Layered Light Field Reconstruction for Defocus Blur

KARTHIK VAIDYANATHAN, JACOB MUNKBERG, PETRIK CLARBERG, and MARCO SALVI
Intel Corporation

We present a novel algorithm for reconstructing high-quality defocus blur from a sparsely sampled light field. Our algorithm builds upon recent developments in the area of sheared reconstruction filters and significantly improves reconstruction quality and performance. While previous filtering techniques can be ineffective in regions with complex occlusion, our algorithm handles such scenarios well by partitioning the input samples into depth layers. These depth layers are filtered independently and then combined together, taking into account inter-layer visibility. We also introduce a new separable formulation of sheared reconstruction filters that achieves real-time preformance on a modern GPU and is more than two orders of magnitude faster than previously published techniques.

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

General Terms: Algorithms

Additional Key Words and Phrases: Depth of field, reconstruction, defocus blur, light field reconstruction, Fourier analysis

ACM Reference Format:

Karthik Vaidyanathan, Jacob Munkberg, Petrik Clarberg, and Marco Salvi. 2015. Layered light field reconstruction for defocus blur. ACM Trans. Graph. 34, 2, Article 23 (February 2015), 12 pages.
DOI: http://dx.doi.org/10.1145/2699647

1. INTRODUCTION

Depth of field is a widely used camera lens effect in computer-generated imagery of movies and games today. Offline renderers can accurately simulate depth of field by sampling numerous positions on the lens using distribution ray tracing, stochastic rasterization, or multilayer rendering. On the other hand, real-time renderers typically approximate this effect by blurring samples from a traditional 2D rasterizer, resulting in objectionable visual artifacts but fast performance. Although significant research progress has been made in the area of real-time stochastic rendering, the large number of samples required to produce noise-free images remains a key challenge in making these higher-quality rendering approaches practical for real time.

Authors' addresses: K. Vaidyanathan (corresponding author), J. Munkberg, P. Clarberg, and M. Salvi, Intel Corporation; email: karthik.vaidyanathan@intel.com.

DOI: http://dx.doi.org/10.1145/2699647

Recent advances in light field reconstruction techniques have made it possible to reproduce low-noise defocus blur images with a small number of samples; however, the computational overhead of these reconstruction algorithms precludes real-time performance. Reconstruction techniques based on frequency analysis and sheared filtering are particularly promising. These techniques suppress noise by deriving reconstruction filters that tightly bound the sheared frequency spectrum of the light field.

Unfortunately, these filters are ineffective in regions with complex occlusion, such as when in-focus and out-of-focus objects contribute to the same pixel. Moreover, variations in the light field can lead to different filtering parameters for each pixel. This prevents efficient separable filter formulations and greatly impacts performance.

This article introduces a reconstruction algorithm for defocus blur that significantly improves the reconstruction quality and performance of sheared reconstruction filters. Our technique handles regions with complex occlusion by partitioning the light field samples into depth layers. Since each layer has a reduced depth range, the corresponding spectral bounds of the light field become narrower, resulting in more effective reconstruction filters.

However, partitioning the light field into depth layers presents a challenge since different layers can occlude each other in complex ways. In order to address inter-layer occlusion, we derive filters to reconstruct the average radiance *and* average visibility for each layer. These estimates are then used to composite the partitions from front to back and reconstruct the final pixel color. We also analyze the impact of using an average visibility estimate instead of the exact visibility, and show that reconstruction quality is not compromised.

Finally, we introduce an algorithm for efficiently evaluating sheared filters. Unlike previous work where a unique filter is evaluated per pixel, we divide the domain into tiles and use a small number of fixed sheared filters within each tile. By deriving a novel separable formulation for these filters, we show that filtering costs can be amortized over the large number of pixels in a tile, enabling real-time performance.

2. RELATED WORK

In recent years, substantial progress has been made in the area of light field analysis and reconstruction for a wide range of applications, such as light field rendering, soft shadows, indirect illumination, motion blur, and camera defocus. The key insight is that, although a light field is high dimensional, variations in the signal are often highly anisotropic. Directions of slow variation can be both sparsely sampled and reconstructed using wide filters for noise reduction, without significant loss of fidelity. This has been explored in a multitude of recent papers.

Chai et al. [2000] analyze the spectrum of the light field for plenoptic sampling and show that it is bounded by the minimum and maximum depth values in the scene. They use this observation to determine a minimum sampling rate and a reconstruction filter for light field rendering. They also show that the light field frequency bounds become narrower when the scene is partitioned

into depth layers, and hence fewer samples are required for reconstruction. However, they do not address occlusion between layers and ignore visibility in their analysis. We extend their approach by handling occlusion across depth layers and deriving an efficient sheared reconstruction filter.

Light field frequency analysis [Durand et al. 2005] has also been applied in many other areas. Soler et al. [2009] use frequency analysis for adaptive sampling with defocus blur. Egan et al. [2009] use frequency analysis to derive a sheared reconstruction filter for motion blur. This work has also been extended to soft shadows [Egan et al. 2011b; Mehta et al. 2012] and directional occlusion [Egan et al. 2011a]. All these methods provide specialized derivations of the bandlimits in their respective applications. Recently, Belcour et al. [2013] presented a unified approach for predicting the bandwidth of radiance in the 5D domain of space, angle, and time. Key to their method is a compact representation of the variation and anisotropy of radiance using 5D covariance matrices, from which 2D sheared filters in the image plane are derived.

We build on previous approaches to sheared filtering and derive a sheared filter for defocus blur based on frequency analysis. However, as observed by Lehtinen et al. [2011], most previous sheared filtering approaches revert to a less effective axis-aligned filter near occlusion boundaries or in regions with varied motion, producing a noisy result. To address this, we combine our analysis with partitioning into depth layers to effectively reduce the noise in the reconstructed output. This avoids reverting to an axis-aligned filter in areas with complex occlusion. Additionally, we design a *separable* sheared filter that avoids a complex search in the 4D light field and amortizes the filtering cost over a large number of pixels. This is key to our improved performance.

To composite the different depth layers, we introduce a new approach inspired by previous work on analyzing the spectrum of the visibility function [Durand et al. 2005; Lanman et al. 2008; Egan et al. 2011b; Ramamoorthi et al. 2012]. We extend their analysis to reconstruct the visibility function in addition to the radiance, and use this to resolve visibility across layers. This ensures accurate, low-noise results also in areas of complex occlusion.

There are also a multitude of nonfrequency-analysis-based methods that are applicable to rendering of defocus blur. Hachisuka et al. [2008] use gradient-based adaptive sampling and reconstruction to render distribution effects. Their technique spends a considerable amount of time searching in a high-dimensional data structure to locate nearby samples. Lehtinen et al. [2011] combine higher-dimensional reprojection and visibility reconstruction to upsample light fields including motion, defocus, and soft shadows from very few input samples. Their work was recently extended to indirect illumination [Lehtinen et al. 2012]. The reprojection step also requires a search in a higher-dimensional light field, and a complex local visibility reconstruction, taking tens of seconds per frame even with an optimized GPU implementation.

A separate line of research uses statistical noise estimation methods to perform denoising in Monte Carlo rendering of depth of field and other distribution effects. Many modern methods [Sen and Darabi 2012; Kalantari and Sen 2013] produce surprisingly good results. However, these general methods naturally cannot compete on equal terms with more specialized reconstruction filters, which may exploit domain-specific knowledge (e.g., camera geometry and sample depths) to provide better and more temporally stable results. It should also be noted that many denoising algorithms rely on noise estimation to guide adaptive sampling [Rousselle et al. 2012; Kalantari and Sen 2013], which is a sensible strategy in ray-tracing-based Monte Carlo rendering. However, our goal is primarily reconstruction from uniformly sampled light fields, which may be created efficiently using, for example, stochastic rasterization [Akenine-Möller et al. 2007].

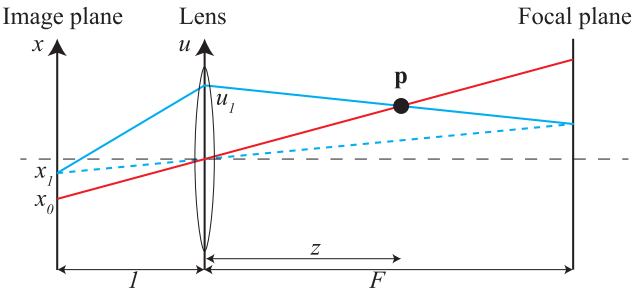

Fig. 1. We parameterize our light field using an image plane coordinate x, and a lens coordinate u. In the illustration, we see that the two rays (x_1, u_1) and $(x_0, 0)$ both hit the point $\mathbf{p}$. Using similar triangles, we see that $x_0 = x_1 + \frac{z-F}{zF}u_1$, where $c(z) = \frac{z-F}{zF}$ is the circle of confusion at the image plane.

Shirley et al. [2011] introduce an image space denoising filter for stochastically rendered images. Since their technique does not consider the sheared nature of the light field, it tends to over-blur the result for effective noise reduction. For defocus blur, they also statically partition the scene into a few layers to improve performance; however, they use a separate filter per pixel that has a significant computational cost.

Lee et al. [2010] render defocus blur by ray tracing into a layered color and depth buffer. This technique allows for artistic control of depth-of-field effects, but requires depth peeling to render the multilayer frame buffer. Also since the layers are generated using a pin-hole camera, visible parts of the scene can get occluded, resulting in visible errors. Lei and Hughes [2013] take a similar approach by rendering a few sharp layers using a pin-hole camera. They then splat and gather the samples on the image plane to approximate defocus blur for a particular focus depth.

Finally, there are many simpler postprocessing algorithms for defocus blur [Potmesil and Chakravarty 1981; Demers 2004; Hammon 2007; Yu et al. 2011] that use a pin-hole rendering and depth buffer to simulate defocus blur. These approaches lack visibility information and often produce artifacts. However, they are commonly used in real-time rendering due to their high performance and simplicity.

3. SHEARED FILTERS FOR DEFOCUS BLUR

Our filter design is based on the Fourier spectra for depth of field, which has been previously derived by Soler et al. [2009]. For clarity, we show the derivation of the Fourier transform and then use the properties of the spectra to derive a sheared reconstruction filter. Our derivation closely follows previous work, such as Egan et al. [2011b]. In later sections, we derive an efficient sheared filter formulation that, combined with depth partitioning, results in an effective and practical reconstruction technique.

Geometry Setup. For clarity of presentation, we first analyze a 2D light field. We parameterize the light field using two parallel planes: an image plane with coordinate x and a lens coordinate u one unit away. This setup is shown in Figure 1.

Let $\mathbf{p}$ be a point in the scene, which is a source of radiance l. We assume that the outgoing radiance from $\mathbf{p}$ does not vary when viewed from different points on the lens and that $\mathbf{p}$ is visible from the entire lens (occlusion will be handled later). If we have a finite aperture (i.e., not a pin-hole camera) and $\mathbf{p}$ is out of focus at depth z, the radiance from $\mathbf{p}$ will be smeared over multiple pixels. For example,

in Figure 1, **p** is hit both by a ray at $(x_0, 0)$ (a ray from x_0 through the center of the lens) and by the ray (x_1, u_1).

Due to our assumption that **p** is a Lambertian source visible from the entire lens, the radiance l for the two rays are the same, that is, $l(x_1, u_1) = l(x_0, 0)$. From the geometry of our setup, we see that

$$x_0 = x_1 + \frac{z - F}{zF} u_1 = x_1 + c(z)u_1, \tag{1}$$

where $c(z) = \frac{z-F}{zF}$ is the signed *circle of confusion* at the image plane, that is, $c(z) < 0$ for points in front of the focal plane and positive beyond.

This implies that, in the absence of occlusion, for a Lambertian source at depth z, we have

$$l(x, u) = l(x + c(z)u, 0). \tag{2}$$

3.1 Frequency Analysis

The *irradiance*, $e(x)$, at a point x on the image plane (typically the center of each pixel) is given by integrating the radiance over the lens[1]

$$e(x) = \int l(x, u)a(u)\, du, \tag{3}$$

where $a(u)$ describes the shape and size of the camera aperture. For the purpose of our analysis, we assume $a(u)$ is a Gaussian centered around $u = 0$.

In the following, we will use capital letters to denote Fourier transforms, and let Ω_x and Ω_u represent the frequencies along the x and u axes, respectively.[2] The integral in Eq. (3) can be seen as a 2D convolution of the product of l and a with a constant function 1 in u and a Dirac delta in x:

$$e(x) = (l(x, u)a(u)) * \delta(x). \tag{4}$$

Note that $\delta(x)$ is defined in 2D space and represents a line along the u axis.

Applying a 2D Fourier transform, we can express the spectrum of the irradiance as a convolution evaluated at $\Omega_u = 0$ (it is zero elsewhere):

$$\begin{aligned} E(\Omega_x) &= (L(\Omega_x, \Omega_u) * \delta(\Omega_x)A(\Omega_u)) \cdot \delta(\Omega_u) \\ &= \int L(\Omega_x, \Omega_{u'})A(\Omega_{u'})d\Omega_{u'} \quad \text{for } \Omega_u = 0, \end{aligned} \tag{5}$$

where we have used that $a(u) = a(-u)$ (shortly we rename $\Omega_{u'} = \Omega_u$ for clarity). In the following sections, we will study the bandlimits of L and A in order to derive an efficient reconstruction filter.

3.2 Radiance Frequency Response

We denote the radiance at the center of the lens:

$$l^0(x) = l(x, 0). \tag{6}$$

As shown in Eq. (2), for a Lambertian source in the absence of occlusion, we can express the radiance in direction (x, u) as

$$l(x, u) = l^0(x + c(z)u). \tag{7}$$

[1]We assume a constant *lens form factor* [Kolb et al. 1995] and ignore it in our analysis.

[2]We work in ordinary frequencies, where the unit of Ω_x is pixel^{-1} if x is measured in pixels, and so on.

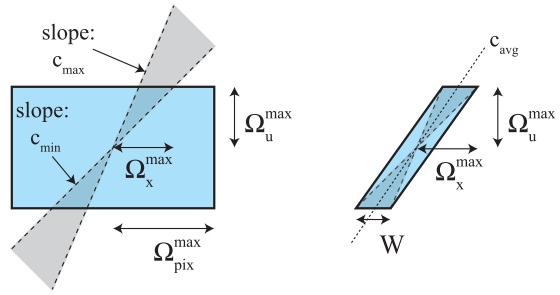

Fig. 2. Filter for the displayable frequencies in the (Ω_x, Ω_u) plane (illustration adapted from Egan et al. [2011b] © ACM 2011). The left illustration shows a simple axis-aligned filter, using bandlimits given by the extents of a pixel and the aperture function, respectively. The right illustration shows a sheared filter capturing approximately the same frequency content.

Transforming this to the Fourier domain, we get [Egan et al. 2011b]:

$$L(\Omega_x, \Omega_u) = L^0(\Omega_x)\delta(\Omega_u - c(z)\Omega_x). \tag{8}$$

It can be seen that the radiance frequency response of an emitter **p** at depth z is sheared along the line $\Omega_u - c(z)\Omega_x = 0$. Now, consider a set of unoccluded emitters in a narrow depth range $z \in [z_{\min}, z_{\max}]$. As shown by Chai et al. [2000], the frequency response will be mostly contained within a wedge bounded by the slopes $c_{\min} = c(z_{\min})$ and $c_{\max} = c(z_{\max})$, since $c(z)$ is monotonically increasing for $z > 0$. This is shown in Figure 2.

3.3 Frequency Bounds of Sheared Filter

In this section, we will first estimate bandlimits for the frequency content of the product of the radiance and aperture function, and then use these bounds to design a sheared filter.

From Eq. (5), we see that the frequencies that contribute to the final irradiance are given by the product of L and A. We assume that the aperture function $a(u)$ is a Gaussian with standard deviation σ_u. Although not physically realizable, a Gaussian aperture simplifies the analysis[3] and makes it possible to formulate an efficient separable filter. In practice, the majority of the signal is captured by the central peak, for instance, 99% lies within $3\sigma_u$. Following the notation of Mehta et al. [2012], we express this Gaussian filter in the primal domain as

$$w(u; \sigma_u) = \frac{1}{\sqrt{2\pi}\sigma_u} e^{-\frac{u^2}{2\sigma_u^2}}. \tag{9}$$

For the purpose of our analysis, the corresponding bandlimit in the frequency domain is defined as $\Omega_u^{\max} = (2\pi\sigma_u)^{-1}$. Note that this is not a strict bound since a Gaussian has infinite support. Similarly, a screen space filter $w(x; \sigma_x)$ can be defined. We use a Gaussian pixel filter with standard deviation σ_{pix}. We set the corresponding Fourier domain bandlimit to $\Omega_{\text{pix}}^{\max} = (2\pi\sigma_{\text{pix}})^{-1}$.

Given these bandlimits, a simple filter in the frequency domain is given by

$$w\big(\Omega_x;\ \Omega_{\text{pix}}^{\max}\big)\ w\big(\Omega_u;\ \Omega_u^{\max}\big). \tag{10}$$

These bandlimits are shown in blue in Figure 2 (left). In the primal domain, the corresponding filter over the pixel footprint and the entire lens is

$$w_{\text{simple}}(x, u) = w(x; \sigma_{\text{pix}})w(u; \sigma_u). \tag{11}$$

[3]The Fourier transform of a Gaussian with standard deviation σ is a scaled Gaussian with standard deviation $1/(2\pi\sigma)$ in ordinary frequencies.

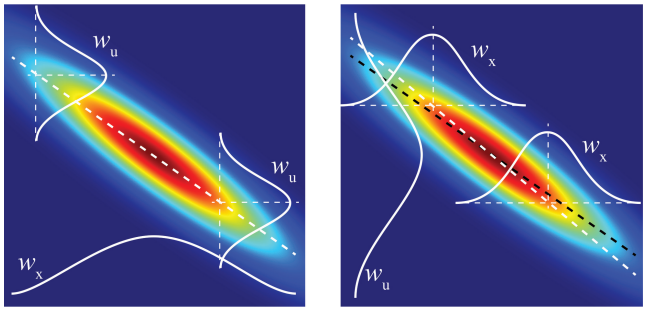

Fig. 3. Left: the primal domain filter from Eq. (15), obtained from the frequency analysis, consisting of a filter along x and a sheared filter in u. Right: the transformed filter from Eq. (23), which is a sheared filter in x and a filter along u. This filter is obtained by modifying the filter widths and shear (the slope from the left filter is included in black in the right illustration to more clearly depict the difference). Note that the resulting 2D filter kernel is exactly the same in both cases.

We use this filter if $c_{\min}$ and $c_{\max}$ have different signs *and* $\max|c| \geq \Omega_u^{\max}/\Omega_{\text{pix}}^{\max}$ (i.e., the frequency content lies in a wide wedge).

However, if $c_{\min}$ and $c_{\max}$ are similar, a filter that more tightly bounds the frequency content of LA can be derived by exploiting Eq. (8). This is shown in Figure 2 (right).

We will first derive frequency bounds and the filter for the case where the depth range does not cross the focal plane and $\max|c| \geq \Omega_u^{\max}/\Omega_{\text{pix}}^{\max}$. For this case, we follow previous work [Egan et al. 2011b] and bound the wedge in the frequency domain with a parallelogram (Figure 2, right). The width W of the parallelogram is given by

$$W = \Omega_u^{\max}\left(\frac{1}{c_{\min}} - \frac{1}{c_{\max}}\right), \tag{12}$$

and its slope is

$$c_{\text{avg}} = \frac{2c_{\min}c_{\max}}{c_{\min}+c_{\max}}. \tag{13}$$

To transform the simple axis-aligned filter from Eq. (10) to the sheared filter, we first scale the filter in Ω_x by a factor

$$S_{\Omega_x} = \frac{W}{2\Omega_{\text{pix}}^{\max}}. \tag{14}$$

Then, we apply a horizontal shear by $1/c_{\text{avg}}$ in Ω_x per unit Ω_u.

Similar to Egan et al. [2011b], we obtain an equivalent primal domain filter by taking the filter from Eq. (11) and first applying the inverse scaling $1/S_{\Omega_x}$ in the x dimension, followed by a negative vertical shear by $-1/c_{\text{avg}}$ in u per unit x.

The resulting filter can thus be expressed as a product of two 1D Gaussian filters:

$$w_{\text{shear}}(x,u) = w(x;\sigma_x)\, w\left(u + \frac{x}{c_{\text{avg}}};\sigma_u\right), \tag{15}$$

where

$$\sigma_x = \frac{\sigma_{\text{pix}}}{S_{\Omega_x}} = \frac{1}{\pi W}. \tag{16}$$

The resulting filter shape is shown in Figure 3 (left). This reconstruction filter includes both the aperture function and a screen space filter, where the width of the screen space filter is larger for samples out of focus.

Note that, while σ_u is directly determined by the aperture function, our selection of $\Omega_u^{\max}$ influences the spatial extent of the filter. A larger cutoff necessitates inclusion of a wider range of spatial frequencies, and thus a larger W in Eq. (16) with a narrower filter as a result.

3.4 Transforming the Filter Axes

In this section, we show that the Gaussian filter in Eq. (15) can alternatively be expressed as a product of an aperture filter that is purely a function in u and a sheared screen space filter. We later show that this modified filter can be efficiently evaluated as a preintegration step computed once for each layer, followed by a separable 2D filter in x and y.

To find such a suitable separable formulation, we first express $w_{\text{shear}}(x,u)$ as a 2D *elliptical Gaussian filter* [Heckbert 1989] of the form

$$w_{\text{shear}}(x,u) = \frac{1}{2\pi|\mathbf{M}|}e^{-\frac{1}{2}\mathbf{x}(\mathbf{M}^T\mathbf{M})^{-1}\mathbf{x}^T}, \tag{17}$$

where $\mathbf{x} = (x,u)$, and

$$\mathbf{M} = \begin{bmatrix}\sigma_x & 0\\ 0 & \sigma_u\end{bmatrix}\begin{bmatrix}1 & -\frac{1}{c_{\text{avg}}}\\ 0 & 1\end{bmatrix}. \tag{18}$$

To reformulate the filter as an aperture filter independent of x, times a sheared screen space filter, that is, of the form

$$w(x+\eta\,u;\sigma_x')w(u;\sigma_u'), \tag{19}$$

we search for another warping matrix

$$\mathbf{N} = \begin{bmatrix}\sigma_x' & 0\\ 0 & \sigma_u'\end{bmatrix}\begin{bmatrix}1 & 0\\ -\eta & 1\end{bmatrix}. \tag{20}$$

To obtain the same filter, we must enforce

$$\mathbf{N}^T\mathbf{N} = \mathbf{M}^T\mathbf{M}. \tag{21}$$

Solving Eq. (21), we obtain

$$\sigma_x' = \frac{\sigma_x\sigma_u}{\gamma}, \quad \sigma_u' = \gamma, \quad \text{and} \quad \eta = \frac{\sigma_x^2}{c_{\text{avg}}\gamma^2}, \tag{22}$$

where $\gamma = \sqrt{\sigma_u^2 + \frac{\sigma_x^2}{c_{\text{avg}}^2}}$. Our reconstruction filter (shown in Figure 3, right) can then be expressed as

$$\begin{aligned} w_{\text{shear}}(x,u) &= w(x;\sigma_x)\, w\left(u+\frac{x}{c_{\text{avg}}};\sigma_u\right)\\ &= \underbrace{w(x+\eta\,u;\sigma_x')}_{w_x}\,\underbrace{w(u;\sigma_u')}_{w_u}. \end{aligned} \tag{23}$$

This means that we can obtain the *exact* same elliptical Gaussian filter $w_{\text{shear}}(x,u)$ by a modified scaling, followed by a shear in x per unit u. Thus, we have derived a filter where the aperture filter depends only on u, and we instead have a sheared screen space filter in x. As we show next, this enables a very efficient filter implementation.

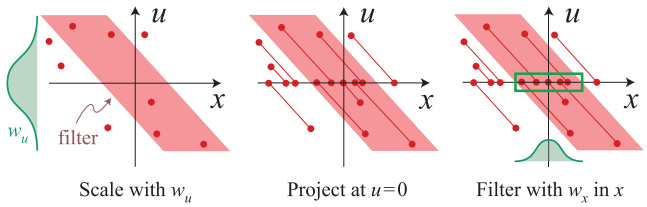

Fig. 4. Our filtering formulation described in (x, u) space. Each input radiance sample $l(x_i, u_i)$ is weighted by $w_u(u_i)$, and is then projected to $u = 0$ parallel to the line $x + \eta u = 0$ and accumulated. Finally, for each pixel, the filter w_x is evaluated over the accumulated values.

3.5 Efficient Filter Formulation

Let us return to the expression for the irradiance of a pixel x on the image plane (Eq. (3)). It can now be estimated as

$$\begin{aligned} e(x) &= \int l(x,u)a(u)\,du \\ &\approx \iint l(x',u)w(x'-x,u)dudx'. \end{aligned} \tag{24}$$

In other words, for light field radiance samples in a small depth range, we apply a filter kernel in (x, u) centered around the current pixel when reconstructing the irradiance.

If the reconstruction filter is separable in x and u as shown by Mehta et al. [2012], Eq. (24) is easily separable into an inner integral over u, followed by a convolution in screen space.

However, for depth layers out of focus, we want to exploit the sheared filter $w = w_{\text{shear}}(x, u)$ from Eq. (23)

$$\begin{aligned} e(x) &\approx \iint l(x',u)w_{\text{shear}}(x'-x,u)dudx' \\ &= \iint l(x',u)w_x((x'-x)+\eta u)w_u(u)dudx' \\ &\quad [\text{ Let } q = x' + \eta u,\ dq = dx' \text{ }] \\ &= \int \underbrace{\left[\int w_u(u)l(q-\eta u,u)du\right]}_{I_l(q)} w_x(q-x)dq \\ &= \int I_l(q)w_x(x-q)dq \;=\; (I_l * w_x)(x), \end{aligned} \tag{25}$$

since w_x is symmetric.

Given a set of radiance samples, the outer integral is a convolution in screen space with the filter w_x, and the inner integral represents a *preintegration* over the parameterized line $(q - \eta u, u)$.

Evaluating the inner integral at a pixel x would normally require an expensive search to gather samples along this line. We avoid this search by projecting each sample along the line to $u = 0$ and accumulating its weighted radiance at the target pixel q, thereby transforming the gather operation into a scatter operation.

As shown in Figure 4, an efficient implementation of this filter is given by the following steps:

(1) multiply each radiance sample $l(x_i, u_i)$ with $w_u(u_i)$;
(2) for all samples $l(x_i, u_i)$ in the current depth layer, accumulate $l(x_i, u_i)w_u(u_i)$ at the pixel $q = x_i + \eta u_i$;
(3) for each pixel, apply the screen space filter $w_x(x)$ over the accumulated sums.

Evaluation of the inner integral (first and second steps) can be performed *once* for each layer for all samples (e.g., in a tile), while the third step is unique for each pixel within each layer. This is in contrast to a naïve filter implementation, where a sheared filter kernel in (x, u) is repeated for each pixel in all layers. This efficient filter formulation is key to our performance.

It should be noted that the accumulation of weighted samples in step 2 may introduce very minor differences as, in practice, the number of bins (pixels) is finite and thus some quantization is introduced. This effect may be minimized by bilinearly splatting each sample to the four nearest bins, although we have not found this necessary.

Next, we will look at how small shears are handled and summarize our approach in relation to previous work.

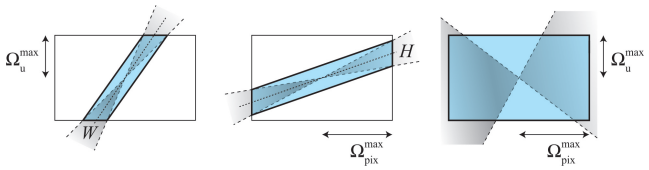

Fig. 5. In a majority of cases, the standard sheared filter (left) is used. For small slopes, a sheared filter is instead derived from $\Omega_{\text{pix}}^{\max}$ and the height H of the wedge (middle), and for very wide wedges, we revert to the simple axis-aligned filter (right).

3.6 Filter Bounds for Small Shears

If $\max|c| < \Omega_u^{\max}/\Omega_{\text{pix}}^{\max}$, the frequency wedge will be close to the Ω_x axis, and we instead use the bandlimit $\Omega_{\text{pix}}^{\max}$ to bound the filter size. Following the derivation in Section 3.3, we derive a parallelogram with height

$$H = \Omega_{\text{pix}}^{\max}(c_{\max} - c_{\min}), \tag{26}$$

and slope

$$\eta' = \frac{1}{2}(c_{\min} + c_{\max}). \tag{27}$$

The primal filter transform in this case is given by a scaling in u followed by a shear in x

$$w_{\text{shear}} = w(x + \eta' u; \sigma_{\text{pix}})\, w(u; \sigma_u''), \tag{28}$$

where

$$\sigma_u'' = \frac{2\sigma_{\text{pix}}}{c_{\max} - c_{\min}}. \tag{29}$$

Note that, in this case, we do not need to transform the filter axes, as the Fourier domain transforms (a scale in Ω_u and a shear in Ω_u per unit Ω_x) directly translate into an inverse scale in u and a (negative) shear in x per unit u in the primal domain.

Finally, as mentioned earlier, we fall back on the simple axis-aligned filter (Eq. (11)) if the depth range crosses the focal plane, that is, $c_{\min}$ and $c_{\max}$ are of different signs, and the frequency wedge is very wide. Figure 5 illustrates the three types of filters. For all cases, the efficient implementation described in the previous section is used.

3.7 Discussion

Although our sheared filters have been derived along the lines of previous work such as Egan et al. [2011b], the key difference is the filter axis transformation. This enables a much more efficient implementation, where the integral over the lens (u) can be reused over many samples.

Mehta et al. [2012] use an axis-aligned filter with bandlimits $(\Omega_u^{\max}/\min|c|, \Omega_u^{\max})$ in (Ω_x, Ω_u), when $\max|c| > \Omega_u^{\max}/\Omega_{\text{pix}}^{\max}$, which means that their primal domain filter is narrower as frequencies outside the wedge will be weighted in. With our sheared filter, we expect a smoother result due to our larger filter kernel in the primal domain.

Note that our derivation and the prior implementation extends naturally to a 4D light field $l(x, y, u, v)$. Since the shear is the same in the (x, u) and (y, v) slices, the two frequency bounds are the same. Both the aperture and the screen space filters are expressed as products of two identical 1D Gaussians. Hence, after accumulating the weighted samples at $(q_x, q_y) = (x_i, y_i) + \eta\,(u_i, v_i)$, the screen space filter $w_x(x, y)$ is separable in x and y as well.

4. DEPTH-LAYER COMPOSITING

When there is a large difference between the minimum and maximum depth inside a screen space region, the resulting frequency bounds of the sheared filter can become very large. This leads to a narrow reconstruction filter in the primal domain and limited noise reduction.

Chai et al. [2000] observe that, if the light field samples are partitioned into disjoint depth layers, one can derive a tighter filter for each layer. This is easy to see, as the filter width is closely related to $c(z)$, which varies monotonically with depth for $z > 0$, that is, in front of the lens (refer to Eq. (1)).

However, different layers may occlude each other and the radiance corresponding to a layer may or may not contribute to the overall radiance. Therefore, we need a way of compositing the filtered layers together that takes occlusion between layers into account.

Following Lehtinen et al. [2011], our light field samples can be expressed as

$$(x_i, y_i, u_i, v_i) \longmapsto (z_i, l_i), \tag{30}$$

for example, a depth z_i and a radiance value l_i are stored for each 4D sample $(\mathbf{x}_i, \mathbf{u}_i)$ where $\mathbf{x}=(x, y)$ and $\mathbf{u}=(u, v)$.

Now assume that the samples are sorted into N depth layers, where a layer j contains all light field samples within the depth range $[z_j^{\min}, z_j^{\max}]$. The layers are ordered front to back, where layer 0 is closest to the camera.

As a thought experiment, consider each layer j in isolation, disregarding inter-layer occlusions. Let $\alpha_j(\mathbf{x}, \mathbf{u}) \in [0, 1]$ represent the (unknown) layer opacity for a light field direction $(\mathbf{x}, \mathbf{u})$, and $l_j(\mathbf{x}, \mathbf{u})$ be the corresponding (unknown) layer radiance. With this, we can express the radiance along $(\mathbf{x}, \mathbf{u})$ using standard alpha blending [Porter and Duff 1984] as follows:

$$l(\mathbf{x}, \mathbf{u}) = \alpha_0(\mathbf{x}, \mathbf{u}) l_0(\mathbf{x}, \mathbf{u}) + \sum_{j=1}^{N} \alpha_j(\mathbf{x}, \mathbf{u}) l_j(\mathbf{x}, \mathbf{u}) \prod_{k=0}^{j-1} (1 - \alpha_k(\mathbf{x}, \mathbf{u})). \tag{31}$$

The final irradiance $e(\mathbf{x})$ at a pixel is obtained by integrating Eq. (31) over the aperture function as before (refer to Eq. (3)). In practice, this formulation requires a high-resolution 4D representation of opacity and radiance for each layer, which may be prohibitively expensive. Our main idea is instead to approximate the result by preintegrating the radiance and opacity over the lens separately within each layer. Then, we composite the integrated quantities.

Looking at a 2D slice (x, u), if we first introduce h_j to denote the premultiplied opacity and radiance in layer j

$$h_j(x, u) = \alpha_j(x, u) l_j(x, u), \tag{32}$$

we can expand the expression for the irradiance $e(x)$ on the image plane (Eq. (3)) by inserting Eq. (31), as follows:

$$\begin{aligned} e(x) &= \int l(x, u) a(u) du \\ &= \int [h_0 + (1 - \alpha_0) h_1 + \cdots] a(u) du \\ &= \int h_0 a(u) du + \int (1 - \alpha_0) h_1 a(u) du + \cdots \end{aligned} \tag{33}$$

Now, for all layers $j > 0$, we make the approximation

$$\begin{aligned} &\int \prod_{k=0}^{j-1} (1 - \alpha_k) h_j a(u) du \\ &\approx \prod_{k=0}^{j-1} \underbrace{\int (1 - \alpha_k) a(u) du}_{1 - \bar{\alpha}_k(x)} \cdot \underbrace{\int h_j a(u) du}_{e_j(x)}, \end{aligned} \tag{34}$$

that is, we directly use the product of the *average opacities* over the lens for each layer in front of j when factoring in the radiance from layer j. This is in contrast to the exact result obtained from integrating over the opacity of layers in front of j within the lens integral.

Using the shorthand notation $e_j(x) = \int h_j(x, u) a(u) du$ and $\bar{\alpha}_j(x) = \int \alpha_j(x, u) a(u) du$ introduced in the last step of Eq. (34), we can now express the approximated irradiance as

$$e(x) \approx e_0(x) + \sum_{j=1}^{N} e_j(x) \prod_{k=0}^{j-1} (1 - \bar{\alpha}_k(x)). \tag{35}$$

This can be seen as collapsing the lens dimension into a pin-hole camera, where each pixel holds an irradiance value $e_j(x)$ and an integrated opacity value $\bar{\alpha}_j(x)$ for each depth layer j. These values are then composited together over the depth layers using regular (premultiplied) alpha blending.

Discussion. With the formulation of Eq. (35) we have reduced the blending over 4D depth slices in Eq. (31) to an approximate alpha blending of 2D slices by preintegrating over the lens in each slice. This is computationally much more efficient, but the approximation in Eq. (34) may produce differences in partially occluded regions. To illustrate this, consider two out-of-focus objects, each in a separate layer. Both objects cover approximately the same area in (x, u) space, as shown in Figure 6.

We can see in the image of the epipolar plane that, if $|x|$ is larger than a certain value, both objects are visible from mutually exclusive regions of the lens and therefore do not occlude each other. If we integrate over the lens, that is, evaluate Eq. (33), at such a point, we expect an equal contribution from both partitions, as there is no mutual occlusion, thus a yellow color that fades to black as x increases. More precisely, in that region, $\int (1 - \alpha_0) h_1 a(u) = \int h_1 a(u)$, so the irradiance will be

$$e(x) = \int (h_0 + h_1) a(u) du = e_0(x) + e_1(x). \tag{36}$$

However, using the approximation in Eq. (34), we have lost the information of mutually exclusive visibility, and will instead get an expression for the irradiance as

$$e(x) = e_0(x) + (1 - \bar{\alpha}_0(x)) e_1(x). \tag{37}$$

This effect is visible in Figure 7, where the green background layer gets a smaller weight near the border. If the red circle is smaller, $\bar{\alpha}_0$

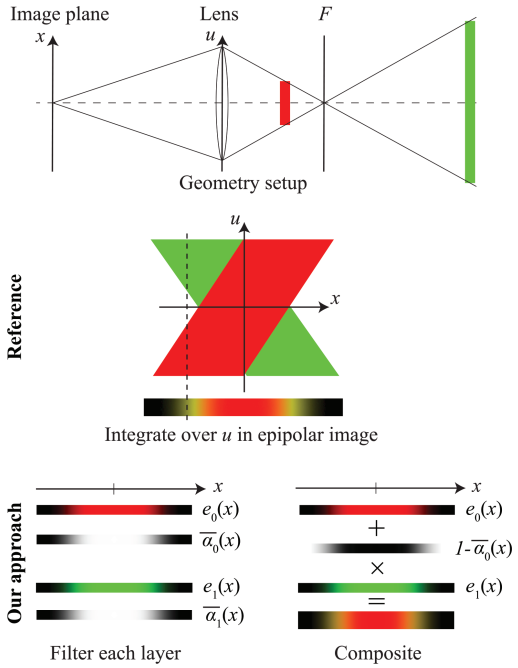

Fig. 6. A comparison of our compositing technique against a reference solution for a failure case with two out-of-focus objects. Note that the reconstructed irradiance differs slightly from the reference solution. This is due to our approximation in Eq. (34).

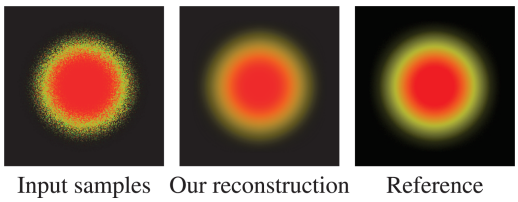

Fig. 7. Verification of the case in Figure 6, rendered in `pbrt` [Pharr and Humphreys 2010].

will be smaller, and the effect is less prominent. Also, if the lens is smaller, the blurry region where this effect shows up is smaller. These cases are shown in Figure 8. Note that, in realistic scenes, the approximation of Eq. (34) is rarely noticeable, as we will see in Section 7.

In the next section, we will see how to estimate the opacity term $\alpha_j(x, u)$ for each layer, taking inter-layer occlusion into account.

5. LAYER OPACITY

In this section, we describe our layer opacity estimation and filtering. We start by estimating opacity within each layer based on a sparse set of light field samples. We then reconstruct the integrated opacity over the lens for each layer $\bar{\alpha}_j(x)$ using a (layer-specific) sheared filter.

Estimating Layer Opacity From Light Field Samples. Let us again look at the problem in the 2D subspace (x, u). The opacity $\alpha_j(x, u)$ for a depth layer j is a scalar function with a value between zero and one for each direction in the light field. If a ray (x_i, u_i)

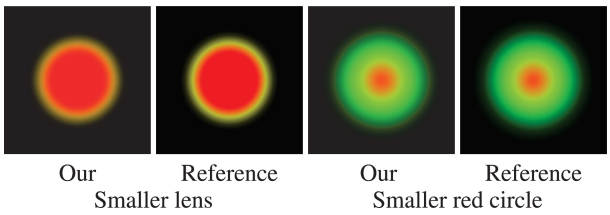

Fig. 8. Variations of the case in Figure 6, rendered in `pbrt`. Note that if the lens is smaller, fewer samples will hit layer 1 and the visibility approximation will be less evident (left image pair). Also, if the foreground layer has less coverage, the visibility approximation is again less intrusive (right image pair).

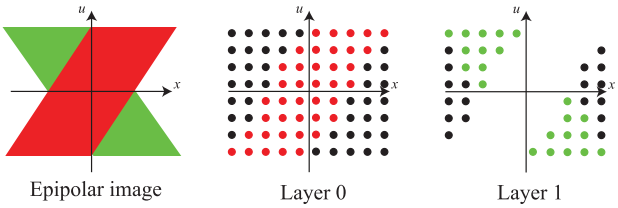

Fig. 9. Sample distribution for each layer for the test scene in Figure 6. The foreground layer (layer 0) has the best opacity (and radiance) estimate as all samples are valid. The background layer (layer 1) has a degraded estimate as some of the samples are occluded by the foreground and hence unknown. Note that this is an illustrative example. In our implementation, we use stochastically distributed light field samples.

passes through all layers $\{0, \ldots, j-1\}$ and hits something in layer j, we set $\alpha_j(x_i, u_i) = 1$. We then make two observations:

(1) If $\alpha_j(x_i, u_i) = 1$, then $\alpha_k(x_i, u_i) = 0,\ \forall k < j$,
(2) If $\alpha_j(x_i, u_i) = 1$, then $\alpha_k(x_i, u_i) = \text{unknown},\ \forall k > j$.

The first observation simply states that there is a "hole" (zero opacity) in all layers $\{0, \ldots, j-1\}$ for the ray (x_i, u_i). The second observation implies that we have no information of the opacity for layers further along the ray. In practice, this means that our opacity estimate will be best for foreground layers and degrade with depth, as fewer and fewer samples reach layers farther away from the camera as shown in Figure 9.

Integrated Layer Opacity. At this point, we have a sparse set of values in each layer representing opacity. In practice, each layer contains samples within a depth range $z \in [z_j^{\min}, z_j^{\max}]$. For each layer and pixel, we want to integrate the opacity over the aperture function. The result can be seen as an image rendered through a pin-hole camera, where each pixel captures the integrated opacity variation over the lens for a scene that only contains this depth layer. We call this integrated opacity $\bar{\alpha}_j(x)$ for layer j

$$\bar{\alpha}_j(x) = \int \alpha_j(x, u) a(u) du. \tag{38}$$

If we revisit Figure 1, we see that a point $\mathbf{p}$ will be hit by a ray through pixel x_0 through the center of the lens. Furthermore, due to the refraction in the lens, $\mathbf{p}$ will be hit by other rays. More precisely, in the absence of occlusion, all rays $(x_0 + c(z)u, u)$ will hit $\mathbf{p}$. Similarly to Eq. (7), we can express the opacity $\alpha_j(x, u)$ of a (small) point at depth z as a function of the opacity at the center of the lens of an offset pixel, that is,

$$\alpha(x, u) = \alpha^0(x + c(z)u). \tag{39}$$

In other words, for a depth layer j at z, and a given pixel x, if there is any sample on the parameterized line $(x + c(z)u, u)$, there is a sample visible from some point on the lens at this pixel. The integrated opacity $\bar{\alpha}_j(x)$ is the (aperture-weighted) integral over all samples parallel to the line $x + c(z)u = 0$ (refer to Figure 4).

Note the analogy with the analysis for the irradiance in Section 3.3. We leverage those results and use the *same* reconstruction filter w_{shear} as used for the radiance samples within a layer. This is also similar to previous work for occluder spectra [Lanman et al. 2008], and the visibility function in the context of soft shadows [Egan et al. 2011b; Ramamoorthi et al. 2012].

Following Eq. (25), the reconstructed opacity $\bar{\alpha}_j(x)$ integrated over the lens for a pixel x in layer j can be written as

$$\bar{\alpha}_j(x) \approx \int \underbrace{\left[\int w_u(u)\alpha_j(q - \eta\, u, u)du \right]}_{I_{\alpha_j}(q)} w_x(x - q)dq, \qquad (40)$$

where we have used the substitution $q = x + \eta\, u$. Note that η is a layer-specific value.

6. IMPLEMENTATION

In this section, we look at some practical aspects of how to put together the components (i.e., the sheared filters, layer compositing, and opacity estimates) described in the previous sections into an efficient implementation.

6.1 Practical Filter

In practice, $\bar{\alpha}_j(x)$ and the irradiance $e_j(x)$ for a layer j are evaluated according to the pseudocode in Algorithm 1. This includes a preintegration step that evaluates the inner integrals in Eqs. (25) and (40), followed by a 2D screen space filter.

ALGORITHM 1: Filter at pixel x for layer j

```
for all light field samples (x_i, u_i, z_i) do                ▷ Pre-Integration
    q = x_i + η u_i
    if z_i in depth range of layer j then
        I_α(q) += w_u(u_i)
        I_e(q) += l(x_i, u_i) w_u(u_i)
        n(q) += w_u(u_i)                                      ▷ Normalization factor
    else
        if z_i is in layer k > j then                         ▷ Hole in current layer
            n(q) += w_u(u_i)
        end if
    end if
end for
for all pixels q_i in kernel of w_x do                        ▷ Screen space filter
    Ī_α(x) += I_α(q_i) w_x(x − q_i)
    Ī_e(x) += I_e(q_i) w_x(x − q_i)
    n̄(x) += n(q_i) w_x(x − q_i)
end for
ᾱ_j(x) = Ī_α(x) / n̄(x),   e_j(x) = Ī_e(x) / n̄(x)              ▷ Normalization step
```

We have shown the filter applied to (x, u) for clarity, but it is trivial to extend it to the full 4D light field. Note that we do not store the irradiance and layer opacity functions explicitly, but directly accumulate the sums corresponding to the integrals in Eqs. (25) and (40).

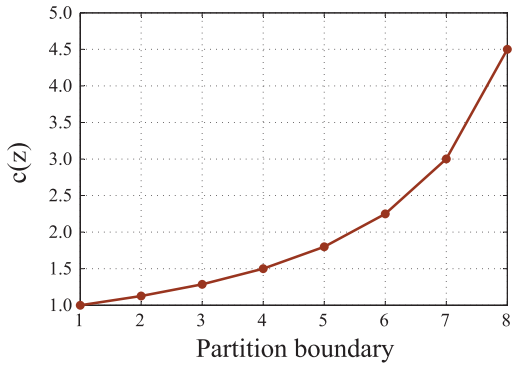

Fig. 10. Circle-of-confusion $c(z)$ values at each partition boundary for a certain depth range behind the focal plane. These partitions have been selected such that the worst-case filter energy for each partition is constant. Partitions in front of the focal plane are similar, but with negative values of $c(z)$.

We can see our method as collapsing the (x, y, u, v) light field into two images as seen from a pin-hole camera for each depth layer, wherein each pixel holds the integrated radiance and opacity values, respectively. Finally, these pin-hole images are composited together using Eq. (35). Note that, in practice, the opacity values can be stored in the alpha component of an RGBA image.

6.2 Selecting Depth Partitions for Layers

We use a simple partitioning scheme with precomputed depth partitions, where the worst-case filter energy for each layer is approximately the same. To simplify our analysis, partitions are derived in terms of circle of confusion $c(z)$ instead of depth.

From Eq. (17), we can see that the energy of our reconstruction filter depends on $|\mathbf{M}| = \sigma_x\sigma_u$, therefore we can select partitions such that the worst-case value of $\sigma_x\sigma_u$ is approximately constant.

For a given value of $\sigma_x\sigma_u$, such partitions can be easily derived using Eq. (16), where c_{min} and c_{max} are given by the circle of confusion at the partition boundaries. The number of layers depends on the selected value of $\sigma_x\sigma_u$, where a larger value results in partitions with narrower depth ranges and therefore produces more layers.

Figure 10 shows an example set of layer partitions where the worst-case filter energy is constant. The partition size increases as we move away from the focal plane and the end partitions have infinite extents.

Note that the partition boundaries represent the worst-case values of c_{min} and c_{max}. The actual range might be smaller depending on the samples that map to a particular partition, which can be used to derive better filters. Moreover, several partitions may not contain any samples and therefore need not be processed. We optimize our algorithm for these scenarios by analyzing the input samples to determine the active partitions, as well as the actual range of $c(z)$ for each partition.

In order to avoid searching for the layer corresponding to each input sample, we divide the range of $c(z)$ into small, equally sized bins of size 0.5 pixels. A sample can then be directly mapped to a bin by quantizing the value of $c(z)$ for this sample. If one or more samples map to a bin, then the bin is marked as active. After all the input samples have been processed, the active bins are analyzed to determine the active layer and the corresponding shear range.

Since the number of bins is small compared to the number of input samples, the cost of analyzing input samples is significantly reduced with this approach. Note that, since each bin has a size of

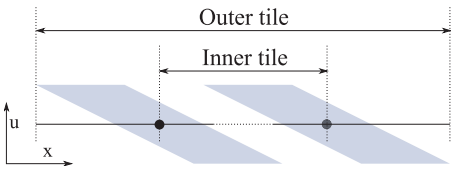

Fig. 11. An inner tile and the filter footprints at the corners of the inner tile. The filter footprints can cover samples outside the inner tile. An outer tile includes these samples. Adjacent outer tiles can overlap each other.

Table I. Number of Bits Allocated for Output Terms

Stage	I_e Red	I_e Blue	I_e Green	I_α	n
Pre-Integration	16	16	16	8	8
2D Filter	11	10	10	16	16

half a pixel, the partition boundaries are also quantized to multiples of 0.5 pixels. For our implementation, we use 30 precomputed partitions with 15 partitions on either side of the focal plane.

6.3 Tiling and Precision

The depth distribution of samples in the entire scene can span a wide range of values, requiring a large number of partitions. To address this problem and to facilitate parallel implementations of our method, we divide the samples into tiles of size 32×32 pixels and process each tile independently. Because we use a sheared reconstruction filter, samples from outside a tile can contribute to the filtered result inside a tile. We therefore add a border to define an *outer tile*, which includes additional samples from neighboring inner tiles required for analysis and filtering, as shown in Figure 11.

We use an outer tile size of 64×64 pixels, which provides an additional ring of 16 pixels around the inner tile, therefore, the maximum filter width that can be supported is 32×32 pixels. To limit artifacts from filter truncation, we clamp σ_x to a quarter of the maximum filter width, that is, 8 pixels.

For the GPU implementation, we use shared local memory to store the intermediate results from the filtering steps for faster access. This also allows us to leverage local integer atomic add operations on the GPU to implement the preintegration step, where samples are projected to the center of the lens and accumulated on a 2D grid. For this purpose, some of the terms are stored with reduced precision, which enables better utilization of the available shared local memory by packing multiple terms into a single 32-bit word. Table I shows the number of bits used to store different terms at the output of each stage. We find that using reduced precision does not result in any perceivable differences compared with the floating point version. Figure 12 shows a comparison of images reconstructed using the floating point CPU version and the GPU version that uses reduced precision for storing intermediate terms.

The overall 4D reconstruction algorithm can now be summarized as shown in Algorithm 2.

7. RESULTS

We evaluate the reconstruction quality with our algorithm based on layered reconstruction (OUR), Lehtinen et al.'s [2011] temporal light field reconstruction (TLFR), as well as a modified version of our algorithm where the sheared filters for each layer are replaced with axis-aligned reconstruction filters based on Mehta et al.'s work [2012] (OURAA). The reconstruction algorithms are implemented strictly as a-posteriori techniques and therefore do not rely on adaptive sampling. We do not directly use the approach of

Fig. 12. A comparison of reconstructed images using the floating point CPU version (top left) and the GPU version with intermediate results stored as reduced precision integers (top right). There are no perceivable differences between the two images (bottom left). Some differences can be seen after scaling the difference 16 times (bottom right).

ALGORITHM 2: Summary of the 4D reconstruction algorithm. The final irradiance is given by e.

```
for all tiles on the screen do
    Detect active partitions and range of c(z).
    e = (0, 0, 0), v = 1
    for all layers j from front to back do
        Apply Algorithm 1 to layer j.
        e += v e_j                    ▷ Composite using Equation 35.
        v *= 1 − ᾱ_j
    end for
end for
```

Mehta et al. [2012], as it relies on a-priori analysis of the light field and adaptive sampling to handle regions with complex occlusion. Without adaptive sampling, the image quality with Mehta et al.'s approach can be quite poor, as shown in Figure 13.

Our analysis is limited to a simplified camera model with a Gaussian aperture function for the lens, therefore we do not explore more realistic camera effects like bokeh. The Gaussian aperture function is truncated to a finite range of $\pm 3\sigma_u$ for generating lens samples.

For axis-aligned filtering, we first integrate input samples over the lens and then apply an axis-aligned Gaussian filter in screen space. Similar to Mehta et al. [2012], we scale the axis-aligned filter by a constant value $\frac{1}{k}$ in order to preserve the tail of the spectral response of the Gaussian lens. We use $k = 2$ for our comparisons as we observe significant overblurring in the reconstructed image with larger values of k.

The input sample information for all the reconstruction algorithms consists of the sample position (x, y, u, v), color, and depth z, rendered with 8 samples per pixel at an image size of 1280×720 pixels. The input color values are tone mapped and then converted to 24-bit color (8 bits per component).

For images reconstructed with TLFR, the light field information is upsampled to 128 samples per pixel. We do not use importance sampling over the lens for generating input samples for reconstruction. However, importance sampling is used for generating reference images, as well as during upsampling with TLFR.

Fig. 13. A region with complex visibility rendered with 8 samples per pixel and reconstructed using an axis-aligned filter based on Mehta et al.'s work [2012] (top left) and with a modified version (top right) that incorporates depth partitioning and handles inter-layer visibility using our algorithm. Sheared filtering (bottom left) further improves the reconstruction quality and compares well with a reference image (bottom right) rendered with 256 samples.

Our evaluation is based on three test scenes: CITADEL, SAN MIGUEL, and DRAGON. CITADEL is a scene from the Unreal SDK by Epic Games, Inc., and is representative of a typical real-time game with moderate depth complexity and texture detail. DRAGON and SAN MIGUEL are scenes included with the `pbrt` ray tracer [Pharr and Humphreys 2010]. DRAGON has low depth complexity and a high-frequency texture, while SAN MIGUEL has very high depth complexity and is representative of a scene for offline rendering.

7.1 Quality

Figure 14 shows a comparison of the image quality for each of the three test scenes using different reconstruction techniques, as well as a reference image rendered with 256 samples per pixel. Our layered reconstruction technique achieves a significant improvement in image quality and also handles regions with complex occlusion, which can be especially observed in the SAN MIGUEL scene. Figure 15 shows the different layers used to reconstruct this scene as well as the differences between the reconstructed result and a reference image.

7.1.1 *Comparison with Axis-Aligned Filtering.* An axis-aligned filter has significantly larger spectral bounds than a sheared filter and is therefore less effective at suppressing noise, as seen in the DRAGON scene. This difference is less noticeable around the shear extrema, where the shear slope is aligned with the x, y or u, v axes, such as in the heavily blurred backgrounds of Figures 14(b) and (d).

7.1.2 *Comparison with TLFR.* Compared to our technique, TLFR has slightly more residual noise in some areas with very large defocus blur, as seen in the background of Figures 14(b). This is because our algorithm can potentially use all samples in a 32×32 tile for filtering, while TLFR is limited to the 128 locally reconstructed samples.

However, the residual noise produced by TLFR has good spectral characteristics compared to our algorithm, which tends to produce

Fig. 14. The SAN MIGUEL (top), CITADEL (middle), and DRAGON (bottom) scenes reconstructed with our layered light field reconstruction algorithm. A comparison of different reconstruction techniques is also shown, including references rendered with 256 samples per pixel. The CITADEL scene is courtesy of Epic Games, Inc. © 2013, the DRAGON model is courtesy of the Stanford 3D Scanning Repository, and SAN MIGUEL was modeled by Guillermo M. Leal Llaguno (`www.evvisual.com`). The TLFR implementation is based on the source code provided by Lehtinen et al. [2011].

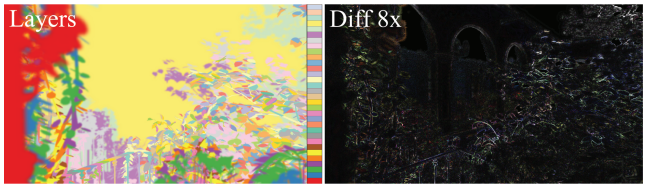

Fig. 15. The different layers in the SAN MIGUEL scene are shown on the left, where each layer has been color-coded as shown in the legend. The layers blend together seamlessly as can be seen in the image on the right, which shows the difference between the reconstructed result and a 256 spp reference. The difference image has been scaled by 8× to highlight the differences.

filtered low-frequency noise. Therefore, our algorithm may produce perceptibly lower quality than TLFR in some regions close to the focal plane, such as the floor in Figure 14(f).

TLFR tends to grow geometry around silhouette edges, which can be seen around the base of the plant stem in Figure 14(b). This is the result of errors in surface approximation, which relies on a global estimate of the sample discrepancy that may be locally incorrect. This approach is improved in Lehtinen et al. [2012], which derives a local estimate of the sample discrepancy.

7.1.3 *Visibility Approximation.* The visibility approximation introduced in Section 4 can produce slightly different results with our algorithm in partially occluded regions. Figure 14(f) shows such a case, where the lines in the partially occluded background region appear to be bent or warped in the reference image as well as the image produced with TLFR, but not in the image reconstructed with our algorithm.

This warping effect is produced due to partial occlusion of the background layer, where it is mostly visible from one corner of the lens but mostly occluded when viewed from the other corners. Therefore, the background warps to the view from this corner of the lens.

Since our algorithm approximates inter-layer occlusion using the integrated opacity (Eq. (35)), it cannot reproduce such effects that result from variations in the occlusion function with respect to different views from the lens. A more thorough analysis of the visibility approximation with specific test cases has been presented in Section 4.

7.1.4 *Partially Occluded Regions.* Partially occluded regions have fewer samples available for reconstruction as shown in Figure 9, which can affect reconstruction quality (see Figure 14(a)). This presents a difficulty for a-posteriori reconstruction techniques, but the problem can be mitigated in various ways.

For example, Shirley et al. [2011] adapt the size of the filter for a partially occluded background pixel based on the circle of confusion of the foreground layer, while Lehtinen et al. [2012] adapt their reconstruction technique based on an estimate of the local sample discrepancy. Currently, we do not adapt our reconstruction filters based on the sample distribution and leave this for future exploration.

7.1.5 *Maximum Number of Layers.* With fewer layers, the depth range inside a layer is likely to increase, resulting in less effective filters and more noise. Therefore, by selecting a smaller number of layers, we can trade off image quality for faster reconstruction times. Figure 16 shows the reconstruction quality for the DRAGON scene with a different number of average layers. With very few layers, the reconstruction filter might become ineffective in tiles having a large depth complexity, which can make the transition between tiles more visible. This is seen in the region around the tail of the dragon in Figure 16 (right).

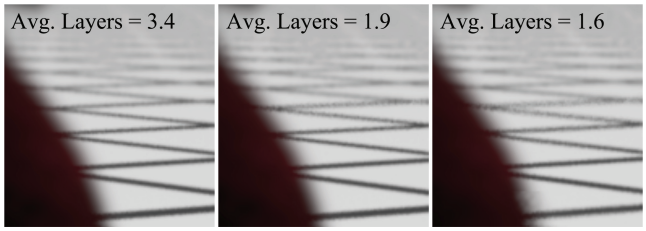

Fig. 16. Reconstructed images with a decreasing number of average layers. The corresponding execution times on the CPU are 7.5, 5.1, and 4.7 seconds respectively.

Table II. GPU Platforms

Platform	Form Factor	GPU	TDP
GPU A	Discrete	NVIDIA Geforce GTX 680	195 W (GPU)
GPU B	Integrated	Intel Iris Pro 5200	47 W (CPU+GPU)

Table III. Comparison of Execution Time in ms

	TLFR		OUR			
Scene	CPU	GPU A	CPU	GPU A	GPU B	Avg. Layers
DRAGON	73,920	12,721	7,472	31.1	131.2	3.5
CITADEL	74,963	14,023	9,130	39.5	150.8	4.8
SAN MIGUEL	113,907	22,333	14,072	75.9	269.9	9.5

7.2 Performance

We present performance results for a single-threaded CPU implementation of our algorithm, as well as a GPU implementation based on Microsoft Direct3D 11 compute shaders. Both these implementations are compared to the original multithreaded CPU and CUDA implementations of TLFR.

All CPU performance data is captured on a 6-core, 12-thread Intel i7-3960X CPU running at 3.3 GHz on a Windows 7 64-bit machine with 32GB of main memory. The performance of our GPU implementation is measured on two different platforms, GPU A and GPU B. These platforms have different power profiles as listed in Table II. Performance of the CUDA implementation of TLFR is measured only on GPU A, which is a CUDA-capable GPU.

Table III shows a comparison of execution times for the three test scenes on the CPU as well as GPU platforms. Our algorithm achieves real-time performance for typical scenes on GPU A and is more than two orders of magnitude faster than TLFR.

Out of the three test scenes, SAN MIGUEL has the longest reconstruction time due to its high depth complexity, which results in more active layers per tile. Table IV shows execution times for the different steps in our algorithm, measured on GPU A. The execution time is dominated by the preintegration and 2D filtering steps. The 2D filter can possibly be further optimized using fast approximate Gaussian filtering techniques [Akenine-Möller et al. 2008].

8. CONCLUSION

Effective reconstruction filters can go a long way in making light field sampling practical and present an exciting opportunity for enabling stochastic rendering in future real-time graphics pipelines. We have presented a new reconstruction technique for sparsely

Table IV. Execution Time for Different Steps in ms

Algorithm step	DRAGON	CITADEL	SAN MIGUEL
Select partitions	0.6	0.65	0.76
Pre-integration	20	25.75	49.3
2D filter, composite, etc.	10.5	13.1	25.84

sampled depth of field that, to our knowledge, is the first technique with real-time performance.

By building on previous work on sheared filtering but transforming the problem into a form much more amenable to efficient implementation, we have shown that the cost of the filtering step can be significantly reduced. We partition the light field in depth and combine sheared filtering with our novel method for estimating and filtering the opacity of each layer. This enables a very high image quality without perceivable noise, even in regions of complex occlusion. It would be interesting to explore the possibility of extending these ideas to other problems such as motion blur, soft shadows, and ambient occlusion.

ACKNOWLEDGMENTS

We thank the anonymous reviewers, Intel's Advanced Rendering Technology (ART) team, and Aaron Lefohn for all their feedback. We also thank David Blythe, Tom Piazza, and Chuck Lingle for supporting this research, and Uzi Sarel and Nir Benty for their help. The CITADEL test scene is courtesy of Epic Games, Inc. © 2013, the DRAGON model is courtesy of the Stanford 3D Scanning Repository, and SAN MIGUEL was modeled by Guillermo M. Leal Llaguno (`www.evvisual.com`). We are also grateful to Lehtinen et al. [2011] for sharing the source code for their implementation.

REFERENCES

T. Akenine-Möller, E. Haines, and N. Hoffman. 2008. *Real-Time Rendering* 3rd Ed. AK Peters Ltd.

T. Akenine-Möller, J. Munkberg, and J. Hasselgren. 2007. Stochastic rasterization using time-continuous triangles. In *Proceedings of the 22nd ACM SIGGRAPH/EUROGRAPHICS Symposium on Graphics Hardware (GH'07).* 7–16.

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

J.-X. Chai, X. Tong, S.-C. Chan, and H.-Y. Shum. 2000. Plenoptic sampling. In *Proceedings of the 27nd Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH'00).* ACM Press, New York, 307–318.

J. Demers. 2004. Depth of field: A survey of techniques. In *GPU Gems*, R. Fernando, Ed., Addison-Wesley, 375–390.

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

K. Egan, F. Durand, and R. Ramamoorthi. 2011a. Practical filtering for efficient ray-traced directional occlusion. *ACM Trans. Graph.* 30, 6, 180:1–180:10.

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

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

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

E. Hammon Jr. 2007. Practical post-process depth of field. In *GPU Gems 3,* H. Nguyen, Ed., Addison-Wesley, 583–606.

P. S. Heckbert. 1989. Fundamentals of texture mapping and image warping. M.S. thesis, University of California, Berkeley. https://www.cs.cmu.edu/∼ph/texfund/texfund.pdf.

N. K. Kalantari and P. Sen. 2013. Removing the noise in monte carlo rendering with general image denoising algorithms. *Comput. Graph. Forum* 32, 2, 93–102.

C. Kolb, D. Mitchell, and P. Hanrahan. 1995. A realistic camera model for computer graphics. In *Proceedings of the 22nd Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH'95).* ACM Press, New York, 317–324.

D. Lanman, R. Raskar, A. Agrawal, and G. Taubin. 2008. Shield fields: Modeling and capturing 3D occluders. *ACM Trans. Graph.* 27, 5, 131:1–131:10.

S. Lee, E. Eisemann, and H.-P. Seidel. 2010. Real-time lens blur effects and focus control. *ACM Trans. Graph.* 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 Trans. Graph.* 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 Trans. Graph.* 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 Symposium on Interactive 3D Graphics and Games (I3D'13).* ACM Press, New York, 119–128.

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

M. Pharr and G. Humphreys. 2010. *Physically Based Rendering: From Theory to Implementation* 2nd Ed. Morgan Kaufmann, San Fransisco.

T. Porter and T. Duff. 1984. Compositing digital images. In *Proceedings of the 11th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH'84).* ACM Press, New York, 253–259.

M. Potmesil and I. Chakravarty. 1981. A lens and aperture camera model for synthetic image generation. In *Proceedings of the 8th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH'81).* ACM Press, New York, 297–305.

R. Ramamoorthi, J. Anderson, M. Meyer, and D. Nowrouzezahrai. 2012. A theory of Monte Carlo visibility sampling. *ACM Trans. Graph.* 31, 5, 121:1–121:16.

F. Rousselle, C. Knaus, and M. Zwicker. 2012. Adaptive rendering with non-local means filtering. *ACM Trans. Graph.* 31, 6, 195:1–195:11.

P. Sen and S. Darabi. 2012. On filtering the noise from the random parameters in Monte Carlo rendering. *ACM Trans. Graph.* 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 Symposium on Interactive 3D Graphics and Games (I3D'11).* ACM Press, New York, 9–14.

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

X. Yu, R. Wang, and J. Yu. 2011. Real-time depth of field rendering via dynamic light field generation and filtering. *Comput. Graph. Forum* 29, 7, 2099–2107.

Received October 2014; revised August 2014; accepted September 2014