A Fast Filter for Physically-Based Rendering

Brandon Wang
Ravi Ramamoorthi, Ed.
James O'Brien, Ed.

Electrical Engineering and Computer Sciences
University of California at Berkeley

Technical Report No. UCB/EECS-2013-118
http://www.eecs.berkeley.edu/Pubs/TechRpts/2013/EECS-2013-118.html

May 31, 2013

Contents

Acknowledgements

I'd like to thank my advisor, Ravi Ramamoorthi, for being an invaluable source of wisdom and guidance for my research project, my work in Computer Graphics, and my future career. I'd also like to thank Soham Mehta for the immeasurable work we share in the larger projects this report contributes to.

Without the guidance and knowledge of the Computer Graphics educators at UC Berkeley, namely Professor Carlo Sequin, Professor James O'Brien, and Professor Ravi Ramamoorthi, I would not have been able to find the field as amazing as I have. Thanks also to Milos Hasan, Michael Tao, Jiamin Bai, Yeon Jin Lee, Fu-Chung Huang, Florian Hecht, Dikpal Reddy, and the entire Visual Computing Lab for their guidance.

I thank the Siebel Scholars Foundation for gracefully funding my Master's year at Berkeley. I would also like to thank Adobe, Intel, NVIDIA, and Pixar for providing the equipment this work utilizes.

Finally, I'd like to thank my family and friends, for keeping me sane through the process.

Chapter 1

Introduction

Ray tracing allows for physically-accurate renderings of various phenomena, producing a high-quality image. Because of its computational complexity, ray tracing is typically reserved for offline rendering, where each image is allowed virtually unlimited time to render. Much work has been done to accelerate ray tracing to enter the interactive domain, requiring each image to be produced in less than a few seconds.

Current ray tracers can interactively produce images with only a limited subset of their possible offline effects. My work with Professor Ravi Ramamoorthi and Soham Mehta focuses on physical phenomena that are too computationally expensive to render interactively. We focus on two effects: soft shadows cast by area lights and diffuse indirect lighting, but the principles behind our work can be applied to various other phenomena.

Building on a novel frequency analysis of specific phenomena, we create a filter that will produce high-quality images, with an order of magnitude less samples from the ray tracer, compared to a traditional Monte Carlo ray tracer. Interactivity is achieved by minimizing filtering overhead, and sampling through a real-time ray tracer, such as NVIDIA's OptiX.

Chapter 2

Related Work

Much work has been done to extend real-time techniques to render complex effects. However, these methods either exhibit artifacts, make speed vs. accuracy tradeoffs, or are not interactive. Techniques for real-time shadows include extensions of Shadow Maps [1] and Shadow Volumes [2] [3]–[10]. Techniques for diffuse indirect illumination include approximate methods [11], [12], point-based methods [13]–[15], and precomputation-based methods [16], [17]. Our work converges to a high-quality ground truth result, is interactive, and requires no significant precomputation.

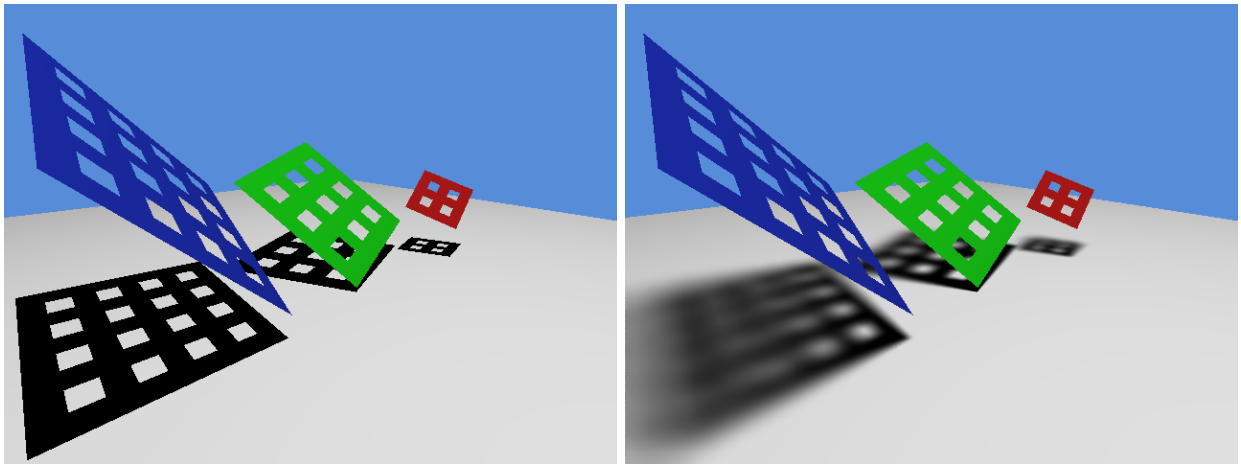
We focus on the traditionally slow, but physically-accurate ray and path tracing approach, introduced by Cook [18] and Kajiya [19]. Various methods exist to accelerate ray tracing itself [20]–[23], but our work is orthogonal to these techniques. In fact, we use our work in conjunction with the NVIDIA OptiX GPU ray tracing framework [24] to accelerate our sampling process.

Our work is most closely related to techniques that reduce the number of samples taken in ray tracing. [25]–[27]. The theoretical basis of our techniques are built on frequency analysis of shadows and light transport [28]–[36]. However, reconstruction of samples is typically slow, and is often slower than sampling itself. Our filter focuses on having a minimal computational overhead.

We operate on a noisy rendering, and essentially de-noise the render, building on [37]–[41]. Many filters in this domain are applied to Monte Carlo rendering, but add significant overhead or are not physically-accurate [42]–[47]. Similar filters have been devised for global illumination [48]–[50], but the filter we present is an image-space filter based on frequency analysis.

Chapter 3

Soft Shadows



(a) Hard Shadows (b) Soft Shadows

Figure 3.1: Hard and soft shadows.

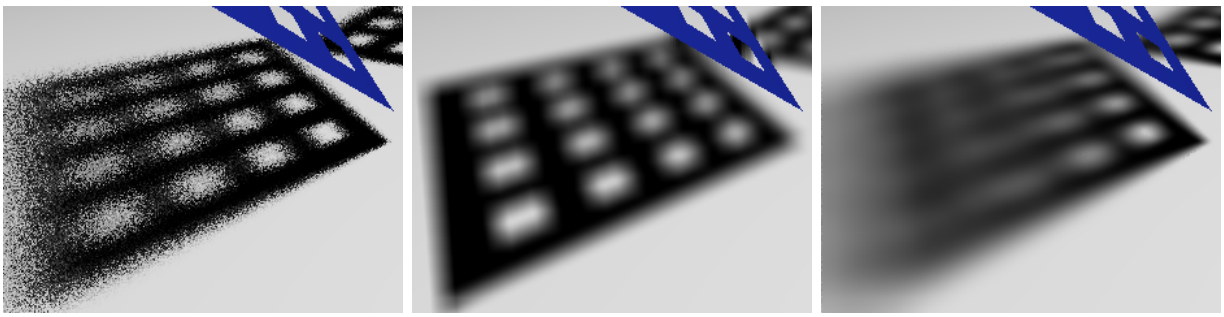
Shadows cast by objects lit by an area light exhibit regions of soft penumbra, areas of partial illumination. The result is a soft shadow, with a gradient from dark regions to light regions. As most real-world shadows do not have hard edges, soft shadows greatly add to the realism of a rendered scene.

The ray tracing solution to render a shadow is to cast a ray from the point of interest towards the light, and check for any objects that occlude the light. This works well for a point light, where only a single ray is needed. However, for area lights, the light must be sampled and the occlusion

value is integrated over the surface of the light. This requires many more rays, and increases computation time significantly.

The number of rays required for an accurate rendering depends on the scene being rendered. Large light sources may require thousands of samples for convergence. Undersampling results in high frequency noise, which greatly detracts from the perceptual accuracy of the image.

Blurring hard shadows with a uniform kernel is a common solution to removing hard shadows from renders. These blurred shadows, however, greatly differ from the physically accurate version.



(a) Undersampled Soft Shadows (b) Blurred Hard Shadows (c) Our Soft Shadows

Figure 3.2: Comparison of undersampled soft shadows, blurred hard shadows, and our physically-based soft shadows

3.1 Formulation

Focusing on area light shadows, the incoming light comes from the fraction of the area light that is unoccluded from the point we are shading. This is captured as an integration over the area of the light, with an occlusion function that is typically sampled through ray tracing.

We assume that the area light source is planar, with a Gaussian distribution, and concern ourselves with only diffuse occlusion. The occlusion is separated from the shading, and is a close approximation to the true effect.

Formally, our goal is to compute the color of a point at $\vec{x}$, $h(\vec{x})$.

$$h(\vec{x}) = r(\vec{x}) \int_S f(\vec{x}, \vec{y}) l(\vec{y}) d\vec{y} \tag{3.1}$$

$f(\vec{x}, \vec{y})$ is the binary occlusion or shadow function, which is 1 if the point $\vec{y}$ on the light is visible from $\vec{x}$, and 0 otherwise. $l(\vec{y})$ is the light source intensity, and can optionally be combined with BRDFs. S denotes the surface of the light. $r(\vec{x})$ is the irradiance from the light, approximated from the light's center, but can also contain textures. Our interest lies in the integration over the light.

3.2 Filter Formulation

(3.1), as implemented in a traditional ray tracer, treats the color at each point, $h(\vec{x})$, as an independent calculation. We notice that soft shadows are relatively low frequency, typically much lower than the rate at which the points are being sampled.

These low spatial frequency shadows are a result of integrating over the area of the light, where neighboring points will sample similar paths from the point to the light. We would like to share theses similar samples across points, but do so in an accurate manner.

Our work, presented in [51], performs a frequency analysis of area light shadows, and formulates a filter that can utilize noisy and undersampled values of $h(\vec{x})$ and reconstruct an accurate value by sharing samples across pixels. This allows for a drastic reduction in the number of samples the ray tracer is required to produce, thus significantly accelerating renders. Although we require additional computation to processes these samples, we will optimize the computation to have minimal overhead.

The reconstruction is done after integration in (3.1), allowing the filter to execute as a post-process.

The filter, ω, utilizes undersampled occlusion values, $h_n(\vec{x})$ to obtain a filtered final color $h(\vec{x})$. $h_n(\vec{x})$ is defined similarly to (3.1), but without $r(\vec{x})$, the approximated light intensity. Our filter is only concerned with the integration over the surface of the light.

$$\begin{aligned} h(\vec{x}) &= r(\vec{x}) \int h_n(\vec{x'}) w(\vec{x} - \vec{x'}, \beta(\vec{x})) d\vec{x'} \\ h_n(\vec{x}) &= \int_S f(\vec{x}, \vec{y}) l(\vec{y}) d\vec{y} \end{aligned} \tag{3.2}$$

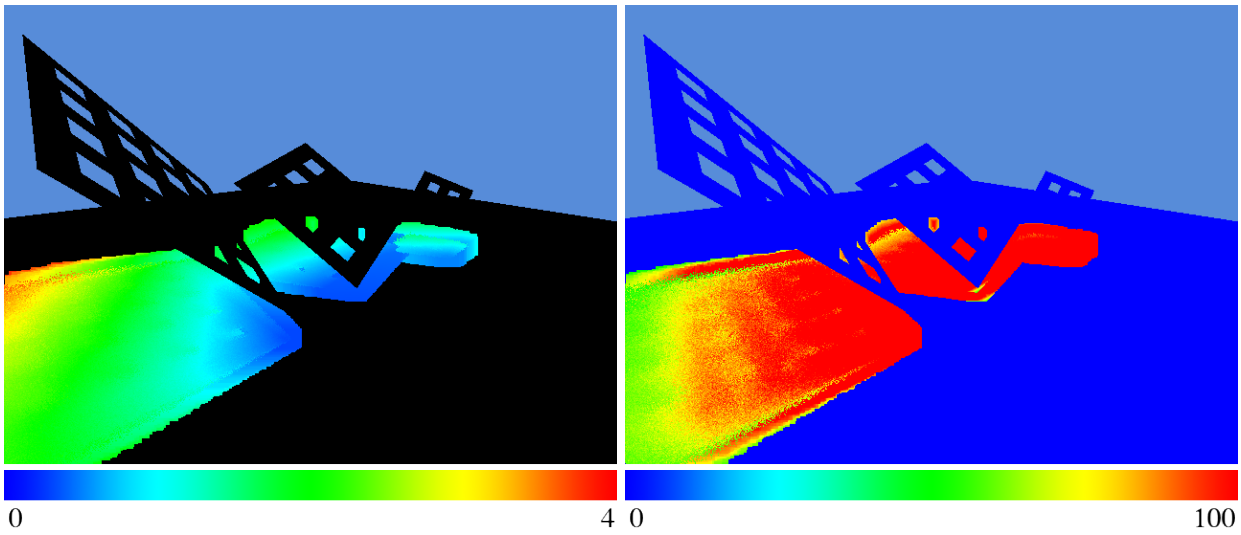
The filter is Gaussian with an adaptive distribution, depending on per-point values. This is essentially sharing samples between two different integrated occlusion values at $\vec{x}$ and $\vec{x'}$. Here, we measure distance, $||\vec{x} - \vec{x'}||$, along the plane parallel to the light.

$$w(\vec{x} - \vec{x'}, \beta) = \frac{1}{\sqrt{2\pi\beta}} \exp\left(-\frac{||\vec{x} - \vec{x'}||^2}{2\beta^2}\right) \tag{3.3}$$

The key to maintaining realistic shadows is the adaptive filter weight, β, which depends on scene properties per sampled point $\vec{x}$. We are able to share more samples between points with softer shadows, giving a wide filter radius, as can be seen in 3.3a. The full derivation of the filter weight is given in [51].

$$\beta(\vec{x}) = \frac{1}{3} max\left(\sigma\left(\frac{d_1}{d_2^{max}} - 1\right), \frac{d_1}{d_2^{max} d}\right) \tag{3.4}$$

d_1 is the distance from the receiver to the light, d_2^{max} is the maximum distance from the occluder to the light, d is the projected distance of the pixel corresponding with $\vec{x}$. σ is the standard deviation of our light, assumed to be Gaussian.



(a) Scale β (b) Samples per pixel

Figure 3.3: Visualizations of filter scale and sampling rates

The frequency analysis also allows us to define an adaptive sampling rate per pixel (spp). We sample more in regions that require more information per noisy $h_n(\vec{x})$, such as hard edges, and sample less in regions that do not require as much information, such as regions in a soft penumbra, as seen in figure 3.3b. Our sampling rate, n, is defined as:

$$\begin{aligned} n(\vec{x}) &= 4(1+\frac{s_1}{s_2})^2\left(\frac{2}{s_2}\sqrt{\frac{A_p}{A_l}}+(1+s_2)^{-1}\right)^2 \\ s_1 &= \frac{d_1}{d_2^{min}}-1 \\ s_2 &= \frac{d_1}{d_2^{max}}-1 \end{aligned} \tag{3.5}$$

A_p is the projected area of a pixel, and A_l is the area of the light source, in square meters.

3.3 Filtering Algorithm

Because the filter (3.2) operates on a noisy estimate of the incoming light, we can generate a noisy, low-sampled image and filter the image for our final result.

By filtering in image-space, we can only sample values from visible points. We lose information in regions that are not seen by the camera, but reduce our 3D world-space integration to a 2D integration. Our filter width can be converted to image-space by calculating the projected areas of each pixel. In practice, however, we found a large user-defined constant pixel value to suffice.

The algorithmic complexity of this image filter calls for a $O(n^2)$ implementation, where n is the largest dimension of our filter size. We found that this $O(n^2)$ filter added significant computational overhead, prohibiting our implementation from being interactive.

To accelerate our filter, we draw inspiration from uniform gaussian filters, which are linearly separable, converting the filter from a 2D filter to two 1D filters, giving a runtime of $O(n)$.

We can attempt to split our filter in the same manner.

$$w(\vec{x} - \vec{x'}, \beta(\vec{x})) = w(\vec{x} - \vec{x''}, \beta(\vec{x}))w(\vec{x'} - \vec{x''}, \beta(\vec{x})) \tag{3.6}$$

The split requires our $\beta(\vec{x})$ to be constant across both passes of the filter. Although this can be separated into two passes, our first pass must filter each $\vec{x'}$ with multiple values of $\beta(\vec{x})$, nullifying the performance gains from separating the filter into two passes. We notice that the filter weights of two points, $\beta(\vec{x})$ and $\beta(\vec{x'})$ vary slowly with the distance between the two points, $||\vec{x} - \vec{x'}||$, for continuous gradients of shadows. As the weight given to two samples falls off with distance, we can approximate the exact filter with one that holds similar $\beta(\vec{x})$ values.

$$\begin{aligned} w(\vec{x} - \vec{x'}, \beta(\vec{x})) &\approx w(\vec{x} - \vec{x''}, \beta(\vec{x}))w(\vec{x''} - \vec{x'}, \beta(\vec{x'})) \\ h'_n(\vec{x}) &= \int_{i_0} h_n(\vec{x'})w(\vec{x} - \vec{x'}, \beta(\vec{x}))d\vec{x'} \\ h(\vec{x}) &= \int_{i_1} h'_n(\vec{x'})w(\vec{x} - \vec{x'}, \beta(\vec{x}))d\vec{x'} \end{aligned} \tag{3.7}$$

This is our two-pass split of (3.2), into two dimensions. i_0 and i_1 are the neighboring image-space pixels in each image dimension.

This approximation is exact when the values of $\beta(\vec{x})$ are the same across sampled points. In practice, the error of using this approximated filter is very small, and almost unperceivable. As can be seen in figure 3.6, we still converge to a ground truth result, despite this approximation. [51] goes on to derive a decreasing filter width with increasing samples, guaranteeing convergence to ground truth with infinite samples. The results listed in this report, however, do not include this sample-count based filter size.

Using the pre-integration of samples allows our sampling rate to be decoupled with the memory footprint of our filter - increasing sampling rates does not increase memory usage, a prohibitive bottleneck of many reconstruction methods. By using the linearly separated filter, we can filter in mere milliseconds on a NVIDIA GTX 570 GPU. Furthermore, while sampling is dependent on scene complexity, our filter is not, allowing a minimal overhead on complex scenes.

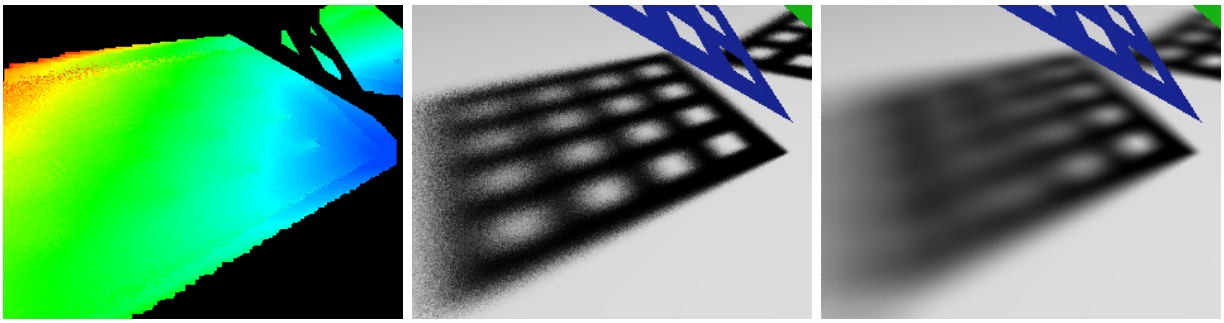
3.4 Implementation

We leveraged the use of NVIDIA's OptiX GPU-accelerated ray tracing framework. Our implementation consists of three passes:

1. **Sparse Sampling**: We first sparsely sample per-pixel occlusion. While sampling the occlusion function, we store the distances of the nearest and farthest occluders, as well as the distance to the light. This information is given when ray tracing shadows, and adds minimal overhead to a standard ray tracer.

2. **Adaptive Sampling**: Each pixel is then adaptively sampled according to (3.5). The samples are integrated to obtain noisy per-pixel occlusion values. This step is closest to a traditional Monte Carlo rendering of a scene.

3. **Filtering**: The values from the first pass are used to construct the filter widths, and the noisy occlusion values from the second pass are filtered to obtain our final occlusion values. For practical purposes, we impose a threshold on the acceptable difference in β values between two samples, to avoid large errors in using our approximate filter.



(a) Sparse sampling (b) Adaptive sampling (c) Filtering

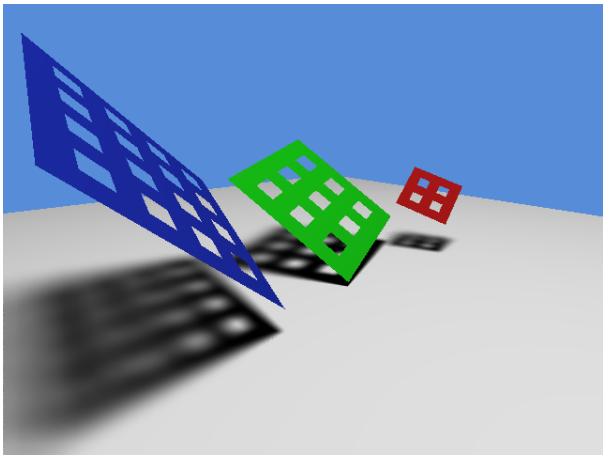
Figure 3.4: Implementation steps.

3.5 Results

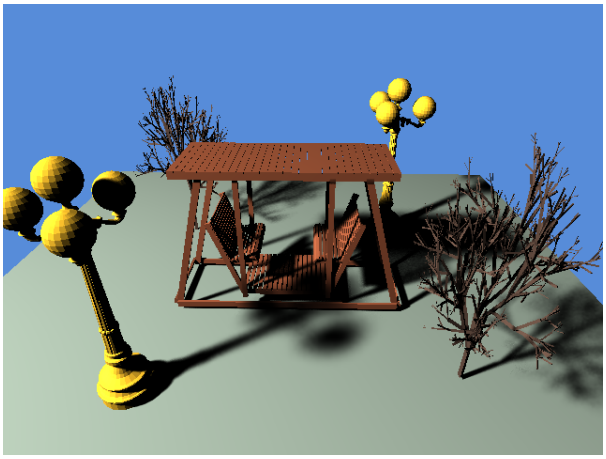
Figure 3.5 contains example scenes rendered using our method. These scenes render interactively, with timings shown in Table 3.1. Grids (figure 3.5a) is geometrically simple, but exhibits complex shadow behavior. More complex scenes, such as Bench (figure 3.5b) have a much higher sampling time, but our filter performs at a constant speed. Tentacles (figure 3.5c) and Spheres (figure 3.5d) exhibit complex shadows on curved surfaces.

We quantify the quality of our renders by taking the RMS error between our render and a ground truth image in figure 3.6. For comparison, we offer errors for renders in which we filter a non-adaptively sampled render, as well as a traditional Monte Carlo render. The filter used in these measurements utilizes the filter scale from equation (3.4). The plot is on a logarithmic scale - our method utilizing 27 average samples per-pixel compares to a Monte Carlo render with 165 samples per-pixel. Note that this filter converges towards a ground truth value, despite being approximate.

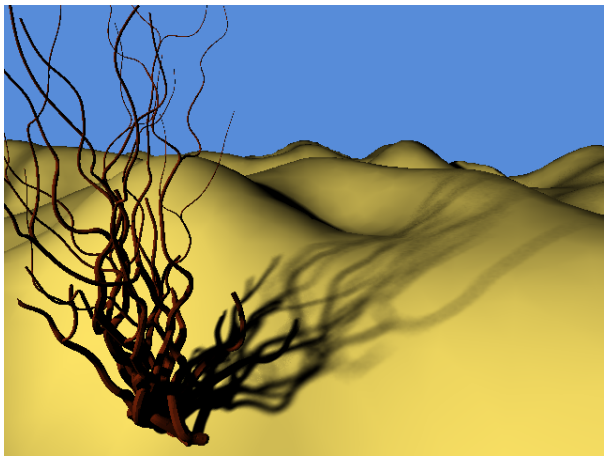
Our full implementation is open-source, and is available, along with supplementary videos at `http://graphics.berkeley.edu/papers/UdayMehta-AAF-2012-12/`.



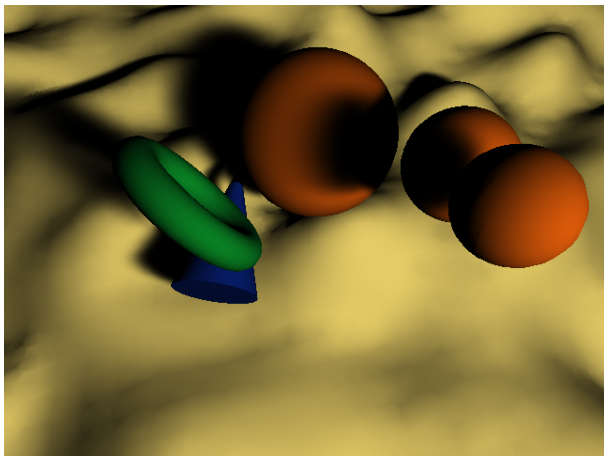
(a) Grids



(b) Bench



(c) Tentacles



(d) Spheres

Figure 3.5: Example Scenes

Scene	Vertices	Avg. SPP	Raytracing (ms)	Filtering (ms)	Total (ms)	FPS (**filtered**/unfiltered)
Grids	0.2 K	14.2	20.4	5.01	25.4	**39** / 49
Bench	309 K	28.0	425	4.78	430	**2.3** / 2.3
Tentacles	145 K	26.3	288	4.79	293	**3.4** / 3.5
Spheres	72 K	33.8	342	4.99	347	**2.9** / 2.9

Table 3.1: Timings of our scenes rendered at 640×480, using an NVIDIA GTX 570 GPU.

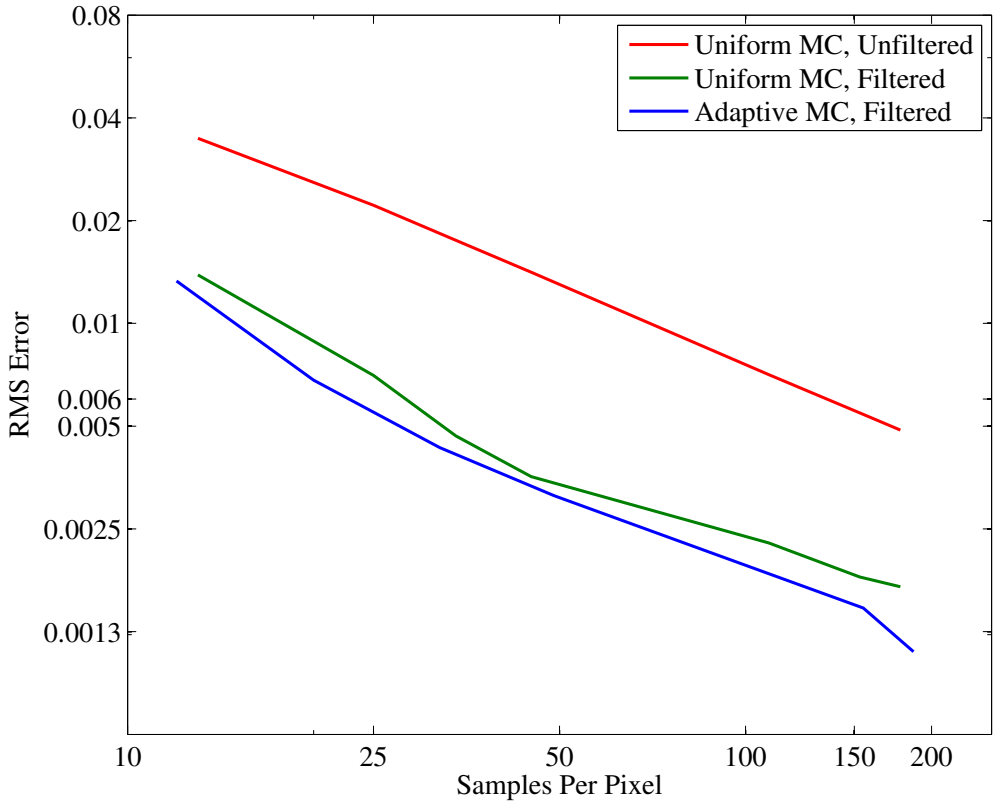
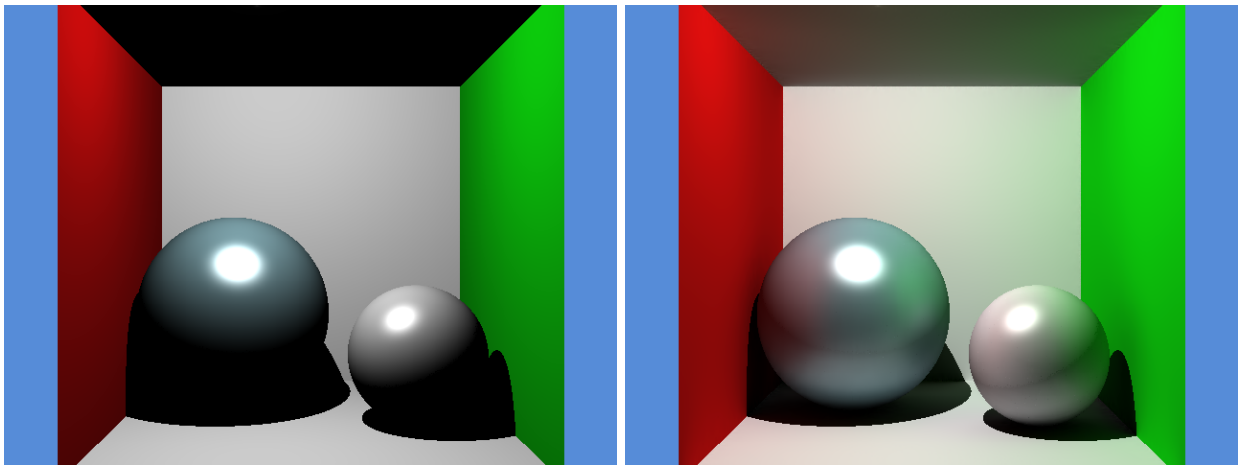


Figure 3.6: A log-log plot of the RMS pixel error vs average sampling rate for the Grids scene (figure 3.5a). Our method, shown in blue, has a consistently lower error than a traditional Monte Carlo render, shown in red.

Chapter 4

Diffuse Indirect Lighting



(a) Direct Lighting Only

(b) Direct and Indirect Lighting

Figure 4.1: Effect of diffuse indirect lighting.

We considered only incoming light directly from light sources for our area light soft shadow formulation. However, for more realistic scenes, we must also consider indirect lighting. Indirect lighting refers to the light that is reflected off a surface, and not directly from a light source. We focus on diffuse indirect lighting, the indirect light that is reflected off diffuse surfaces.

Figure 4.1 shows the added realism indirect lighting gives to a scene. The colors of the walls can be clearly seen on the spheres, and portions of shadows are illuminated.

4.1 Formulation

Our formulation is similar to the one used in soft shadows. However, for diffuse indirect lighting, we must use a path traced result, which we define recursively, similarly to [19].

$$L_o(\vec{x}, \omega_o) = \int_S L_i(\vec{x}, \omega_i) f(\omega_i, \omega_o) \cos(\omega_i) d\omega_i \tag{4.1}$$

L_o represents radiance, or emitted light, from a point $\vec{x}$ in direction ω_o. L_i refers to the light coming inward from ω_i to $\vec{x}$, taking occlusion into account. S refers to the half hemisphere aligned with the object's normal. $f(\omega_i, \omega_o)$ is the surface's BRDF.

Note that this is a recursive definition, as $L_i(\vec{x}, \omega_i) = L_o(\vec{y}, -\omega_i)$, if the closest object to $\vec{x}$ along ω_i is $\vec{y}$. This is done using recursive ray tracing, with a controlled number of allowed recursions, or bounces. Our focus is the integration of the L_i term.

4.2 Filter Formulation

As with soft shadows, (4.1) treats each point as an independent calculation. Diffuse indirect lighting effects are also typically low frequency, and we can utilize the frequency analysis of [52].

We can again filter the indirect illumination much as we did occlusion. The noisy L_{on} is filtered to produce L_o. We do not make the light intensity approximation in (3.2), and can filter (4.1) directly.

$$\begin{aligned} L_o(\vec{x}, \omega_o) &= \int w(\vec{x} - \vec{y}, \beta(\vec{x})) L_{on}(y) dy \\ L_{on}(\vec{x}, \omega_o) &= \int_S L_{in}(\vec{x}, \omega_i) f(\omega_i, \omega_o) \cos(\omega_i) d\omega_i \end{aligned} \tag{4.2}$$

Our filter is once again an adaptive Gaussian. This remains almost unchanged from (3.3). Our distance is now measured in world-space, and not along the plane parallel to a light.

$$w(\vec{x} - \vec{x'}, \beta) = \frac{1}{\sqrt{2\pi\beta}} \exp\left(-\frac{||\vec{x} - \vec{x'}||^2}{2\beta^2}\right) \tag{4.3}$$

Again, the key for filtering our effect is the adaptive filter weight, β, depending on scene properties. The full derivation of the filter weight is presented in [52].

$$\beta(\vec{x}) = \max\left(\frac{2z_{min}}{\Omega_h^{max}}, \frac{d}{0.15}\right) \tag{4.4}$$

d is the projected distance per pixel. z_{min} is the minimum world-space distance to any reflector.

The filter of integrated values of (4.2) depends not only on the frequency of incoming light, but also the BRDF of the surface, $f(\omega_i, \omega_o)$. Taking this into account, the values of Ω_h^{max} differ depending on the properties of $f(\omega_i, \omega_o)$. We present the values of Ω_h^{max} for Lambertian diffuse surfaces, $\Omega_{h,diff}^{max}$, and Blinn-Phong specular surfaces, $\Omega_{h,spec}^{max}$. [52] provides the full derivation of these values, and outlines derivations for additional surface types.

$$\begin{aligned} \Omega_{h,diff}^{max} &= 2.8 \\ \Omega_{h,spec}^{max}(m) &= 3.6 + 0.084m \end{aligned} \tag{4.5}$$

m refers to the Blinn-Phong exponent.

As with area light soft shadows, we can utilize a similar derivation for an adaptive sampling rate. This allows us to increase samples in regions with a small filter radius, and decrease samples in regions with a large filter radius.

$$n(\vec{x}) = 0.4(0.9\Omega_h^{max}\frac{\sqrt{A_p}}{z_{min}} + 0.3)^2 \cdot (\Omega_h^{max})^2 \left(1 + 0.9\frac{z_{max}}{z_{min}}\right)^2 \tag{4.6}$$

4.3 Implementation

The filtering algorithm does not change from section 3.3. The only change arises from using different parameters to determine the filter width. For scenes with specular objects, we maintain separate buffers for the diffuse and specular components lit by the indirect light, and filter them separately.

Our interactive implementation assumes that a point light illuminates a scene of Lambertian diffuse and moderately specular objects, but our equations can be directly applied to area and volume lights, with diffuse and specular receivers.

Although diffuse indirect lighting is quite a bit different than soft shadows as a physical phenomena, we have a very similar implementation to our soft shadows version, using NVIDIA's OptiX, using the same three passes. z_{min} is now sampled where d_1 and d_2^{max} were sampled, and we separate the indirect illumination calculation from the direct illumination calculation.

Indirect diffuse lighting requires more samples than area light soft shadows, integrating over a hemisphere at each point. Area light soft shadows require an integration only over the solid angle subtended by the area light. Because of this increased sample requirement, we opted for the dual-GPU NVIDIA GTX 690, accelerating our sampling steps.

A traditional ray tracer may be easily parallelized by computing each pixel separately. Our filter, however, requires the result of its neighboring pixels. For our GPU-accelerated filter to operate correctly, we copy memory between separate GPUs, increasing our overhead. Although our overhead increases from moving from a single GPU to dual GPUs, we do not expect as an significant increase in overhead in moving to more GPUs.

Our implementation copied the relevant buffers to the host CPU, which then copied a combined buffer to a single GPU. This gives us reasonable filtering times, but it can be further accelerated by using a direct GPU to GPU transfer.

4.4 Results

Figure 4.3 showcases example scenes running at interactive speeds. A version of the Cornell Box with a Stanford Dragon (figure 4.3a) clearly exhibits the effects of diffuse indirect lighting. Both walls contribute to the final color of the dragon and spheres, and also partially illuminate shadows. Sponza (figure 4.3b) and Conference (figure 4.3c) show interactive speeds on highly complex diffuse scenes. Sibenik (figure 4.3d) contains a glossy, textured ground.

We can again quantify the quality of our renders, compared to traditional Monte Carlo rendering. Once again, our method holds consistently less error at equal sampling rates, and converges towards a ground truth rendering as we increase our sample counts.

Our implementation is open-source, and is available, along with supplementary videos at `http://graphics.berkeley.edu/papers/Udaymehta-IPB-2013-07/`.

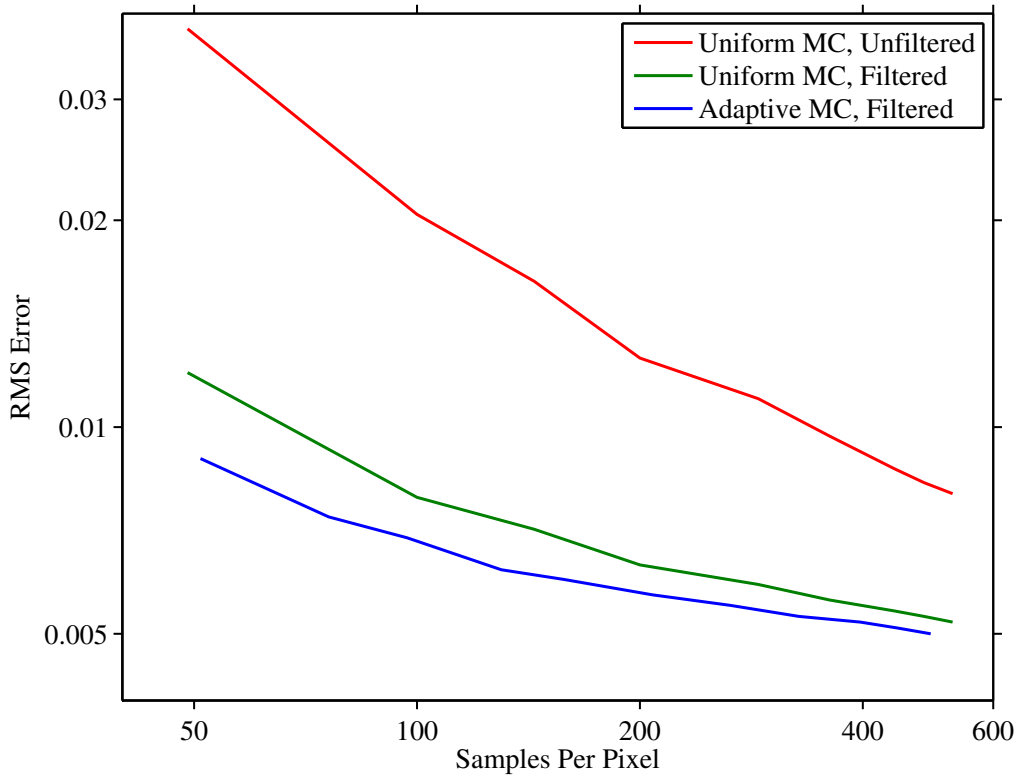
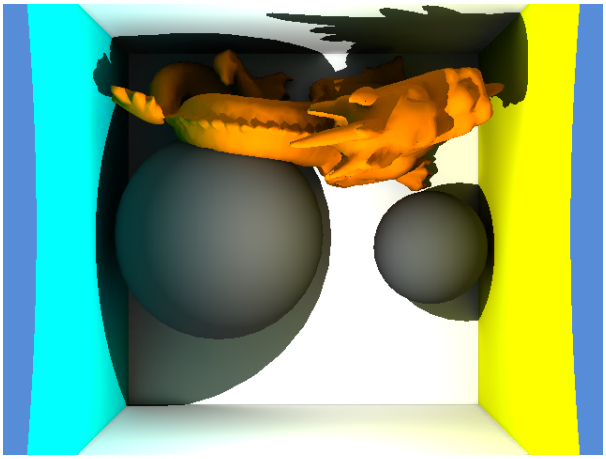
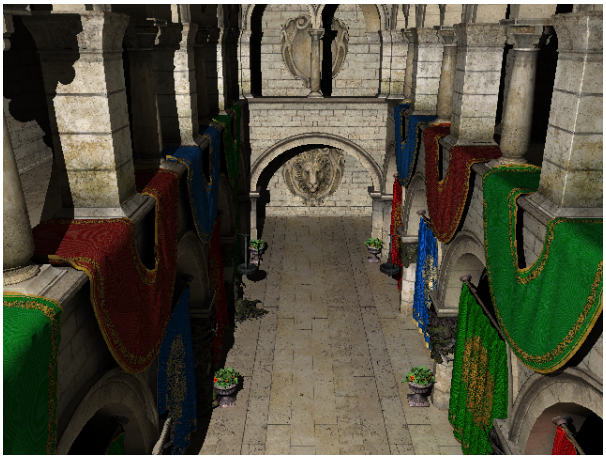


Figure 4.2: A log-log plot of the RMS pixel error vs average sampling rate for the Conference scene (figure 4.3c). Our method, shown in blue, has a consistently lower error than a traditional Monte Carlo render, shown in red.

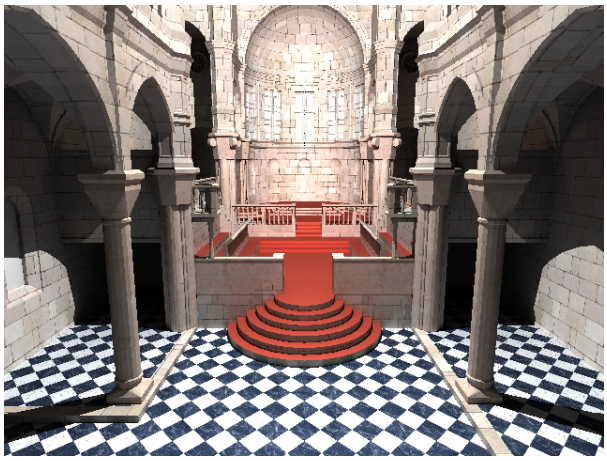


(a) Cornell Box



(b) Sponza

(c) Conference



(d) Sibenik

Figure 4.3: Example Scenes

scene	triangles	avg. spp	bounces	sampling (ms)	filtering (ms)	total (ms)	fps (**filtered**/unfiltered)
Cornell Box	16.7 K	59	1	318	61	379	**2.64** / 3.14
			2	690	67	757	**1.32** / 1.44
Sponza	262 K	63	1	761	65	826	**1.21** / 1.31
Conference	331 K	60	1	361	55	416	**2.40** / 2.77
Sibenik	75 K	86	1	550	60	610	**1.64** / 1.82
			2	1510	64	1574	**0.64** / 0.67

Table 4.1: Timings of our scenes rendered at 640×480, on a NVIDIA GTX 690.

Chapter 5

Extensions

Although soft shadows and diffuse illumination are very different physical effects, we notice that they are rather low frequency, when compared to the pixels they are being sampled at. This is because they must integrate over an area, and neighboring pixels will sample similar paths.

By performing a frequency analysis on various effects, an analytical bandlimit can be determined. Using this physical bandlimit, we can filter a noisy image to effectively share samples across neighboring pixels. Utilizing this frequency analysis, we devise a fast, constant memory filter to share samples across pixels.

Though only soft shadows and diffuse interreflections have been explicitly shown, many effects that are computationally expensive due to integration can be analyzed in the same way. Examples of straightforward extensions for which this filter can operate on with an appropriate analysis include motion blur, depth of field, and area light illumination.

Chapter 6

Conclusion

In this report, we attempt to accelerate computationally expensive renders by using a filter with a low overhead. By using our filter, we are able to generate renders of equal error to a traditional Monte Carlo render, with an order of magnitude less samples.

Our filter focuses on simplicity and efficiency, and is straightforward to implement, even in existing ray tracing systems. The key contribution of our work is the efficiency of the filter, which runs in a few milliseconds, and is invariant to scene complexity. Using the filter allows accurate, interactive renders of complex effects.

References

[1] L. Williams, "Casting curved shadows on curved surfaces," in *SIGGRAPH 78*, 1978, pp. 270–274.

[2] F. Crow, "Shadow algorithms for computer graphics," in *SIGGRAPH 77*, 1977, pp. 242–248.

[3] G. Guennebaud, L. Barthe, and M. Paulin, "Real-time soft shadow mapping by backprojection," in *EGSR 06*, 2006, pp. 227–234.

[4] ——, "High-quality adaptive soft shadow mapping," *Computer Graphics Forum*, vol. 26, no. 3, pp. 525–533, 2007.

[5] T. Annen, Z. Dong, T. Mertens, P. Bekaert, and H. Seidel, "Real-time all-frequency shadows in dynamic scenes," *ACM Transactions on Graphics (Proc. SIGGRAPH 08)*, vol. 27, no. 3, pp. Article 34, 1–8, 2008.

[6] C. Soler and F. Sillion, "Fast Calculation of Soft Shadow Textures Using Convolution," in *SIGGRAPH 98*, 1998, pp. 321–332.

[7] U. Assarsson and T. Möller, "A geometry-based soft shadow volume algorithm using graphics hardware," *ACM Transactions on Graphics (SIGGRAPH 03)*, vol. 22, no. 3, pp. 511–520, 2003.

[8] S. Laine, T. Aila, U. Assarsson, J. Lehtinen, and T. Möller, "Soft shadow volumes for ray tracing," *ACM Transactions on Graphics*, vol. 24, no. 3, pp. 1156–1165, Aug. 2005.

[9] R. Overbeck, R. Ramamoorthi, and W. Mark, "A real-time beam tracer with application to exact soft shadows," in *EGSR 07*, 2007, pp. 85–98.

[10] G. Johnson, W. Hunt, A. Hux, W. Mark, C. Burns, and S. Junkins, "Soft irregular shadow mapping: fast, high-quality, and robust soft shadows," in *I3D 2009*, 2009, pp. 57–66.

[11] T. Ritschel, C. Dachsbacher, T. Grosch, and J. Kautz, "The state of the art in interactive global illumination," *Computer Graphics Forum*, vol. 31, no. 1, pp. 160–188, 2012.

[12] C. Crassin, F. Neyret, M. Sainz, S. Green, and E. Eiseman, "Interative indirect illumination using voxel cone tracing," *Computer Graphics Forum*, vol. 30, no. 7, pp. 1921–1930, 2011.

[13] T. Ritschel, T. Engelhardt, T. Grosch, H. Seidel, J. Kautz, and C. Dachsbacher, "Micro-rendering for scalable, parallel final gathering," vol. 28, no. 5, 2009.

[14] R. Wang, R. Wang, K. Zhou, M. Pan, and H. Bao, "An efficient GPU-based approach for interactive global illumination," *ACM Transactions on Graphics*, vol. 28, no. 3, 2009.

[15] D. Maletz and R. Wang, "Importance point projection for GPU-based final gathering," *Computer Graphics Forum (EGSR 11)*, vol. 30, no. 4, pp. 1327–1336, 2011.

[16] M. Hasan, F. Pellacini, and K. Bala, "Direct to Indirect Transfer for Cinematic Relighting," *ACM Transactions on Graphics (Proc. SIGGRAPH 06)*, vol. 25, no. 3, pp. 1089–1097, 2006.

[17] P. Sloan, J. Kautz, and J. Snyder, "Precomputed Radiance Transfer for Real-Time Rendering in Dynamic, Low-Frequency Lighting Environments," *ACM Transactions on Graphics (Proc. SIGGRAPH 02)*, vol. 21, no. 3, pp. 527–536, 2002.

[18] R. Cook, T. Porter, and L. Carpenter, "Distributed Ray Tracing," in *SIGGRAPH 84*, 1984, pp. 137–145.

[19] J. Kajiya, "The Rendering Equation," in *SIGGRAPH 86*, 1986, pp. 143–150.

[20] D. van Antwerpen, "Improving SIMD efficiency for parallel monte carlo light transport on the GPU," in *High Performance Graphics*, 2011.

[21] I. Wald, C. Benthin, P. Slusallek, T. Kollig, and A. Keller, "Interactive global illumination using fast ray tracing," in *Rendering Techiques (EGWR 02)*, 2002.

[22] I. Wald, W. R. Mark, J. Günther, S. Boulos, T. Ize, W. Hunt, S. G. Parker, and P. Shirley, "State of the Art in Ray Tracing Animated Scenes," in *STAR Proceedings of Eurographics 07*, D. Schmalstieg and J. Bittner, Eds. The Eurographics Association, Sep. 2007, pp. 89–116.

[23] E. Eisemann and X. Décoret, "Visibility sampling on gpu and applications," 2007. [Online]. Available: http://maverick.inria.fr/Publications/2007/ED07a

[24] S. Parker, J. Bigler, A. Dietrich, H. Friedrich, J. Hoberock, D. Luebke, D. McAllister, M. McGuire, K. Morley, A. Robison, and M. Stich, "OptiX: A general purpose ray tracing engine," *ACM Transactions on Graphics*, vol. 29, no. 4, pp. 66:1–66:13, 2010.

[25] B. Guo, "Progressive radiance evaluation using directional coherence maps," in *SIGGRAPH 98*, 1998, pp. 255–266.

[26] T. Hachisuka, W. Jarosz, R. Weistroffer, K. Dale, G. Humphreys, M. Zwicker, and H. Jensen, "Multidimensional adaptive sampling and reconstruction for ray tracing," *ACM Transactions on Graphics*, vol. 27, no. 3, 2008.

[27] R. Overbeck, C. Donner, and R. Ramamoorthi, "Adaptive Wavelet Rendering," *ACM Transactions on Graphics (SIGGRAPH ASIA 09)*, vol. 28, no. 5, 2009.

[28] J. Chai, S. Chan, H. Shum, and X. Tong, "Plenoptic Sampling," in *SIGGRAPH 00*, 2000, pp. 307–318.

[29] R. Ramamoorthi, M. Koudelka, and P. Belhumeur, "A Fourier Theory for Cast Shadows," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 27, no. 2, pp. 288–295, 2005.

[30] F. Durand, N. Holzschuch, C. Soler, E. Chan, and F. Sillion, "A Frequency Analysis of Light Transport," *ACM Transactions on Graphics (Proc. SIGGRAPH 05)*, vol. 25, no. 3, pp. 1115–1126, 2005.

[31] D. Lanman, R. Raskar, A. Agrawal, and G. Taubin, "Shield fields: modeling and capturing 3D occluders," *ACM Transactions on Graphics (SIGGRAPH ASIA 08)*, vol. 27, no. 5, 2008.

[32] K. Egan, F. Hecht, F. Durand, and R. Ramamoorthi, "Frequency analysis and sheared filtering for shadow light fields of complex occluders," *ACM Transactions on Graphics*, vol. 30, no. 2, 2011.

[33] K. Egan, Y. Tseng, N. Holzschuch, F. Durand, and R. Ramamoorthi, "Frequency analysis and sheared reconstruction for rendering motion blur," *ACM Transactions on Graphics*, vol. 28, no. 3, 2009.

[34] C. Soler, K. Subr, F. Durand, N. Holzschuch, and F. Sillion, "Fourier depth of field," *ACM Transactions on Graphics*, vol. 28, no. 2, 2009.

[35] K. Egan, F. Durand, and R. Ramamoorthi, "Practical filtering for efficient ray-traced directional occlusion," *ACM Transactions on Graphics (SIGGRAPH Asia 11)*, vol. 30, no. 6, 2011.

[36] L. Belcour, C. Soler, K. Subr, N. Holzschuch, and F. Durand, "5D covariance tracing for efficient defocus and motion blur," *ACM Transactions on Graphics (to appear) [MIT-CSAIL-TR-2012-034]*, 2013.

[37] H. Rushmeier and G. Ward, "Energy preserving non-linear filters," pp. 131–138, 1994.

[38] H. Jensen and N. Christensen, "Optimizing path tracing using noise reduction filters," in *WSCG 95*, 1995, pp. 134–142.

[39] M. McCool, "Anisotropic diffusion for monte carlo noise reduction," *ACM Transactions on Graphics*, vol. 18, no. 2, pp. 171–194, 1999.

[40] R. Xu and S. Pattanaik, "A novel monte carlo noise reduction operator," *IEEE Computer Graphics and Applications*, vol. 25, no. 2, pp. 31–35, 2005.

[41] M. Meyer and J. Anderson, "Statistical acceleration for animated global illumination," *ACM Transactions on Graphics*, vol. 25, no. 3, pp. 1075–1080, 2006.

[42] P. Sen and S. Darabi, "On filtering the noise from the random parameters in monte carlo rendering," *ACM Transactions on Graphics*, vol. 31, no. 3, 2012.

[43] K. Dabov, A. Foi, V. Katkovnik, and K. Egiazarian, "Image denoising by sparse 3D transform-domain collaborative filtering," *IEEE Transactions on Image Processing*, vol. 16, no. 8, pp. 2080–2095, 2007.

[44] J. Lehtinen, T. Aila, J. Chen, S. Laine, and F. Durand, "Temporal light field reconstruction for rendering distribution effects," *ACM Transactions on Graphics*, vol. 30, no. 4, 2011.

[45] J. Lehtinen, T. Aila, S. Laine, and F. Durand, "Reconstructing the indirect light field for global illumination," *ACM Transactions on Graphics*, vol. 31, no. 4, 2012.

[46] T. Li, Y. Wu, and Y. Chuang, "SURE-based optimization for adaptive sampling and reconstruction," *ACM Transactions on Graphics (SIGGRAPH Asia 2012)*, vol. 31, no. 6, 2012.

[47] F. Rouselle, C. Knaus, and M. Zwicker, "Adaptive rendering with non-local means filtering," *ACM Transactions on Graphics (SIGGRAPH Asia 2012)*, vol. 31, no. 6, 2012.

[48] P. Shirley, T. Aila, J. Cohen, E. Enderton, S. Laine, D. Luebke, and M. McGuire, "A local image reconstruction algorithm for stochastic rendering," in *ACM Symposium on Interactive 3D Graphics*, 2011, pp. 9–14.

[49] P. Bauszat, M. Eisemann, and M. Magnor, "Guided image filtering for interactive high quality global illumination," *Computer Graphics Forum (EGSR 11)*, vol. 30, no. 4, pp. 1361–1368, 2011.

[50] H. Dammertz, D. Sewtz, J. Hanika, and H. Lensch, "Edge-avoiding a-trous wavelet transform for fast global illumination filtering," in *High Performance Graphics (HPG)*, 2010, pp. 67–75.

[51] S. Mehta, B. Wang, and R. Ramamoorthi, "Axis-aligned filtering for interactive sampled soft shadows," *ACM Transactions on Graphics (SIGGRAPH Asia 12)*, vol. 31, no. 6, 2012.

[52] S. Mehta, B. Wang, R. Ramamoorthi, and F. Durand, "Axis-aligned filtering for interactive physically-based diffuse indirect lighting," *ACM Transactions on Graphics (SIGGRAPH 13)*, vol. 32, no. 4, 2013.