-of-field images. We use the GPU implementation of LLFR, with their default filter width of $n = 16$ pixels, rather than $n = 32$ as used in our results. Since the LLFR code takes as input only uniformly sampled light fields, we disable adaptive sampling and use a constant 9 spp for both depth-of-field scenes compared. LLFR requires segmentation of the scene into depth layers, where the same filter can be applied within a layer. Our method makes no such assumptions. As shown in Figure 10, LLFR produces more noise in some regions, because the sheared spectrum it is using is not as compact as ours due to layering. However, LLFR has a slightly lower reconstruction time than our method, which is partly due to their use of half-precision floating point numbers for all stored data.

4D sheared filtering. Our method is equally accurate as the full 4D brute-force sheared filter of [Egan et al. 2011b] (compare Figures 6, 7, 8 (c) and (g)). We implemented the simple 4D sheared filter without any of our factorizations in CUDA, so we could compare using the same framework as our method. We used a single filtering pass for the 4D sheared filter, which accumulates radiance from all samples in the neighborhood of a pixel. As seen in the figures, this is about 40 − 50× slower than our approach. For all the depth of field results using 4D sheared filtering, adaptive sampling is used

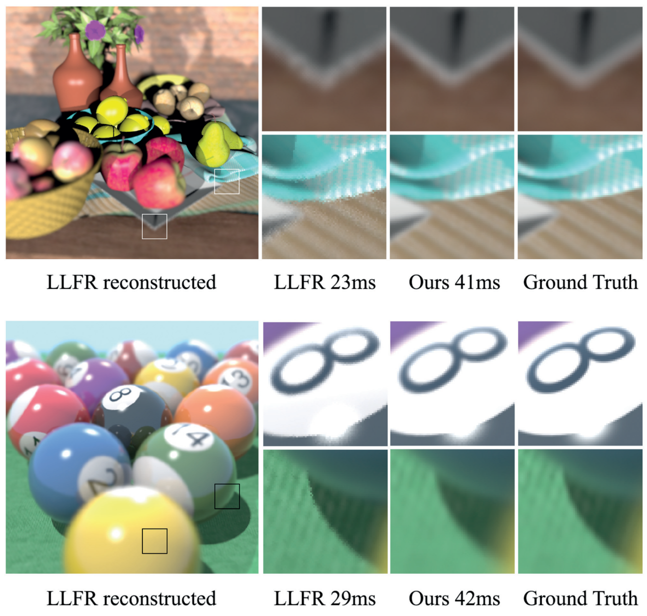


Fig. 10. We compare insets of the STILL LIFE scene (top row) and the POOL scene (bottom row) to LLFR [Vaidyanathan et al. 2015], with a uniform 9 spp for both scenes and 12 layers for LLFR (the default). Overall, LLFR has good quality and takes less reconstruction time than our method. However, it usually produces more visible noise, as mentioned in Sec. 6.3, and is limited to depth of field only. Our method makes no assumptions about depth layers and is general enough for soft shadows and indirect illumination as well.

according to the general sampling rate formula for sheared filters derived in Egan et al. [2009]. At in-focus regions, since the shape of the spectrum is no longer a double wedge, 4D sheared filtering falls back to brute force Monte Carlo as described in Egan et al. [2009].

7. DISCUSSIONS AND LIMITATIONS

Complexity. The actual complexity of our four-step filtering should be $O(l^3) + O(nl) + O(l^2) + O(n)$. Clearly, steps 1a and 1b have cost $O(l^3 + nl)$ which is l more than steps 2a and 2b. In practice, $l^2 < n$, since l is typically 4 or less and n is 32. Therefore, step 1b dominates. Furthermore, since the $O(l^3)$ step 1a happens within each pixel, an efficient pixel-level parallelized implemention is used to avoid much of the overhead caused by conflicts in pixel-access. Hence, the $O(l^3)$ term can be absorbed into the numerical constant, and the total complexity would be $O(nl)$ per pixel.

Multiple Effects. Our method currently focuses on seperated single effects. Simultaneously handling multiple effects [Mehta et al. 2014; Munkberg et al. 2014], including soft shadows, depth of field and motion blur, requires filtering higher dimensional (often 5D or 6D) data, while the spectrum of each slice of the data for a single effect still remains a sheared shape. This can be difficult, since it further increases dimensionality and nonseparability of samples for different effects. However, by introducing a reasonable approximation, we demonstrate that multiple effects could be separated into a combination of several individual effects. Please refer to the Appendix for detailed derivations and results.

Limitations. For soft shadows, minor artifacts could emerge due to projections of world space receivers to screen space, when the

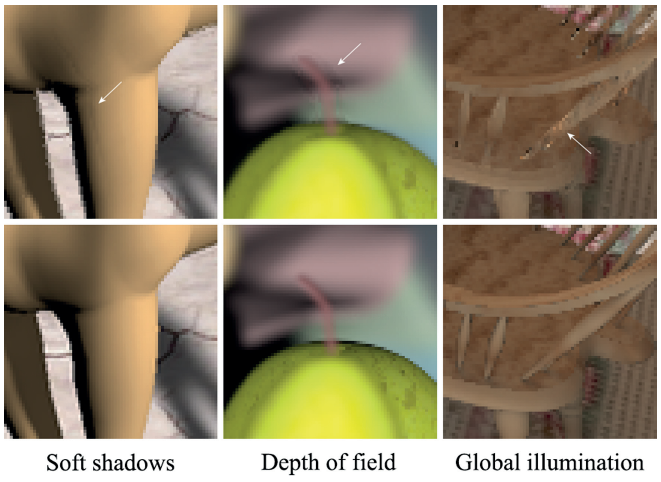


Fig. 11. Comparisons of our method (top row) and the ground truth (bottom row) in difficult regions. Our method produces minor artifacts in certain regions as pointed out and discussed in Section 7.

receiver becomes almost perpendicular to the viewing direction (projection from receiver to screen space collapses to a single point), or when the light becomes normal to the receiver (the x_1, x_2 axes become parallel). For depth of field, inaccuracies may result when the slope range at a pixel is large. In diffuse indirect lighting, a small bias is introduced in sampling due to using truncated v-plane sampling, but the bias is usually not perceivable.

When a large filter is applied to locations where the filter sizes vary rapidly, inaccuracies could occur since our method uses separable passes. This is often encountered when filtering near in-focus regions for depth of field effects, resulting in slight overblur around these regions. We do not filter between neighboring pixels if they are distant in world space, or if they have very different normals (angle threshold 20°). Pixels at which many such neighboring pixels are rejected may retain noise or artifacts. Figure 11 shows some of these difficult regions and points out the artifacts.

In depth of field rendering, our method falls back to axis-aligned filtering in regions where $s_{\max} \cdot s_{\min} < 0$. Therefore, ghosting artifacts may appear on the boundary where this switch occurs, for example, around the stem of the pear in the Still Life Scene (Figure 11 middle) and the topmost apple. These transitions and ghosting artifacts also occur in the original 4D sheared filtering, since the problem is not related to separability.

Our method also suffers from the general problem of sheared filtering — noisy occluding geometry. Since we are using a fairly low number of sample rays, the occluding geometry may not be accurately captured — thus the shape of the sheared filter itself could be inaccurate and noisy. This will consequently lead to flickering between frames. However, video comparisons show that our method still performs better than previous methods, even when the previous approaches use many more samples and more time.

Similar to [Egan et al. 2009], our method needs to store and filter the entire 4D light field $f(x_1, x_2, y_1, y_2)$, which is of complexity $O(N^2 l^2)$. This introduces significant storage overhead, practically limiting our method to about 25 spp for a resolution of $2K \times 2K$ (for our specific hardware configuration). For HD applications, a block-wise sampling and filtering configuration could be derived from our work with an expected but small additional performance cost for inter-block operations. We leave this for future work.

8. CONCLUSIONS AND FUTURE WORK

We demonstrate an interactive GPU-based method of sheared filtering for Monte-Carlo rendering of distribution effects. We propose a novel factorization of the 4D sheared filter into two 2D filters, and we further split each 2D filter into two 1D filters. We also derive a complexity analysis for our method, and compare it to axis-aligned filtering. Our results show soft shadows, depth of field and diffuse global illumination at interactive speeds for complex scenes, and we are 4× faster than axis-aligned filtering for the same quality, with a $5 - 6\times$ reduction in sample count.

In future work, we plan to extend our range of applications to environment lighting or spherical harmonic occlusion [Egan et al. 2011a], and generalize our method for filtering Monte Carlo images involving higher dimensional integrals, including simultaneous primary and secondary distribution effects [Mehta et al. 2014; Munkberg et al. 2014]; initial results are shown in the appendix. A simple extension for adaptive sampling could also be considered.

Sparse sampling, followed by sophisticated filtering and reconstruction, has emerged as an important method to dramatically speed up Monte Carlo rendering. However, the slow performance of methods like sheared filtering have limited the performance gains and interactivity. We have taken an important step towards real-time physically accurate rendering, by developing the first factored GPU 4D sheared filtering method, and expect many future developments that enable both low sampling rates and high performance.

APPENDIX: MULTIPLE EFFECTS

In this section, we focus on how to efficiently separate multiple effects (soft shadows, depth of field and diffuse global illumination) into independent single effects. Similar to Mehta et al. [2014], we observe that soft shadows and diffuse global illumination represent direct and indirect lighting respectively, and they could therefore be naturally separated. Thus, without loss of generality, we demonstrate the derivation of our separation scheme only for the combination of soft shadows and depth of field effects. Figure 12(e) shows a separate pass for the filtered indirect illumination (using the algorithm in the main text), that is added to the final result in Figure 12(a).

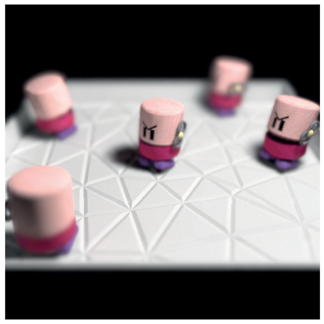
We refer to a simplified notation, using x as (2D) screen coordinate, u as lens coordinate, and y as light coordinate. Intuitively, the pixel radiance due to direct illumination is a two-step integral. The first or inner step filters out the correct outgoing radiance due to the area light, and the second or outer step filters for the lens. The equation is given by

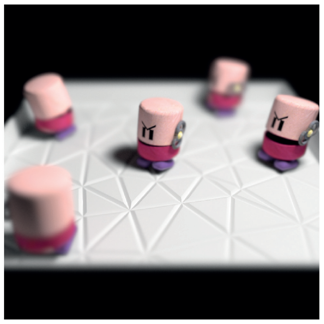
$$L_{dir}(x) = \int_u \left(\int_y f(x, u, y) V(x, u, y) \, \mathrm{d}y \right) k(x, u) \, \mathrm{d}u. \qquad (16)$$

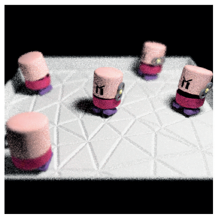
where $f(x, u, y)$ is the BRDF term, $V(x, u, y)$ is the visibility term, and $k(x, u)$ represents the texture or reflectance. Note that, this equation is difficult to separate, because both the BRDF term and the visibility term depend on samples from the lens and samples from the area light.

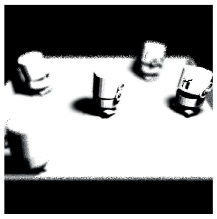
To solve the problem, we first denote the product of the BRDF term and the visibility term as $F(x, u, y) = f(x, u, y) \cdot V(x, u, y)$. Then we introduce an approximation by replacing F for each pixel with its average over every related lens sample u, or $F(x, u, y) \approx \bar{F}(x, y)|_u$. Then Equation (16) becomes

$$L_{dir}(x) \approx \int_u \left(\int_y \bar{F}(x, y)|_u \, \mathrm{d}y \right) k(x, u) \, \mathrm{d}u. \qquad (17)$$


(a) Our method, 9 spp, 580 ms


(b) Ground truth, 12288 spp


(c) Unfiltered


(d) Filtered visibility

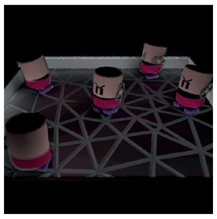

(e) Filtered indirect illumination

Fig. 12. The TOASTERS scene rendered with an area light, depth of field and global illumination. Our method achieves visually plausible results but uses only 9 samples per pixel, each sample with 1 ray for depth of field, 1 ray for soft shadows and 1 ray for indirect illumination.

We have essentially factored out the inner integral over the light for soft shadows, and the outer integral over the lens for defocus.

We now propose a two-step filtering algorithm. For each pixel, we sample the lens to shoot primary rays. For each valid hit, we shoot one (or more) secondary shadow rays and compute the corresponding F term. After this, we average primary rays to get $\bar{F}$.

The following filtering steps are straightforward. We first filter the $\bar{F}(x, y)$ light field samples, using our proposed four-step fast sheared filtering. Then we filter the resulting (x, u) light field from the previous step according to the lens to get the depth of field effect, again with our fast sheared filtering algorithm. Figure 12 shows the final result as well as different stages using our method. Compared to the claimed running time of 3.61 seconds for the same scene using axis-aligned filtering in Mehta et al. [2014], we achieve a 6× speed up, yet achieving a visually convincing result, although differences are noticeable as compared to the ground truth.

Note that, the approximation we proposed is almost as conservative as that introduced in Mehta et al. [2014]. In practice, it also guarantees accuracy. When the effect of u range is small, the approximated $\bar{F}$ is accurate. This indicates that the closer to the focal plane, the more accurate the approximation is. For those areas far from the focal plane, since the outer integral (lens filter) dominates, the output image is largely blurred, so minor inaccuracies of the visibility approximation could be neglected.

The time complexity of our separation scheme is still $O(nl)$, because we simply perform our fast sheared filtering twice. For the storage, the approximation allows us to store the samples for each effect separately; that is, the storage cost is still 4D rather than 6D. So we consider it practical and efficient for our algorithm to handle multiple effects.

ACKNOWLEDGMENTS

We thank the anonymous reviewers for their insightful comments.

REFERENCES

T. Annen, Z. Z. Dong, T. Mertens, P. Bekaert, and H. Seidel. 2008. Real-time all-frequency shadows in dynamic scenes. *ACM Transactions on Graphics* 27, 3, Article 34, 1–8.

U. Assarsson and T. Möller. 2003. A geometry-based soft shadow volume algorithm using graphics hardware. *ACM Transactions on Graphics* 22, 3, 511–520.

L. Belcour, C. Soler, K. Subr, N. Holzschuch, and F. Durand. 2013. 5D covariance tracing for efficient defocus and motion blur. *ACM Transactions on Graphics* 32, 3, 31:1–31:18.

J.-X. Chai, X. Tong, S.-C. Chan, and H.-Y. Shum. 2000. Plenoptic sampling. In *Proceedings of SIGGRAPH'00*. 307–318.

P. Clarberg and J. Munkberg. 2014. Deep shading buffers on commodity gpus. *ACM Transactions on Graphics* 33, 6, 227:1–227:12.

C. Crassin, F. Neyret, M. Sainz, S. Green, and E. Eiseman. 2011. Interative indirect illumination using voxel cone tracing. *Computer Graphics Forum* 30, 7, 1921–1930.

F. Crow. 1977. Shadow algorithms for computer graphics. In *Proceedings of SIGGRAPH'77*. 242–248.

H. Dammertz, D. Sewtz, J. Hanika, and H. P. A. Lensch. 2010. Edge-avoiding À-trous wavelet transform for fast global illumination filtering. In *Proceedings of the Conference on High Performance Graphics*. 67–75.

M. Delbracio, P. Musé, A. Buades, J. Chauvier, N. Phelps, and J.-M. Morel. 2014. Boosting Monte Carlo rendering by ray histogram fusion. *ACM Transactions on Graphics* 33, 1, 8:1–8:15.

F. Durand, N. Holzschuch, C. Soler, E. Chan, and F. Sillion. 2005. A frequency analysis of light transport. *ACM Transactions on Graphics* 24, 3, 1115–1126.

K. Egan, F. Durand, and R. Ramamoorthi. 2011a. Practical filtering for efficient ray-traced directional occlusion. *ACM Transactions on Graphics* 30, 6.

K. Egan, F. Hecht, F. Durand, and R. Ramamoorthi. 2011b. Frequency analysis and sheared filtering for shadow light fields of complex occluders. *ACM Transactions on Graphics* 30, 2, 9:1–9:13.

K. Egan, Y. Tseng, N. Holzschuch, F. Durand, and R. Ramamoorthi. 2009. Frequency analysis and sheared reconstruction for rendering motion blur. *ACM Transactions on Graphics* 28, 3, 93:1–93:13.

G. Guennebaud, L. Barthe, and M. Paulin. 2006. Real-time soft shadow mapping by backprojection. In *Proceedings of EGSR'06*. 227–234.

G. Guennebaud, L. Barthe, and M. Paulin. 2007. High-quality adaptive soft shadow mapping. *Computer Graphics Forum* 26, 3, 525–533.

T. Hachisuka, W. Jarosz, R. Weistroffer, K. Dale, G. Humphreys, M. Zwicker, and H. Jensen. 2008. Multidimensional adaptive sampling and reconstruction for ray tracing. *ACM Transactions on Graphics* 27, 3, 33:1–33:10.

J. Hasenfratz, M. Lapierre, N. Holzschuch, and F. Sillion, 2003. A survey of real-time soft shadow algorithms. *Computer Graphics Forum* 22, 4, 753–774.

G. Johnson, W. Hunt, A. Hux, W. Mark, C. Burns, and S. Junkins, 2009. Soft irregular shadow mapping: fast, high-quality, and robust soft shadows. In *Proceedings of I3D'09*. 57–66.

N. K. Kalantari and P. Sen. 2013. Removing the noise in Monte Carlo rendering with general image denoising algorithms. *Computer Graphics Forum* 32, 2, 93–102.

S. Laine, T. Aila, U. Assarsson, J. Lehtinen, and T. Möller. 2005. Soft shadow volumes for ray tracing. *ACM Transactions on Graphics* 24, 3, 1156–1165.

S. S. Lee, E. Eisemann, and H.-P. Seidel. 2010. Real-time lens blur effects and focus control. *ACM Transactions on Graphics* 29, 4, 65:1–65:7.

J. Lehtinen, T. Aila, J. Chen, S. Laine, and F. Durand. 2011. Temporal light field reconstruction for rendering distribution effects. *ACM Transactions on Graphics* 30, 4, 55:1–55:12.

J. Lehtinen, T. Aila, S. Laine, and F. Durand. 2012. Reconstructing the indirect light field for global illumination. *ACM Transactions on Graphics* 31, 4, 51:1–51:10.

K. Lei and J. F. Hughes. 2013. Approximate depth of field effects using few samples per pixel. In *Proceedings of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games. I3D'13*. ACM, 119–128.

T.-M. Li, Y.-T. Wu, and Y.-Y. Chuang, 2012. SURE-based optimization for adaptive sampling and reconstruction. *ACM Transactions on Graphics* 31, 6, 186:1–186:9.

M. McCool. 1999. Anisotropic diffusion for Monte Carlo noise reduction. *ACM Transactions on Graphics* 18, 2, 171–194.

S. Mehta, B. Wang, and R. Ramamoorthi. 2012. Axis-aligned filtering for interactive sampled soft shadows. *ACM Transactions on Graphics* 31, 6, 163:1–163:10.

S. Mehta, YAO, J., R. Ramamoorthi, and F. Durand. 2014. Factored axis-aligned filtering for rendering multiple distribution effects. *ACM Transactions on Graphics* 33, 5.

S. Mehta, B. Wang, R. Ramamoorthi, and F. Durand. 2013. Axis-aligned filtering for interactive physically-based diffuse indirect lighting. *ACM Transactions on Graphics* 32, 4, 96:1–96:12.

D. Mitchell. 1991. Spectrally Optimal Sampling for Distribution Ray Tracing. In *Proceedings of SIGGRAPH'91*. 157–164.

J. Munkberg, K. Vaidyanathan, J. Hasselgren, P. P. Clarberg, and T. Akenine-Möller. 2014. Layered reconstruction for defocus and motion blur. In *Computer Graphics Forum*. Vol. 33. Wiley Online Library, 81–92.

R. Overbeck, C. C. Donner, and R. Ramamoorthi. 2009. Adaptive wavelet rendering. *ACM Transactions on Graphics* 28, 5.

S. Paris and F. Durand. 2006. A fast approximation of the bilateral filter using a signal processing approach. In *Proceedings of the 9th European Conference on Computer Vision (ECCV'06)*. 568–580.

G. Petschnigg, R. Szeliski, M. Agrawala, M. Cohen, H. H. Hoppe, and K. Toyama. 2004. Digital photography with flash and no-flash image pairs. In *SIGGRAPH'04: ACM SIGGRAPH 2004 Papers*. ACM, New York, 664–672.

M. Potmesil and I. Chakravarty. 1981. A lens and aperture camera model for synthetic image generation. In *Proceedings of SIGGRAPH'81*. 297–305.

R. Ramamoorthi, J. Anderson, M. Meyer, and D. Nowrouzezahrai. 2012. A theory of Monte Carlo visibility sampling. *ACM Transactions on Graphics* 31, 5.

R. Ramamoorthi and P. Hanrahan. 2001. A signal-processing framework for inverse rendering. In *Proceedings of SIGGRAPH'01*. 117–128.

T. Ritschel, C. Dachsbacher, T. Grosch, and J. Kautz. 2012. The state of the art in interactive global illumination. *Computer Graphics Forum* 31, 1, 160–188.

T. Ritschel, T. Engelhardt, T. Grosch, H.-P. Seidel, J. Kautz, and C. Dachsbacher. 2009. Micro-rendering for scalable, parallel final gathering. *ACM Transactions on Graphics* 28, 5.

F. Rouselle, C. Knaus, and M. Zwicker. 2012. Adaptive rendering with non-local means filtering. *ACM Transactions on Graphics* 31, 6, 195:1–195:11.

H. Rushmeier and G. Ward. 1994. Energy preserving non-linear filters. 131–138.

P. Sen and S. Darabi. 2012. On filtering the noise from the random parameters in Monte Carlo rendering. *ACM Transactions on Graphics* 31, 3, 18:1–18:15.

P. Shirley, T. Aila, J. Cohen, E. Enderton, S. Laine, D. Luebke, and M. McGuire. 2011. A local image reconstruction algorithm for stochastic rendering. In *Proceedings of the ACM Symposium on Interactive 3D Graphics*. 9–14.

P. Sloan, J. Kautz, and J. Snyder. 2002. Precomputed radiance transfer for real-time rendering in dynamic, low-frequency lighting environments. *ACM Transactions on Graphics* 21, 3, 527–536.

C. Soler and F. Sillion. 1998. Fast calculation of soft shadow textures using convolution. In *Proceedings of SIGGRAPH'98*. 321–332.

C. Soler, K. Subr, F. Durand, N. Holzschuch, and F. Sillion. 2009. Fourier depth of field. *ACM Transactions on Graphics* 28, 2, 18:1–18:12.

K. Vaidyanathan, J. Munkberg, P. Clarberg, and M. Salvi. 2015. Layered light field reconstruction for defocus blur. *ACM Transactions on Graphics* 34, 2, 23:1–23:12.

X. Yu, R. Wang, and J. Yu. 2010. Real-time depth of field rendering via dynamic light field generation and filtering. *Computer Graphics Forum* 29, 7, 2099–2107.

Received October 2014; revised April 2014, August 2015; accepted August 2015