Real-time Hardware-accelerated Relighting with Approximate Indirect Illumination

Milos Hasan Fabio Pellacini Kavita Bala

July 9, 2005

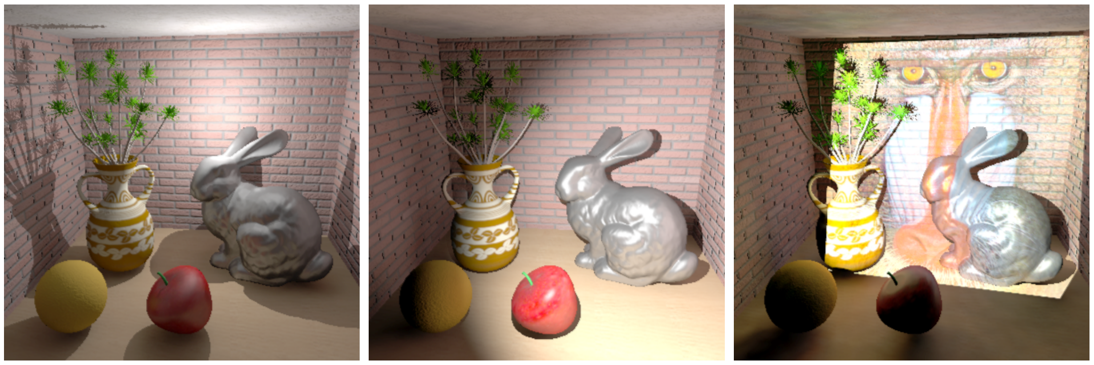


Figure 1: A scene rendered interactively with one light of different types, showing both direct and indirect illumination.

Abstract

Deep framebuffer relighting engines are often used to speed up lighting design in geometrically-complex procedurally-shaded environments where they provide interactive feedback on changes to the direct illumination. This paper presents an extension to these algorithms by providing real-time feedback for one-bounce indirect illumination. This is achieved by relighting a set of cached gather samples generated from the original geometry using a Monte Carlo gathering approach. To improve performance and decrease storage, the gather samples are clustered such that the resulting data structures are efficient for evaluation on modern GPUs. The hardware-accelerated implementation of our algorithm achieves real-time performance and is scalable to environments with high geometric and material complexity while supporting arbitrary direct lighting models, including local ones, and diffuse and glossy materials.

1 Introduction

Lighting design is a fundamental aspect of synthetic image generation, where artists carefully set the parameters of each light to obtain a desired look. This process is often limited by the time required to render images after each change of the lights. This is particularly true for high-quality imagery generated by rendering geometrically-complex environments where real-time rendering cannot be achieved.

To alleviate this problem, various relighting engines have being introduced in the literature [SS89] [BP96] [GH00] and implemented in commercial systems. These algorithms trade-off higher latency of camera movement for increased throughput in image refresh. When lighting models are based on direct illumination, this is achieved by caching geometric and material information for the current view in a deep framebuffer data structure [ST90] and reevaluating the lighting computations upon changes to illumination parameters. These algorithms not only allow for fast user feedback, but can be easily adopted by artists since they support arbitrary direct lighting models, often specified using shaders, as well as scenes of high geometric complexity.

The main drawback of these previous approaches is their inability to cope with indirect illumination that is becoming increasingly important for lighting design in various applications of computer graphics, even in the case of high-complexity environments [TL04]. Our goal is to extend fixed-view deep framebuffer-based relighting algorithms to approximate indirect illumination simulations suitable for lighting design. In particular our approach provides a real-time algorithm for one-bounce indirect illumination computed from procedurally-lit high-complexity environments; this is a particularly important application since procedural direct lighting models are the most often used models in lighting design and since a one bounce solution was proven to be efficient and controllable in the context of complex animations [TL04]. In addition our algorithm supports diffuse and glossy materials lit with various types of lights, such as local point and spot lights, as well as more complex local lighting models that could be authored using shaders, thus making our approach suitable for lighting design in artistic settings.

In order to achieve our goal, we propose to augment the data stored in each pixel, called *view sample*, of a deep framebuffer with an additional set of samples, called *gather samples*, generated by sampling the hemisphere of the corresponding view sample using a Monte Carlo approach. The gather samples contain position, normal and material properties of points of the scene seen by the hemisphere of each visible position. After each light movement, the direct illumination of the gather samples is evaluated and accumulated to estimate the indirect illumination of each view point. Unfortunately, the number of gather samples required to provide a high quality indirect illumination solution is so high that it becomes impossible to shade them at interactive speed. Our algorithm addresses this issue by reducing the number of gather samples thanks to a combination of two clustering algorithms as well as by mapping all lighting computations to a GPU implementation.

By combining a deep framebuffer approach with clustered gather samples, our hardware-accelerated implementation is capable of achieving real-time feedback for changes to the indirect illumination by arbitrary modifications of the parameters of procedurally-shaded lights for a scene of high complexity as shown in the front figure.

2 Related Work

We briefly review related work in several areas: precomputed radiance transfer (PRT)-style approaches, relighting engines, and interactive global illumination.
Precomputed radiance transfer approaches support relighting from distant sources in the form of environment maps by representing and rendering pre-computed shading in basis functions such as spherical harmonics [SKS02, KSS02, SHHS03], and wavelets [NRH03, NRH04]. Recent work [AKDS04] eliminates the restriction of distant lighting by supporting local lighting but only for one object. Extending the approach to a whole complex scene might be possible, but it does not follow trivially.
Interactive global illumination approaches cache and recompute shading when changes are made to scenes [BWG03, DBMS02, TPWG02, WKB*02]. Camera motion and object motion are well supported in these systems. However, relighting requires significant shading recomputation; therefore, artifacts due to slow shading updates are typically visible in these systems until the change propagates.
Relighting engines, as described earlier, store deep framebuffer information to recompute shading for dynamic scenes [GH00, SS89, BP96]. These techniques store geometry and material information to permit fast recomputation of shading. However, due to the amount of information storage required, these techniques are limited to direct illumination rendering only (with some limited indirect illumination in [BP96] for a Whitted style ray tracer).

3 Algorithm

3.1 One-Bounce Monte Carlo Solution

The goal of our algorithm is to evaluate the lighting values $C_i = D_i + I_i$ for the set of *view points* v_i, visible in the current view, considering the direct illumination D coming from the scene's lights as well as the indirect contribution I due to one-bounce indirect effects.

In the case of point lights, the direct illumination can be computed by summing the contribution of each light weighted by the surface response and the visibility function and can be written as

$$D_i = D(v_i) = \sum_{l=1}^{N_l} V_{il} L_{il} \rho_{il} \tag{1}$$

where N_l is the number of lights, V_{il} is the visibility function, L_{il} the light intensity and ρ_{il} the surface response evaluated for light l and surface point i. In our case, the values of L and ρ are computed using shaders, thus allowing artistic freedom in the choice of responses.

The indirect illumination coming from one-bounce interactions can be written as:

$$I_i = I(v_i) = \int_{\Omega} D(g_i)\rho_{ig_i} d\omega_{\perp}$$

where Ω is the hemisphere at v_i and g_i is the surface point visible from v_i in direction ω. This term can be evaluated using Monte Carlo integration, i.e. by accumulating the direct illumination of a set of N_g points obtained by sampling the hemisphere of v_i. When the hemispherical directions are randomly generated using a probability distribution *pdf*, we get:

$$I_i = I(v_i) \approx \frac{1}{N_g} \sum_{j=1}^{N_g} \frac{D(g_{ij})\rho_{ig_{ij}}}{pdf_{ij}} = \sum_{j=1}^{N_g} D(g_{ij}) w_{ij} \qquad (2)$$

Using this approach, an estimate of the indirect illumination for a visible point v_i is equivalent to a weighted average of direct illumination computed on a set of N_g *gather points* g_{ij}.

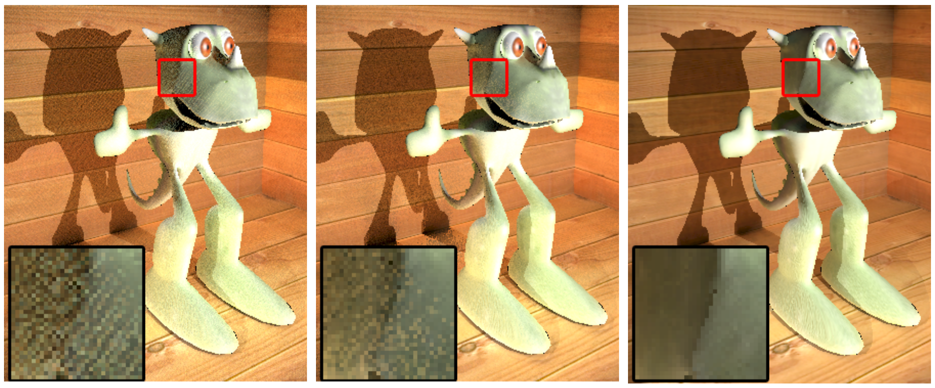
3.2 Simple Relighting Using a Monte Carlo Approach

The previous discussion leads to a conceptually simple relighting algorithm. A deep framebuffer for the main view can be used to store the position, normal and material information of each view sample along with a list of gather samples with the same point properties and associated weights. Once this data structure is populated, images can be relit by repeated evaluations of the direct illumination equation.

However, the large numbers of gather samples necessary to achieve reasonable quality make this approach impractical in terms of both storage and time complexity. For example, consider a 512 x 512 image with 400 gather samples per pixel and purely diffuse materials. In this case, the storage of position, normal and diffuse reflectance in 32-bit floating point format requires 3.6 GB.

3.3 Data Reduction Using a Clustering Approach

We propose to solve this problem by reducing the number of samples required to reconstruct indirect illumination. We do so by combining two optimizations based on clustering algorithms. First, our algorithm clusters each set of gather samples g_{ij} in order to obtain a set of smaller size $\widetilde{g_{ij}}$ but of comparable quality. To further reduce computation and storage, we will cluster the view samples themselves to choose the set of points $\widetilde{v_k}$ where the indirect contribution will be computed. The values of the indirect illumination for all view samples can



a) Unclustered b) Clustered gather samples c) Clustered gather and view

Figure 2: Comparison of images rendered with and without clustering for data sets of the same size.

then be reconstructed by averaging the clustered view samples using radial basis functions. Using these optimizations, the indirect illumination can be written as

$$I_i = I(v_i) \approx \sum_{k=1}^{N_{vc}} c_{ik} I(\widetilde{v_k}) \approx \sum_{k=1}^{N_{vc}} c_{ik} \left(\sum_{j=1}^{N_{gc}} D(\widetilde{g_{kj}}) w_{kj} \right) \tag{3}$$

where $N_{vc} < N_v$ is the number of clustered view samples, c_{ik} are their weights and $N_{gc} < N_g$ is the number of clustered gather points. The image quality obtained using these clustering algorithms is illustrated in Figure 3 where we compare three images generated respectively using all samples, clustered gather samples, and clustered view and gather samples; the storage and performance are kept roughly equivalent in all three cases.

3.3.1 Clustering Algorithm

The clustering problem of a set of samples can be stated as the task of finding k cluster centers that minimize the sum of the distances of each sample to its closest cluster center. While it is quite hard to find the global minimum of such a function, approximation algorithms perform quite well. *k-means* clustering (used e.g. in [SHHS03]) is an iterative algorithm that finds a strong local minimum by iterating the following two steps: assign each point to the nearest cluster, then recompute all cluster centers.

While the k-means algorithm has good local behavior, its results depend on the initial positions of the cluster centers. To improve global behavior, we optionally use a two-phase approach, where we initialize the k-means algorithm with cluster centers derived from a hierarchical iterative procedure that guarantees well-distributed starting positions. Our hierarchical algorithm works by iteratively splitting the cluster with largest total distance into

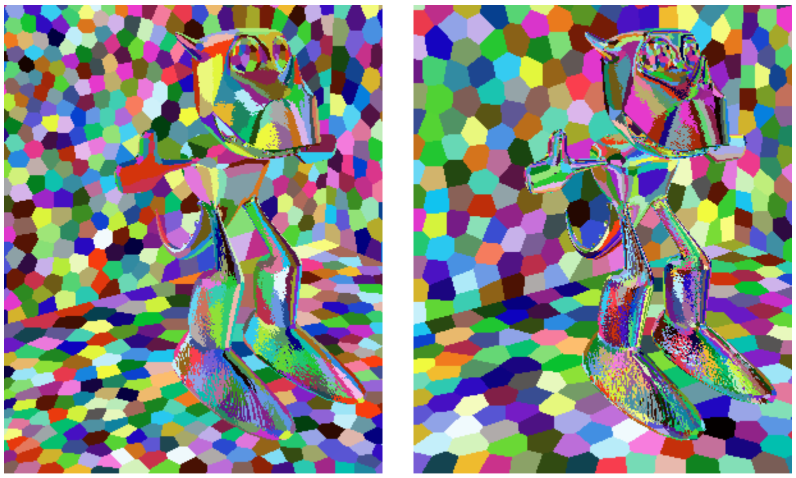


Figure 3: Comparison of clustering algorithms. The number of clusters is 1000 in both images.

two smaller clusters using the k-means procedure on the two partitions. We initialize this hierarchical procedure by assigning all points to one cluster and keep splitting until the number of desired clusters is obtained. Note that since the splitting runs the k-means algorithm for only two clusters, it is much faster than running the k-means algorithm on the whole sample set, so the speed hit is small.

3.3.2 Applying the Clustering Algorithm

We apply our two-phase clustering algorithm to the view and gather samples with a six-dimensional metric defined as the weighted sum of the L_2 distances for positions and normals. The weighting should roughly ensure that two samples with completely different normals are about as distant as two samples across the scene. The normal and material properties of the clustered gather and view samples are computed as the average of the sample values in each cluster. The position of the clustered samples is defined by picking the location of the closest point in the cluster to the center (in 3D), thus ensuring proper evaluation of the visibility function.

Figure 3 shows the clustering generated by applying our procedure to a set of view samples. The use of a 6D metric is particularly important when trying to capture surface details such as bump and displacement maps or high-complexity geometry. Figure 3 shows a comparison of random and hierarchical initialization; the latter one tends to move more clusters towards the more difficult parts of the image, even though they might take fewer pixels.

In order to handle diffuse and glossy objects, our algorithm computes two separate sets of view clusters whose size is determined by the ratio of diffuse vs specular energy visible through the camera. This allows us to reconstruct these two componentsseparately; in particular, similarly to [TPWG02], our system interpolates irradiance for the Lambertian contribution and radiance for the glossy contribution and reconstructs object textures per pixel.

4 Implementation

While the clustering optimizations presented provide us with considerable data reduction, the number of direct illumination evaluations required for high quality rendering remains quite high. To cope with this challenge, our implementation maps all relighting computation to the GPU, achieving significant speed-ups over an equivalent CPU algorithm.

As we have seen previously, the evaluation of the direct illumination of a light for one sample involves computing the light and surface response and evaluating the visibility function. In our implementation, the surface and light response are encoded in shaders written using the Cg high-level shading language [MGAK03], thus allowing us to easily change the lighting model including non-physical ones. The front figure shows various light types we have implemented including point and spot lights as well as lights that project textures; more complex lighting models such as [Bar97] can easily be added. In order to evaluate the surface response, parameters for view and clustered gather samples are packed in 16-bit floating point textures. Our view samples contain position, normal, diffuse and specular material parameters for a total of 26 bytes per sample. Our clustered gather images pack position, normal and diffuse coefficient (premultiplied by the sample weight) in 18 bytes; their direct illumination is accumulated using floating-point framebuffer blending and indexed during a reconstruction pass where coordinates and weights, 6 bytes, are stored in link images.

Visibility evaluation is currently supported using six shadow buffers packed in a floating point cube texture providing a good solution for point lights. Using shadow maps allow us to access geometry only once for the whole scene, while looking up this results multiple times during rendering. Multiple lights are supported by only updating the results of moving lights, while maintaining the accumulated results of all other lights in a separate cache.

The performance of our GPU implementation is mostly affected by two factors, the total amount of texture memory and the texture access patterns, since our computation kernels are quite small compared to the amount of memory accessed at each iteration, an operation that on GPU is typically slower than arithmetic operations [BFH*04]. The first limitation forces us to only support scenes whose caches can fit in texture memory, thus placing an upper bound on the maximum number of samples used interactively. The dependence on texture access pattern is a more interesting limitation that guided the design of our clustering algorithms; in particular we have noted that incoherent texture accesses, such as shadow map queries and dependent texture lookups used for links, can become the main source of slowdown in our GPU implementation. One algorithm we have tested previously was the use of links to directly gather illumination from a hierarchical gather cloud shared by all view samples; while this algorithm might further reduce noise, its performance on the GPU was an order of magnitude smaller than our current implementation, thus resulting in a lower quality for the same framerate. Our previous experiments as well as our current results suggest that incoherent dependent texture lookups should be minimized, thus restricting the possible relighting data structures that can be employed. In our case, we use a two-pass clustering and only allow link accesses, required to limit total memory use, to a small number of samples. Shadow map lookups presented a similar problem, since gather sample positions are very incoherent. We settled on a six-face cube map packing that appears to give better performance than other alternatives we tested.

5 Results

We tested our relighting algorithm on three scenes shown in Figure 4 on a Pentium 4 3.4 GHz with 1 GB of RAM and a 256 MB GeForce 6800 GT NVidia graphics accelerator; various statistics for each example are reported in Table 1. While Scene 1 contains only diffuse objects, scenes 2 and 3 have a variety of materials including glossy objects and bump-mapped surfaces showing that our algorithm can handle complex materials as well.

These examples were rendered at 400 by 400 image resolution using 120 clustered gather samples for each view sample generated from 2000 directions randomly selected using Quasi Monte Carlo sampling. The direct illumination is always computed at full resolution. Indirect illumination reconstruction is handled by interpolating the 10 closest clustered view samples whose total number is 40,000. These numbers were chosen to balance image quality, texture memory usage and shading speed and were found satisfactory in our test cases for most light positions. Cache sizes for these data sets are roughly 100 MB and took between 60 to 100 minutes to generate using not very optimized pre-processing code.

Our hardware-accelerated implementation is capable of reaching interactive performance even for relatively complex environments, such as the one shown in Scene 3. Its scalability with geometric complexity is only limited by the time needed to render shadow maps, which depends on the polygon count and can become significant in some cases. However, level-of-detail approaches could be used to reduce shadow rendering times, which we did not attempt. All other computation is geometry-independent. The other major component that affects the frame-rate for our algorithm is the direct shading evaluation on the gather samples.

The quality of indirect illumination generated by our system is often high for most lighting positions. Noisy artifacts can be observed at times where strong secondary illumination is created, for example with spotlights of very high intensity and very small cone width or when a point light gets very close to an object. In these cases, the visible noise is caused by the inability of our approach to capture small secondary sources, which is typical for gathering approaches. Nonetheless, our approach remains very useful even in these cases since its quality is high enough for lighting design allowing for most artistic decisions to be made interactively.

	Scene 1	Scene 2	Scene 3
Number of Triangles	1930	24716	107159
Framerate (Hz)	15	12	8
Total frametime (ms)	68	82	133
Shadows (ms)	11 (14%)	27 (33%)	76 (57%)
Direct shading (ms)	1 (2%)	1 (1%)	1 (1%)
Indirect shading (ms)	46 (66%)	46 (55%)	46 (34%)
Links resolution (ms)	8 (9%)	8 (8%)	8 (6%)

Table 1: Performance statistics for the scenes tested.

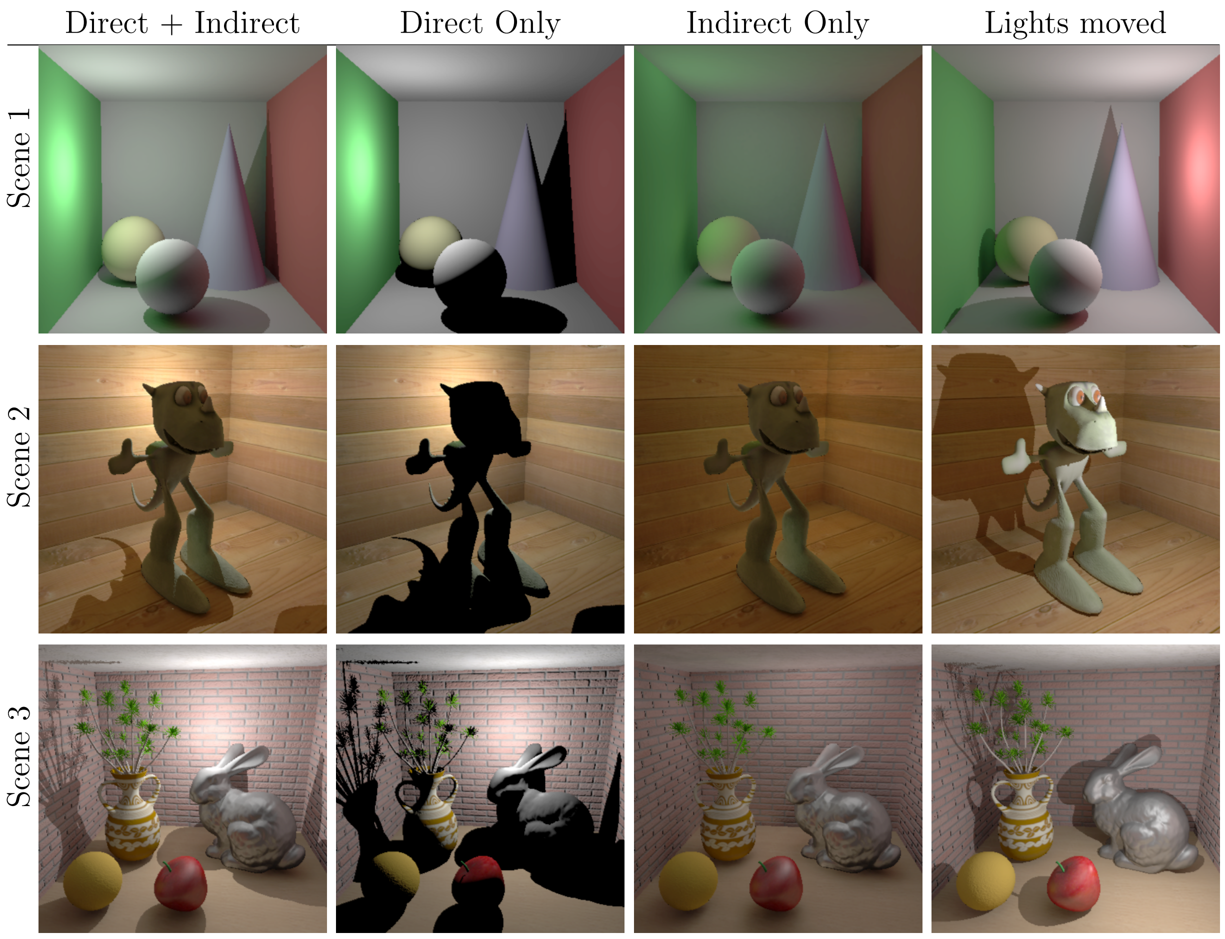


Figure 4: Final frames for three example scenes: all images are gamma corrected.

6 Conclusion

This paper presents a novel extension to deep framebuffer relighting engines capable of computing one-bounce indirect illumination in procedurally-shaded, relatively high complexity environments for a variety of material types and direct illumination models, including local ones. In these conditions relighting becomes equivalent to a large number of direct illumination computations that can be sped up considerably using clustering algorithms for the gather and view samples. Data structures generated by these algorithms make it possible for our hardware-accelerate implementation to provide high-quality feedback suitable for lighting design at interactive rates for scenes of high complexity.

In the future we would like to extend our algorithm to multiple bounces of indirect illumination and to better support strong secondary sources. It might be possible to do so by combining a real-time light tracing algorithm with gather lookups accelerated using our approach. Another possible extension would be to try to investigate how to low-pass filter the gather image signal in order to reduce noise. Each of these extensions would require the use of linked data structures that were proven inefficient in our studies thus requiring us to develop more efficient GPU data structure abstraction for handling illumination computation and reconstruction.

References

[AKDS04] Annen T., Kautz J., Durand F., Seidel H.-P.: Spherical harmonic gradients for mid-range illumination. In *Rendering Techniques 2004 Eurographics Symposium on Rendering* (June 2004), pp. 331–336.

[Bar97] Barzel R.: Lighting controls for computer cinematography. *Journal of Graphics Tools 2*, 1 (1997), 1–20.

[BFH*04] Buck I., Foley T., Horn D., Sugerman J., Fatahalian K., Houston M., Hanrahan P.: Brook for gpus: stream computing on graphics hardware. *ACM Transactions on Graphics 23*, 3 (Aug. 2004), 777–786.

[BP96] Briére N., Poulin P.: Hierarchical view-dependent structures for interactive scene manipulation. In *Proceedings of SIGGRAPH 96* (Aug. 1996), pp. 83–90.

[BWG03] Bala K., Walter B., Greenberg D.: Combining edges and points for interactive high-quality rendering. In *SIGGRAPH '03* (July 2003).

[DBMS02] Dmitriev K., Brabec S., Myszkowski K., Seidel H.-P.: Interactive global illumination using selective photon tracing. In *EGRW '02* (2002), pp. 21–34.

[GH00] Gershbein R., Hanrahan P. M.: A fast relighting engine for interactive cinematic lighting design. In *Proceedings of ACM SIGGRAPH 2000* (July 2000), pp. 353–358.

[KSS02] Kautz J., Sloan P.-P., Snyder J.: Fast, arbitrary brdf shading for low-frequency lighting using spherical harmonics. In *EGRW '02* (2002), pp. 291–296.

[MGAK03] Mark W. R., Glanville R. S., Akeley K., Kilgard M. J.: Cg: A system for programming graphics hardware in a c-like language. *ACM Transactions on Graphics 22*, 3 (July 2003), 896–907.

[NRH03] Ng R., Ramamoorthi R., Hanrahan P.: All-frequency shadows using non-linear wavelet lighting approximation. *ACM TOG 22*, 3 (2003), 376–381.

[NRH04] Ng R., Ramamoorthi R., Hanrahan P.: Triple product wavelet integrals for all-frequency relighting. *ACM TOG 23*, 3 (2004), 477–487.

[SHHS03] Sloan P.-P., Hall J., Hart J., Snyder J.: Clustered principal components for precomputed radiance transfer. *ACM Transactions on Graphics 22*, 3 (July 2003), 382–391.

[SKS02] Sloan P.-P., Kautz J., Snyder J.: Precomputed radiance transfer for real-time rendering in dynamic, low-frequency lighting environments. In *SIGGRAPH '02* (2002), pp. 527–536.

[SS89] Séquin C. H., Smyrl E. K.: Parameterized ray tracing. In *Computer Graphics (Proceedings of SIGGRAPH 89)* (July 1989), pp. 307–314.

[ST90] Saito T., Takahashi T.: Comprehensible rendering of 3-d shapes. In *Computer Graphics (Proceedings of SIGGRAPH 90)* (Aug. 1990), pp. 197–206.

[TL04] Tabellion E., Lamorlette A.: An approximate global illumination system for computer generated films. *ACM Transactions on Graphics 23*, 3 (Aug. 2004), 469–476.

[TPWG02] Tole P., Pellacini F., Walter B., Greenberg D.: Interactive global illumination. In *SIGGRAPH '02* (June 2002), pp. 537–546.

[WKB*02] Wald I., Kollig T., Benthin C., Keller A., Slusallek P.: Interactive Global Illumination using Fast Ray Tracing. In *13th Eurographics Workshop on Rendering* (2002), pp. 15–24.