Direct-to-Indirect Transfer for Cinematic Relighting

Miloš Hašan*
Cornell University

Fabio Pellacini
Dartmouth College

Kavita Bala
Cornell University

Figure 1: Two scenes rendered under different lighting configurations, relit at 10-20 fps with multiple bounces of indirect illumination. These scenes include up to 2.1 million polygons, diffuse and glossy materials and procedural light shaders.

Abstract

This paper presents an interactive GPU-based system for cinematic relighting with multiple-bounce indirect illumination from a fixed view-point. We use a deep frame-buffer containing a set of view samples, whose indirect illumination is recomputed from the direct illumination on a large set of gather samples, distributed around the scene. This *direct-to-indirect transfer* is a linear transform which is particularly large, given the size of the view and gather sets. This makes it hard to precompute, store and multiply with. We address this problem by representing the transform as a set of sparse matrices encoded in wavelet space. A hierarchical construction is used to impose a wavelet basis on the unstructured gather cloud, and an image-based approach is used to map the sparse matrix computations to the GPU. We precompute the transfer matrices using a hierarchical algorithm and a variation of photon mapping in less than three hours on one processor. We achieve high-quality indirect illumination at 10-20 frames per second for complex scenes with over 2 million polygons, with diffuse and glossy materials, and arbitrary direct lighting models (expressed using shaders). We compute per-pixel indirect illumination without the need of irradiance caching or other subsampling techniques.

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

Keywords: relighting, real-time rendering, indirect illumination, precomputed radiance transfer

1 Introduction

Lighting design in high complexity scenes is often impeded by the slow performance of rendering algorithms. This is especially true in the case of cinematic lighting that is characterized by complex environments, materials and lighting models.

Many relighting algorithms have been presented to increase speed when recomputing images for static scenes. These algorithms make different simplifications based on their target application. Cinematic quality relighting is usually performed on fixed views while trying to maintain high geometric complexity and flexible choices in materials and direct lighting models. While it has been shown that indirect illumination is desirable in many cinematic lighting tasks [Tabellion and Lamorlette 2004], current state-of-the-art cinematic relighting algorithms [Pellacini et al. 2005] do not support indirect illumination.

Figure 1 shows sample images generated with our algorithm. The main features our system supports are:

- high geometric complexity,
- diffuse and glossy materials (including sharp gloss)
- procedural light shaders, including local effects,
- multiple indirect illumination bounces,
- high resolution (per-pixel) indirect illumination,
- relatively fast precomputation.

On the other hand, our main assumptions are fixed camera and fixed scene.

We adopt a deep frame-buffer approach for relighting from a fixed view-point. The key principle of our algorithm is that the indirect illumination on the points in the deep frame-buffer, called view samples, can be computed by a linear transformation from the direct illumination on a large set of gather samples, carefully distributed in the scene. We call this *direct-to-indirect transfer*.

Given the large size of the view and gather cloud, representing, precomputing and recomputing the light transfer between these clouds requires various new techniques. We represent the direct-to-indirect transfer as a set of sparse matrices in wavelet space, using an algorithm that imposes a wavelet basis on the unstructured gather cloud. These matrices are then encoded using a sparse matrix representation particularly efficient for GPUs, which is tailored for our specific kind of sparsity. We compute the transfer matrices approximately using a hierarchical algorithm and a variation of photon mapping. The precomputation times are in the order of 3 hours for our largest scenes.

We tested our algorithm on various scenes, with up to 2.1 million polygons, diffuse and glossy materials (including sharp gloss), procedural direct lighting models expressed using arbitrary shaders,

*e-mail: {mhasan,kb}@cs.cornell.edu, fabio@cs.darmouth.edu

and four bounces of indirect illumination. In all these cases, our algorithm achieves high-quality rendering at roughly 10 to 20 frames per second. Note that we compute detailed per-pixel indirect illumination at a 640 $\times$ 480 resolution without any use of irradiance caching or other subsampling techniques.

2 Related Work

Relighting engines. Previous work on relighting engines has mostly dealt with efficient recomputation of direct illumination by caching visibility and partial shading in deep frame-buffer data structures. This was pioneered in the G-buffer [Saito and Takahashi 1990] and parameterized ray tracing [Séquin and Smyrl 1989], and extended in ray trees [Briére and Poulin 1996] to support small visibility changes. Advances in graphics hardware have allowed new relighting systems to be built by using GPUs to evaluate direct illumination; some were deployed to major productions [Gershbein and Hanrahan 2000; Pellacini et al. 2005]. Our work shares the same approach of caching visibility and shading information in a deep frame-buffer and evaluating direct illumination on GPUs as in Lpics [Pellacini et al. 2005]. Simultaneously, Tabellion [Tabellion and Lamorlette 2004] described the importance of indirect illumination for cinematic rendering in an off-line context. Our main contribution is to augment these systems with multiple-bounce indirect illumination at interactive rates.

Precomputed light transfer. Recently, much work has been dedicated to relighting scenes under distant illumination [Sloan et al. 2002; Kautz et al. 2002; Ng et al. 2003; Ng et al. 2004; Liu et al. 2004; Wang et al. 2004]. These algorithms precompute and compress the transport between distant lights and a set of scene samples, thus making it possible to recompute the illumination for changes in lighting. The main limitation of these approaches is that they only support distant lights. Some could be extended to "mid-range" lighting by taking more samples and/or computing gradients [Sloan et al. 2002; Annen et al. 2004]. However, this approximation does not fully address local lighting, where the lights can be positioned inside a complex scene. Our approach builds on these approaches (especially [Ng et al. 2003]), but rather than using distant lights, we use a set of gather samples distributed in the scene and recompute direct illumination on these samples; this allows for arbitrary local lighting models. Currently our system does not support environment mapping; we discuss a way of removing this limitation in Section 6.

[Kristensen et al. 2005] precompute and compress indirect illumination for a fixed set of local lights. When a light is moved, the indirect solutions from the closest lights are interpolated to relight the environment. This system can deliver high frame-rates with a moving camera. The drawbacks of this approach are the large precomputation times (probably several days on a single processor), the limitation to omni-directional point lights, and lower accuracy due to strong compression. Our approach is complementary to this system because we focus on supporting flexible direct lighting models and high geometric complexity, though at the price of a fixed camera assumption.

Global illumination systems. Some of the standard choices for rendering global illumination are irradiance caching [Ward et al. 1988] and photon mapping [Jensen 1996]. Lightcuts [Walter et al. 2005] renders complex illumination by renders complex illumination by converting the problem into many point lights and using a hierarchical clustering of lights; our approach also gathers light from point samples and uses a hierarchical clustering. [Wald et al. 2002] present a system based on a fast CPU ray-tracer, which supports dynamic global illumination including light movement. However, the system runs on a cluster of many processors to achieve interactive performance. Several systems cache sparse global illumination samples, allowing for interactive camera and object movement [Walter et al. 1999; Walter et al. 2002; Ward and Simmons 1999; Bala et al. 1999b; Bala et al. 1999a; Dmitriev et al. 2002; Tole et al. 2002; Bala et al. 2003; Gautron et al. 2005; Dayal et al. 2005; Gautron et al. 2005]. However, light movement in these systems is not easily handled, since it invalidates most of the cached samples.

Symbol	Description	Data Type	Size
n_v	Number of view samples	-	-
n_g	Number of gather samples	-	-
$\mathbf{v}_i$	Indirect on view samples	RGB	n_v
$\mathbf{g}_d$	Direct on gather samples	RGB	n_g
$\mathbf{k}_d$	Diffuse coefficients	RGB	n_v
$\mathbf{k}_g$	Glossy coefficients	RGB	n_v
$\mathbf{T}$	Full transfer matrix	RGB	$n_v \times n_g$
$\mathbf{F}$	Final gather matrix	RGB	$n_v \times n_g$
$\mathbf{F}_d$	Diffuse final gather matrix	$\mathbb{R}$	$n_v \times n_g$
$\mathbf{F}_g$	Glossy final gather matrix	$\mathbb{R}$	$n_v \times n_g$
$\mathbf{M}$	Multiple bounce matrix	RGB	$n_g \times n_g$
$\mathbf{W}$	Wavelet projection matrix	$\mathbb{R}$	$n_g \times n_g$
$\mathbf{I}$	Identity matrix	$\mathbb{R}$	$n_g \times n_g$

Table 1: Summary of the notation used in the paper. For matrices and vectors, we also denote the type of each element and the dimensions. In the text, we use a *w* superscript to denote a matrix (vector) that has been projected into wavelets.

Hierarchical/clustering techniques. Our problem of transporting light from gather samples to view samples could theoretically be solved by techniques based on hierarchical N-body algorithms. There is a large body of research on these techniques, starting with [Hanrahan et al. 1991]. While most of these systems do not target relighting, they could be adapted by precomputing a set of cluster-to-cluster links and reusing these when lighting changes. However, we found that our solution based on Haar wavelets gives higher performance, because it allows for a more efficient GPU implementation with on-the-fly coefficient culling, and does not require one iteration per indirect bounce. [Drettakis and Sillion 1997] present an interesting algorithm for interactive object and light movement in a radiosity context; however, it probably would not scale to the demands of cinematic relighting. Wavelet radiosity ([Gortler et al. 1993] and follow-up papers) is another related technique; however, most of this work is concerned with investigating the feasibility of higher-order wavelets and not with the performance and complexity that we focus on.

3 System overview

This section presents the details of the relighting algorithm we are proposing, illustrated in Figure 2. A summary of the notation used throughout the paper can be found in Table 1.

3.1 Assumptions

Fixed camera. We adopt a fixed-camera deep frame-buffer approach for relighting, typical of cinematic relighting algorithms. This has the advantage of better scalability with geometric complexity and allows us to support arbitrary diffuse and glossy materials in the indirect computation.

Fast direct lighting. Our algorithm assumes that computing direct illumination from point lights on sampled geometry is fast using well-known techniques (pixel shaders and shadow mapping),

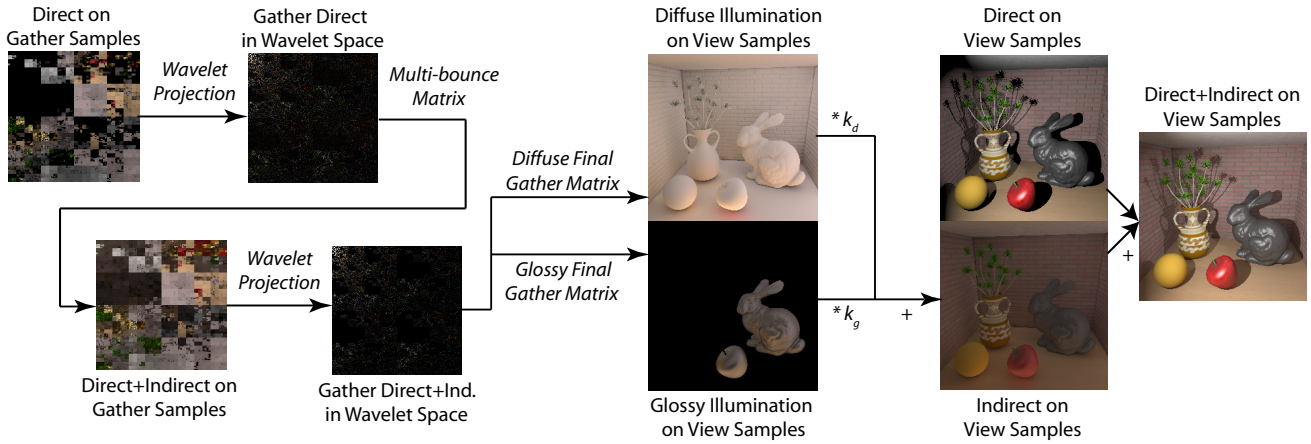


Figure 2: Overview of our relighting algorithm.

an assumption verified in [Pellacini et al. 2005]. Therefore, our focus is on recomputing the indirect illumination when a light moves. Moreover, other techniques apart from point lights could be used to compute the direct illumination, for example existing PRT techniques. In section 6 we outline a solution for including environment lighting.

Light paths. Finally, we compute the indirect illumination generated from the *diffuse direct radiance*; this obviously restricts the set of indirect paths that can be handled. Most importantly we do not capture caustic or caustic-dependent light paths. We believe those paths could be handled by a separate algorithm; however, this remains for future work.

3.2 Direct-to-Indirect Transfer Formulation

We introduce two sets of points: a set V of *view samples*, visible from the view-point, stored in the deep frame-buffer, and a set G of *gather samples* distributed around the scene. When a light moves, the direct illumination is recomputed on the view and gather samples using well-established techniques on the GPU. It is important to note that our algorithm adopts a point-based approach, rather than a patch-based one like radiosity. This gives better scalability, since the number of samples can be much lower than the number of patches in areas of low importance.

Our problem is reduced to recomputing the vector $\mathbf{v}_i$ of indirect radiances on the view samples V. Under the assumption of fixed view-point and light paths given above, this can be done by transferring the diffuse direct radiance on the gather samples through a linear transform $\mathbf{T}$ to the view samples. We can write this as:

$$\mathbf{v}_i = \mathbf{T} \cdot \mathbf{g}_d \tag{1}$$

Our goal of high-fidelity, high-resolution indirect illumination requires large view and gather clouds, typically n_v = 640 × 480 = 300k and n_g = 64k. For such resolutions, the matrix $\mathbf{T}$ is very large and difficult to precompute, store and multiply with. Thus, most of the paper focuses on achieving three goals: first, we want to find an efficient way of compressing the matrix while maintaining high quality indirect illumination; second, we want to be able to efficiently evaluate the transport expressed in the new formulation; third, we want to keep the precomputation time reasonably short.

To make the storage and multiplication tractable, we project the lighting (i.e., $\mathbf{g}_d$) and the rows of the matrix $\mathbf{T}$ into 2D Haar wavelets. and cull many of the less important wavelet coefficients to gain sparsity. A similar idea was introduced in [Ng et al. 2003] for environment map relighting; the main difference in our approach is that we impose the wavelet basis on an unstructured gather cloud.

The non-trivial problem of precomputing the transfer in reasonable time is addressed by using a two-pass approach, splitting the transfer into a lower-precision multi-bounce matrix $\mathbf{M}$ and a higher-precision final gather matrix $\mathbf{F}$, and using wavelets to gain sparsity in both. In this formulation, we can write

$$\mathbf{v}_i = \mathbf{F} \cdot (\mathbf{M} + \mathbf{I}) \cdot \mathbf{g}_d \tag{2}$$

A two-pass idea is common to many practical global illumination algorithms (photon mapping [Jensen 1996], lightcuts [Walter et al. 2005], irradiance decomposition [Arikan et al. 2005]) and used in the context of precomputed transfer from an environment map to subsurface scattering by [Wang et al. 2005].

Algorithm: Figure 2 gives an overview of our approach. When a light is moved, the direct illumination on gather samples G is recomputed using the GPU and projected into the wavelet basis. The multi-bounce transfer matrix $\mathbf{M}^w$ (transformed into wavelets and compressed) transfers this into the indirect illumination on the gather samples. Another wavelet projection is done, and the high-resolution final gather matrix $\mathbf{F}^w$ (also transformed into wavelets and compressed) is finally applied to transfer the illumination (both direct and indirect) from the gather cloud to the view cloud. For better compression, the final gather is actually split into two components corresponding to diffuse and glossy transfer. The indirect illumination is then reconstructed on the view samples by multiplying with the diffuse and glossy coefficient of each sample and combined with the direct illumination on view samples to produce the final image.

3.3 Wavelet Formulation

One-pass formulation: A general matrix-vector multiplication can be expressed in wavelet space by projecting each row of the matrix together with the vector. Formally we can write:

$$\mathbf{v}_i = \mathbf{T} \cdot \mathbf{g}_d = (\mathbf{T} \cdot \mathbf{W}^T) \cdot (\mathbf{W} \cdot \mathbf{g}_d) = \mathbf{T}^w \cdot \mathbf{g}_d^w \tag{3}$$

To obtain sparsity, the full transport matrix can be approximated by culling the less important coefficients in $\mathbf{T}^w$. This formulation makes relighting tractable by lowering the data storage and reducing computation time. Culling less important coefficients can also be applied to the lighting vector.

Two-pass formulation: We chose to separate the transfer into a multi-bounce pass and a final gather pass. We further split the final

gather into diffuse and glossy components. This allows us to reduce storage requirements, since scalar values (instead of RGB) can now be used to represent the final gather matrix (which is the dominant component in terms of storage cost). In other words, the elements of $\mathbf{F}$ are RGB triples, but after separating out the diffuse/glossy coefficients, the elements of the matrices become single real values. We can write:

$$\mathbf{v}_i = \mathbf{v}_i^d + \mathbf{v}_i^g \quad (4)$$

$$\mathbf{v}_i^d = \mathbf{k}_d * (\mathbf{F}_d \cdot (\mathbf{M} + \mathbf{I}) \cdot \mathbf{g}_d) \quad (5)$$

$$\mathbf{v}_i^g = \mathbf{k}_g * (\mathbf{F}_g \cdot (\mathbf{M} + \mathbf{I}) \cdot \mathbf{g}_d) \quad (6)$$

where $\mathbf{k}_d$ and $\mathbf{k}_g$ are the diffuse and glossy coefficients, * denotes an element-wise multiplication and $\mathbf{F}_d$ and $\mathbf{F}_g$ are the diffuse and glossy gather matrices with scalar elements.

All matrix data is projected into wavelets and compressed by culling the least important coefficients; sparse matrix multiplications are performed after projecting the required vector into wavelet space. Thus our equations become:

$$\mathbf{v}_i^d = \mathbf{k}_d * (\mathbf{F}_d^w \cdot \mathbf{W} \cdot (\mathbf{M}^w \cdot \mathbf{W} + \mathbf{I}) \cdot \mathbf{g}_d) \quad (7)$$

$$\mathbf{v}_i^g = \mathbf{k}_g * (\mathbf{F}_g^w \cdot \mathbf{W} \cdot (\mathbf{M}^w \cdot \mathbf{W} + \mathbf{I}) \cdot \mathbf{g}_d) \quad (8)$$

3.4 The Structure of the Gather Cloud

One major issue is how to map the point samples of the gather cloud into wavelets. In environment map relighting systems, the cube-map faces are natural candidates for 2D wavelet projection. However, our gather cloud G is, as defined, an unstructured set of point samples, with no obvious way to impose a 2D Haar wavelet basis onto it. (The problem does not change substantially by considering 1D or 3D wavelets; we settled for 2D mostly because it is amenable to GPU implementation.)

Since we aim at storing the gather data as power-of-2 textures in our system, the number of gather samples, $\mathbf{n}_g$, is always a power of 4. The problem is to flatten the 4^n samples into a $2^n \times 2^n$ array. Let's define a *block* of the array to be a sub-region of the array of size $2^k \times 2^k$, starting at coordinates that are multiples of 2^k ($0 \leq k \leq n$). The flattening should be such that blocks of the array contain reasonably similar samples. This makes the Haar wavelet compression efficient.

We solve this problem by forming a *cluster hierarchy* on top of the gather cloud. Each cluster is either a leaf with only one sample, or it has four child clusters of equal size. The top cluster contains all n_g samples. The hierarchy is thus a perfectly balanced quad-tree. The flattening is implicit in the hierarchy construction, and we can think of the gather array and the gather hierarchy as equivalent. In Section 4.2, we describe how the hierarchy is constructed.

3.5 Sparse Matrix-Vector Multiplication

Recall that at relighting time, we recompute the indirect component as a sparse matrix-vector multiplication. In the following, we describe the data arrangement that allows for an efficient GPU implementation of this multiplication.

We introduce an *image-based* approach to computing this multiplication. As noted in [Ng et al. 2003] and other environment-map-based relighting work, the non-zero elements in $\mathbf{g}_d^w$ essentially give the coefficients of "wavelet lights" that approximate the original (unprojected) lighting vector $\mathbf{g}_d$. Our wavelet lights span the surfaces of the scene, instead of corresponding to environment map blocks. The columns of the matrix $\mathbf{T}^w$ can be viewed as images rendered with the corresponding wavelet light having a coefficient of 1 and all other wavelet lights being off. The multiplication is performed by accumulating each of these images scaled by the intensity of the corresponding wavelet light. Section 4.4 will describe how to encode and perform this sparse multiplication efficiently on a GPU.

Another approach to matrix-vector multiplication is the row-based one, where elements of the output vector are computed one-by-one as dot products of matrix rows with the vector. One major benefit of the image-based formulation is that it can exploit sparsity also in the vector, not just the matrix: we can skip the accumulation of images that correspond to wavelet lights of negligible intensity.

Transport-weighted on-the-fly wavelet culling: We cull wavelet lights that have a small contribution to the final image. In particular, we ignore wavelet lights where the norm of the intensity of the wavelet light multiplied by the norm of the corresponding image is less than ε. This "on-the-fly" wavelet culling (also called "non-linear approximation" in other relighting work) gives a further speed-up when small errors are acceptable in the final image. Note that this type of culling is different from the one used when precomputing transfer matrices (see section 4.3).

3.6 Multi-sample Antialiasing

Out system performs view antialiasing by supersampling, computing direct light at higher resolution and then downsampling. We supersample by a factor of 2 and use a box filter to downsample (albeit other filtering/sampling options are easily integrated). Indirect illumination is always computed per-pixel. To combine direct and indirect, we upsample the indirect to the direct's resolution, while trying to avoid light leaking across edges. We do this by assigning the closest neighbor (based on normal and position) in a 3x3 region around the direct sample. This technique is inspired by the edge-preserving anti-aliasing used in [Bala et al. 2003], except we try to infer discontinuities from positions and normals instead of finding them explicitly.

3.7 Comparison with Monte Carlo

The image quality in our algorithm is affected by several factors: the number of gather samples, their distribution, approximations made when computing the matrices, etc. However, we found that the most important factor is the number of preserved wavelet coefficients. Figure 3 shows that for the numbers of coefficients used in our results (100 for final gather and 40 for multi-bounce) we get a good visual match to the reference Monte Carlo solution. If we treat these images as vectors of $640 \times 480 \times 3$ elements between 0 and 1, the relative error (measured as the norm of the difference divided by the norm of the reference) is 13.6%. However, most of the difference is on the edges, since our algorithm and the path tracer have different anti-aliasing algorithms. If we eliminate these pixels (which account for less than 5% of the image) the relative error is 5.4%.

4 Data Precomputation

Here we describe how we pick the gather samples, how the cluster hierarchy is constructed on top of them, and how the transfer matrices are precomputed.

4.1 Sampling the Gather Cloud

Choosing gather samples is crucial for the quality of our algorithm. We use a divide-and-conquer uniform-sampling algorithm for small box-like scenes, and extend it to importance sampling for larger

(a) multi-bounce: 8 coeffs
final gather: 20 coeffs

(b) multi-bounce: 40 coeffs
final gather: 100 coeffs

(c) pure Monte Carlo
(1 hour on 32 processors)

Figure 3: A scene lit by a sharp spotlight on the floor. The most important factor affecting quality is the number of preserved wavelet coefficients. Our results use the number of coefficients shown in (b), which is enough to get a close visual match with the reference image without sacrificing interactive performance.

environments. Thus we take advantage of the fact that given a fixed view-point there are large differences in how much different parts of the scene contribute to the final indirect lighting. The concept of importance was first introduced in [Smits et al. 1992].

Uniform sampling: Assume we are given a set of triangles, and a budget of n_g desired samples. We want a uniform sampling, so that the expected number of samples on a triangle is proportional to its area. We split the set of triangles into two subsets of roughly equal total area by an axis-aligned plane, split our budget accordingly, and continue recursively.

Eventually we reach one of two cases. Either we only have one triangle left to sample, in which case we sample it uniformly. Or, we might be left with a budget of one sample; in that case we pick a triangle randomly (area-weighted) and pick the sample within that triangle. For each sample, we also store the normal, the area represented by the sample and the diffuse albedo.

Importance sampling: The basic idea is to compute the importance of each triangle and make the expected number of samples of a triangle proportional to the product of its area and importance, rather than just area. Note that we subdivide large triangles, so that the importance evaluation is not too coarse for them.

To compute the triangle importances, we use a variation of photon mapping, where particles are shot from the *view samples*. A number of particles with unit energy are shot from every view sample, and bounced around the scene like in standard photon mapping. To estimate the importance of a given triangle, a density estimation is done at a random point of the triangle. We could alternatively just increase the importance of every triangle that is hit by a particle, but that tends to miss small triangles completely.

4.2 Gather Hierarchy Construction

As noted in Section 3.4, we build a perfectly balanced quad-tree of clusters on top of the gather samples. This tree has two functions. First, it defines the flattening of the samples into a gather image amenable for wavelet compression. Second, it is used to speed up the final gather matrix computation, by allowing us to consider links to clusters of gather samples instead of linking to every single gather sample.

We build the tree top-down, by splitting the set of samples into two equal subsets and continuing recursively. Splitting should be done so that samples in any one of the two subsets are as similar as possible.

To split a set of samples, we use an algorithm that iteratively improves the split. The well-known k-means algorithm would repeat two steps: assign each sample to the partition with the closer centroid, and recompute the centroids. However, this might not split the set equally. Therefore, we use a simple modification: we compute for each sample the value $d(s,c_1) - d(s,c_2)$, where $d(s,c_i)$ is the squared distance from the sample s to centroid c_i. This way, a negative (positive) value implies closeness to c_1 (c_2). We sort the samples by this value and use the first and second half of this array as the new partitioning. We iterate until convergence.

The distance function used above depends both on positions and normals:

$$d(s,c) = K^2\|p_s - p_c\|^2 + d^2\|n_s - n_c\|^2 \qquad (9)$$

where d is the length of the scene bounding-box diagonal. We found good values of K to be about 20 to 40. Giving very low weight to either positions or normals leads to sub-optimal hierarchies, with higher error for the same number of preserved wavelet coefficients.

4.3 Computing the Matrices

Our high-level approach for precomputing each matrix is to compute a row of the dense (unprojected) matrix approximately, project it into wavelet space, and keep the desired number of the most important coefficients. For the final gather, a hierarchical algorithm is used; for the multi-bounce matrix, we use a photon mapping variation.

Area-weighted wavelet culling. Instead of just keeping coefficients with highest values, we first weight them by the number of non-zeros of the corresponding wavelet basis function, i.e. its "area". This gives much better results, since the highest-frequency coefficients (corresponding to the smallest wavelet lights) tend to also have the highest error.

The diffuse final gather matrix: The elements of the diffuse final gather matrix $\mathbf{F}_d$ are one-bounce diffuse contributions of every gather sample to every view sample. These are easy to define analytically: they depend on the visibility between the view and the gather sample, their normals, their distance, and the area represented by the gather sample. This is similar to form factors in radiosity, except we use point samples instead of patches.

A simple brute-force approach to compute the matrix $\mathbf{F}_d^w$ would be to evaluate each row of $\mathbf{F}_d$ completely, and wavelet-project the row. Instead, we use a hierarchical approach similar to N-body algorithms: if a cluster of gather samples is far enough and/or small enough, we can compute a link to a random sample in the cluster

and assume all other links to the rest of the cluster's samples are the same.

To decide whether a cluster should be subdivided, we use an approximate solid angle heuristic: we check if s^2/d^2 is greater than a user-defined threshold, where s is the cluster bounding-box diagonal and d is the distance to the cluster center. We also cull clusters that are completely behind the view sample.

Elements of $\mathbf{F}_d$ are proportional to $1/r^2$, where r is the distance between the corresponding view-gather sample pair. This creates numerical problems if r is small (especially at corners). We simply clamp the values of $\mathbf{F}_d$ to be at most 0.1. This leads to some darkening in the corners; however, in most cases this is not too distracting. Other systems based on point samples have the same problem; for further discussion of this issue see [Walter et al. 2005]. (In patch-based radiosity, analytic form factors can be used to solve this problem.)

The glossy final gather matrix: Our approach to compute $\mathbf{F}_g$ (and $\mathbf{F}_g^w$) is analogous to the diffuse case. There are two differences. First, the elements will have the value of the glossy BRDF factored in. Second, we can update the solid angle heuristic to force more subdivision around the peak of the lobe. (Effectively, the subdivision threshold will depend on the value of the lobe. Similar adaptive subdivision is done in [Walter et al. 2005]).

This technique works well for both sharp and wide lobes, and it is reasonably fast. However, it cannot handle the full range of glossy light paths, because it misses inter-reflections between glossy surfaces. This limitation could be fixed by including in $\mathbf{F}_g$ multiple glossy bounces; however, it is not obvious how to precompute these bounces efficiently, so it remains as an open research problem.

The multiple-bounce matrix: Since the output of the multiple-bounce pass is not viewed directly, we assume it can be computed and stored with lower precision. We use a variation of photon mapping to compute approximate rows of the matrix $\mathbf{M}$. To get $\mathbf{M}^w$, we project the rows and keep important elements, as before.

We cannot shoot photons from light sources, since we do not know their position. Therefore, we shoot the photons from the gather samples themselves. Each gather sample is treated as a Lambertian emitter (note that we know the area that the sample represents). Each photon is deposited and bounced at each surface hit point, as in standard photon mapping. Russian-roulette can be used to get a theoretically infinite number of bounces; however, in our results we limit the photon path length to 3.

Additionally, we store with each photon the ID of the gather sample it originated from. After the shooting and depositing pass is finished, a K-nearest-photon query is done at every gather sample. For each sample g_i, this set of photons gives us a sparse (non-wavelet) approximation of the i-th row of $\mathbf{M}$.

Note that K should be much higher than the number of preserved wavelet coefficients; otherwise the matrix is noisy and hard to pack for GPU multiplication (see below). In our results, we use one million photons total, 2000 photons in the nearest photon query, and 40 wavelet coefficients.

4.4 Packing the Matrix Data

We treat each sparse matrix as a set of sparse images corresponding to wavelet lights, as explained in Section 3.5. Here we show how we pack these images for efficient GPU evaluation.

The key is to realize that the sparsity of the images is of a special kind. In particular, the non-zero elements tend to be quite coherent and localized (especially true for smaller wavelet lights). Because of the nature of wavelet lights, the only surfaces with a significant contribution will be those that are quite close to the light, or do not see the whole wavelet light due to occlusion.

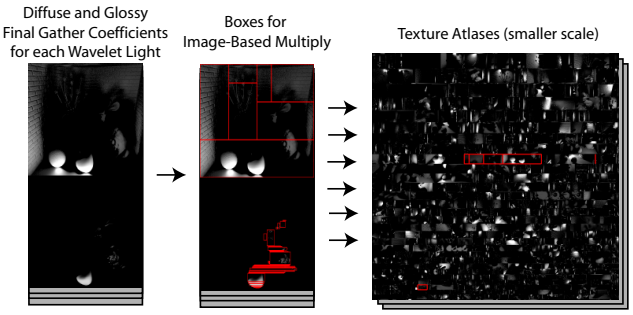


Figure 4: Data layout used to efficiently encode the sparse image-based multiplication.

We exploit this structure by cutting the non-zero parts of an image into one or more rectangular blocks, trying to minimize waste. We start by fitting the tightest rectangle around the non-zero elements. If the waste (i.e., the percentage of zeros in the rectangle) is higher than a threshold, we find the best split into 2 rectangles (i.e., the one with lowest waste), and possibly iterate this process recursively. The waste can be pushed arbitrarily low at the expense of more splits.

The output of this procedure is a large number of small *blocks* (possibly tens of thousands). The total waste of this algorithm is about 50% with our current settings, but can be pushed arbitrarily low at the expense of a larger number of smaller blocks. We pack the blocks into a few 4k $\times$ 4k texture atlases to avoid having to manage a very large number of textures. This is a 2D bin-packing problem, which is NP-complete in theory; however, simple greedy heuristics work well. We represent the current empty space in the texture atlas as a linked list of rectangles. We sort the blocks by height, fit each block to the first available rectangle in the list, and update the list. The waste produced in this step is generally less than about 3%.

5 Mapping Computation to GPUs

One goal of our algorithm design was to achieve high-speed relighting *completely* on the GPU. Porting the transport algorithms to GPU gave us a considerable speed-up; we have measured factors of 3 to 9.

Our implementation computes direct illumination on view and gather samples for each light change. The resulting gather direct, $\mathbf{g}_d$, is then passed through a sequence of transforms and finally reconstructed in an indirect image. Direct and indirect are then accumulated and resampled for antialiasing. Our system supports multiple lights by caching the results of all lights but the one that is currently moving; therefore, having more lights does not decrease performance.

One of the challenges we found in mapping to the GPU was making sure our algorithm kept all data in video memory. For this reason, we quantize normals and diffuse coefficients to 8 bits-per-channel (bpc), while keeping position and specularity information in 16 bpc. All transform data, which constitutes the majority of the dataset size, is also quantized to 8 bits with additional scaling factors stored for each block. Buffers containing partial results are stored in 16 bpc. We have found no artifacts due to quantization in our tests.

Direct illumination on the view and gather samples is evaluated by first rendering shadow maps for the current light and then shading images containing position, normals, diffuse and specular coeffi-

cients for each sample; arbitrary direct illumination models are expressed using light shaders written in a high level shading language. In this respect our algorithm is similar to [Pellacini et al. 2005]. Soft-shadow lights are evaluated by performing multiple passes, recomputing shadow maps and shading images in each pass, and accumulating the results. Shadows for omni lights use six shadow maps arranged in a cube-map. Shadow map computation is accelerated by per-object culling.

Image-based transport. Our algorithm computes light transport as a sequence of 2D Haar wavelet transforms and sparse matrix multiplications. Wavelet transforms are implemented using a multi-pass technique where at each pass the image is shrunk by a factor of 2 and a shader computes the coefficients corresponding to that level.

Most of the execution time of our algorithm is spent in performing sparse matrix multiplications. The image-based formulation introduced previously allows us to perform this multiplication very efficiently on the GPU. We draw one camera-aligned quad for each image block in the sparse matrix. Each quad is shaded by multiplying the matrix elements, corresponding to the current block and stored in the atlases, by the wavelet light intensity, which is the same for each element in the block. Image blocks (and thus the corresponding quads) are drawn sorted by the texture atlas they access, to avoid state changes and shader binding inefficiencies.

Finally, we have implemented wavelet light culling in a vertex shader. Block norms are computed as a preprocess and sent to the vertex shader together with the texture containing the wavelet light intensities. The vertex shader can then send the quad out of the view if it should be culled. This final piece allows us to perform the entire algorithm on the GPU and take advantage of wavelet light culling without any CPU intervention.

Comparison with row-based multiplication. We have also compared the image-based method with a row-based method similar to [Bolz et al. 2003]. In this case, for each output pixel, a list stores indices to wavelet lights and their associated coefficients (either one or three channels).

One advantage of the row-based algorithm is that it can be applied to almost any kind of sparsity in the matrix, while the image-based method requires a very specific kind of sparsity. However, this method has some disadvantages. First, on-the-fly culling is not possible. Second, the data in this format cannot be quantized nor packed as well as in the image-based format, which means that much more GPU memory is needed to achieve the same quality. Third, the row-based algorithm requires at least one dependent texture read, decreasing performance further. The image-based algorithm does not use dependent texture reads; RGB weights require no additional read. A comparison of the two methods for the case of single channel weights (best for row-based) gave us about 50% slowdown (compared to non-culled image-based code).

Given its high performance, together with the advantage of on-the-fly culling and better packing, we believe the image-based formulation is the best alternative for this problem domain.

6 Results

We have tested our algorithm on a 3.2 GHz P4 with 2 GB of RAM and an NVidia 7800 graphics accelerator with 256 MB of RAM. Note that we use only one processor for data precomputation. We show four scenes. Table 2 shows various statistics for our algorithm running at a video resolution of 640×480 with per-pixel indirect illumination and 2×2 supersampled direct illumination. Images of each scene, lit by omni lights and spot lights, are presented in Figure 5; some light configurations show large areas illuminated exclusively by indirect illumination.

	Sponza	**Still**	**Hair**	**Temple**
Triangle count	66k	107k	320k	2124k
Final gather preprocess	**1.2 hrs**	**1.3 hrs**	**2.6 hrs**	**2.2 hrs**
Avg. visibility rays	4164	3863	5159	6754
Final gather size	72 MB	80 MB	100 MB	58 MB
Multi-bounce size	23 MB	19 MB	10 MB	26 MB
Final gather coeffs	100	100	100	100
Multi-bounce coeffs	40	40	20	40
Spot light				
FPS range (cull on)	**21.3 - 24.9**	**13.7 - 18.7**	**18.0 - 24.7**	**13.8 - 25.8**
FPS range (cull off)	13.1 - 15.5	10.9 - 13.0	9.5 - 10.7	11.3 - 13.2
Shadow map rendering	1.4 ms	2.7 ms	3.7ms	4.9 ms
Shading view & gather	3.6 ms	3.5 ms	3.9 ms	3.5 ms
Transform (cull on)	32.9 ms	46.1 ms	27 ms	44.2 ms
Transform (cull off)	62.8 ms	76.4 ms	82.8 ms	56.5 ms
Final filtering	4.7 ms	4.6ms	5 ms	4.9 ms
Omni light				
FPS range (cull on)	**13.7 - 18.2**	**11.4 - 13.5**	**9.7 - 11.5**	**8.5 - 9.1**
FPS range (cull off)	11.3 - 12.2	9.7 - 10.5	8.6 - 10.0	6.7 - 7.5
Shadow map rendering	7.3 ms	11 ms	15.2 ms	47.4 ms
Shading view & gather	10 ms	8.3 ms	12 ms	11.6 ms
Transform (cull on)	44.8 ms	58.2 ms	67.4 ms	47.1 ms
Transform (cull off)	62.7 ms	75.8 ms	83.4 ms	64.2 ms
Final filtering	4.7 ms	4.8 ms	5 ms	5.3 ms

Table 2: Statistics of tested scenes.

The scenes where chosen to show different degrees of geometric and material complexity. We have chosen two architectural environments, Sponza and temple, that display spatially large environments and a geometric complexity of 2.1M polygons (temple). In addition, we tested our algorithm on a hair-ball scene that displays all its highly detailed geometry directly in the view, and finally a still-life scene with smooth and sharp glossy objects, detailed bump-maps and plant leaves.

Run-time performance: In all these cases, we were able to compute accurate indirect illumination at interactive rates of roughly 7–15 frames per second without on-the-fly wavelet culling, and roughly 9–25 frames per second with culling, with almost no perceptible difference. These frame-rates show that our system can handle scenes with high polygon counts, whether this geometric complexity is spread over a large volume, such as in the temple scene, or concentrated in a small one, like in the hair-ball scene.

As seen in Table 2, the run-time of the GPU relighting is dominated by the transfer and occasionally by the shadow map rendering. Wavelet culling can be used to increase transfer performance with no objectionable artifacts. Furthermore, if slight artifacts are acceptable, additional performance can be gained by more aggressive light culling, providing a speed-quality trade-off. Shadow map rendering was only accelerated using per-object culling; we believe further optimizations, including LODs, would speed up this component considerably with no loss of visual quality.

Data size: For each of our scenes, we have chosen a 64k gather cloud, a number we picked as the smallest one that gave us high quality results. (Note that our choices are limited by the power-of-4 requirement.) The corresponding matrix data sets were culled to fit in GPU memory after quantization to roughly 100 MB of data for the two sparse matrices. This is enough for a high-quality solution; future GPUs with more memory will allow for higher image resolutions.

Precomputation time: Our system requires modest precomputation times, in the order of a maximum of 3 hours on one processor. This time is dominated by the final gather matrix computation, which took at most 2.6 hours. Note that these timings are linear in the number of view samples (pixels) – a smaller image would

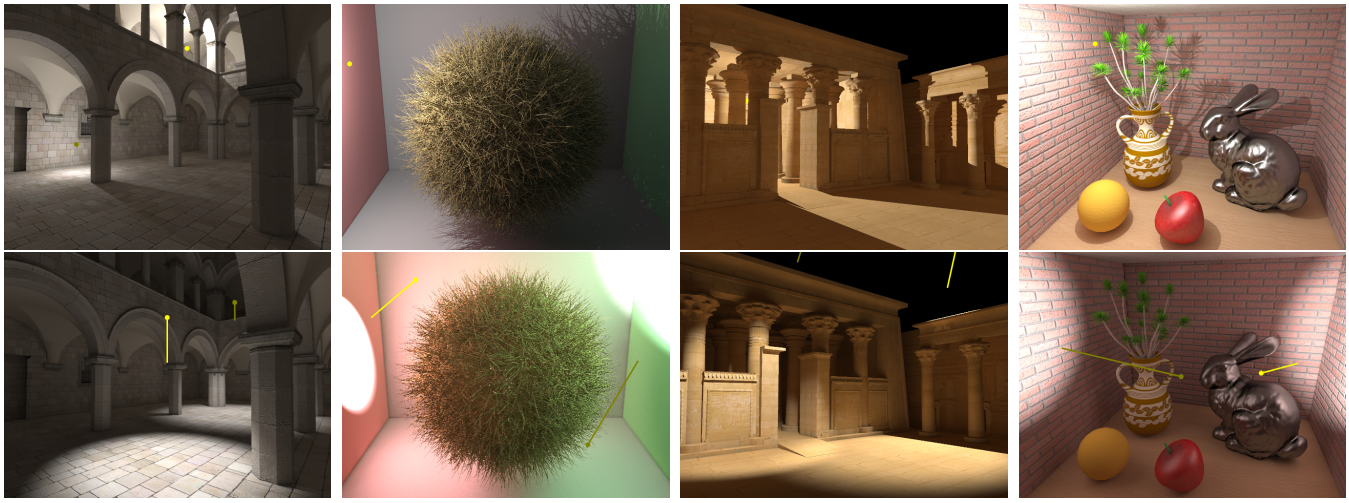


Figure 5: Images rendered with our system using omni and spot lights: Sponza, hair ball, temple, still life.

be correspondingly faster. We check about 5000 rays per pixel, a number comparable to typical load for a high-quality final gather in ray-tracing settings. All other precomputation (sampling view and gather clouds, multi-bounce transfer, packing) takes less than 20 minutes.

Lighting effects: The still-life dataset shows that our system can handle sharp glossy materials, typically problematic in other relighting work, together with high complexity geometric details in the bump maps as well as the plant leaves. Note that the indirect illumination is very detailed, since it is computed fully per pixel, without relying on any kind of subsampling. While the main drawback of our solution is the fixed-view limitation, we believe that the high resolution indirect illumination, together with arbitrary glossy materials and high geometric complexity justifies our assumption.

Figure 6 shows images generated with arbitrary light shaders: a light projecting arbitrary images, a colored light modeled after a widely-used cinematic lighting model [Barzel 1997], and a multi-sampled soft-shadow light. We have also simulated a sun-sky model using a distant light for the sun and ambient occlusion for the sky, useful for outdoor environments. Some of these lights have high frequency details and many degrees of freedom, which are problematic in previous approaches but very common in cinematic lighting. Our system can handle all of these cases, together with any arbitrary direct illumination algorithm, since it precomputes the direct-to-indirect transfer itself, rather than trying to interpolate a small number of fully precomputed solutions. That would miss many interesting lighting details in the indirect, like the crisp reflections and indirect shadows in the still-life scene.

Future work: While this paper did not focus on environment map lighting, we have prototyped an implementation in our system. The basic idea is to split the gather cloud into surface samples and environment samples. The surface samples are shaded by direct lighting as usual. The environment samples have their radiance assigned by a separate shader, which can determine the radiances in an arbitrary way, e.g. by rotated environment map lookups. This framework supports a combination of distant and local lighting together with multiple bounces of indirect illumination.

As future work, we are interested in expanding the kinds of materials supported by our system, including perfectly specular (reflection/refraction) and translucent materials exhibiting various degrees of subsurface scattering.

While we believe our fixed-camera assumption is necessary to handle high-complexity geometry, we are also interested in a moving-camera system for moderately complex environments, while keeping flexible direct lighting models.

7 Conclusions

We have presented an interactive relighting system aimed at cinematic lighting design, which supports multiple-bounce indirect illumination in scenes with high geometric complexity, diffuse and glossy materials (including sharp gloss), and flexible direct lighting models expressed using procedural shaders. Our deep frame-buffer approach uses direct-to-indirect transfer from a set of gather samples distributed in the scene to the view samples corresponding to image pixels. Light transfer is represented as a series of sparse matrix multiplications and encoded in wavelet space after mapping the gather cloud to the wavelet basis. Multiplication is efficiently performed on the GPU thanks to an image-based method that also allows us to cull wavelet lights on the fly. Precomputation times are kept reasonable by splitting the transfer into two passes and evaluating it using a variation of photon mapping and hierarchical final gathering.

Acknowledgments

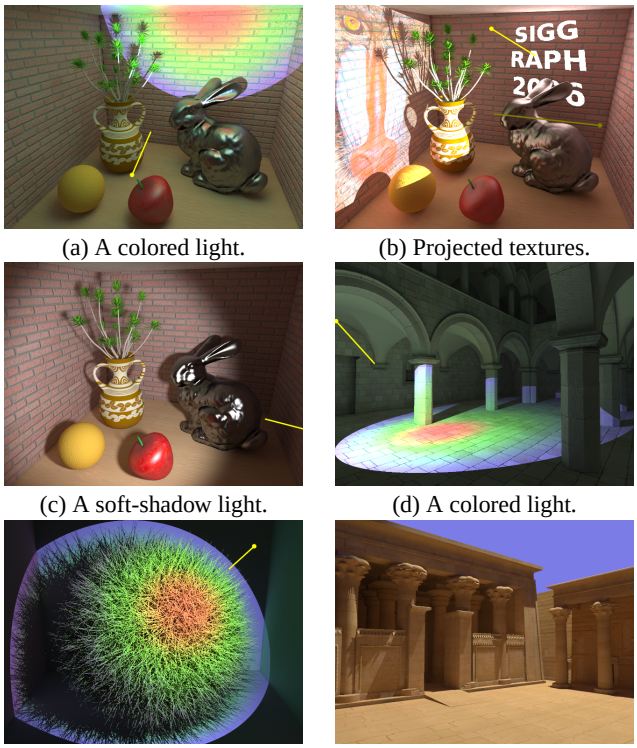
We thank Veronica Sundstedt and Patrick Ledda for the temple model. We thank Bruce Walter and the Program of Computer Graphics for support. This work was supported by NSF Grant CCF-0539996 and a grant from Pixar Animation Studios.

References

ANNEN, T., KAUTZ, J., DURAND, F., AND SEIDEL, H.-P. 2004. Spherical harmonic gradients for mid-range illumination. In *Rendering Techniques 2004 Eurographics Symposium on Rendering*, 331–336.

ARIKAN, O., FORSYTH, D. A., AND O'BRIEN, J. F. 2005. Fast and detailed approximate global illumination by irradiance decomposition. In *Proceedings of ACM SIGGRAPH 2005*, 1108–1114.

BALA, K., DORSEY, J., AND TELLER, S. 1999. Interactive ray-traced scene editing using ray segment trees. In *10th Eurographics Workshop on Rendering*, 39–52.



(a) A colored light. (b) Projected textures.
(c) A soft-shadow light. (d) A colored light.
(e) A colored light. (f) A "sun-sky" lighting model.

Figure 6: Images rendered with our system using various light shaders.

BALA, K., DORSEY, J., AND TELLER, S. 1999. Radiance interpolants for accelerated bounded-error ray tracing. *ACM Transactions on Graphics 18*, 3, 213–256.

BALA, K., WALTER, B., AND GREENBERG, D. 2003. Combining edges and points for interactive high-quality rendering. In *Proceedings of ACM SIGGRAPH 2003*, 631–640.

BARZEL, R. 1997. Lighting controls for computer cinematography. *Journal of Graphics Tools 2*, 1, 1–20.

BOLZ, J., FARMER, I., GRINSPUN, E., AND SCHRÖDER, P. 2003. Sparse matrix solvers on the gpu: conjugate gradients and multigrid. *Proceedings of ACM SIGGRAPH 2003*, 917–924.

BRIÉRE, N., AND POULIN, P. 1996. Hierarchical view-dependent structures for interactive scene manipulation. In *Proceedings of ACM SIGGRAPH 96*, 83–90.

DAYAL, A., WOOLLEY, C., WATSON, B., AND LUEBKE, D. 2005. Adaptive frameless rendering. In *Proceedings of Eurographics Symposium on Rendering*.

DMITRIEV, K., BRABEC, S., MYSZKOWSKI, K., AND SEIDEL, H.-P. 2002. Interactive Global Illumination Using Selective Photon Tracing. In *13th Eurographics Workshop on Rendering*, 25–36.

DRETTAKIS, G., AND SILLION, F. 1997. Interactive Update of Global Illumination Using A Line-Space Hierarchy. In *Proceedings of ACM SIGGRAPH 97*, 57–64.

GAUTRON, P., KRIVANEK, J., BOUATOUCH, K., AND PATTANAIK, S. 2005. Radiance cache splatting: A gpu-friendly global illumination algorithm. In *Proceedings of Eurographics Symposium on Rendering*.

GERSHBEIN, R., AND HANRAHAN, P. M. 2000. A fast relighting engine for interactive cinematic lighting design. In *Proceedings of ACM SIGGRAPH 2000*, 353–358.

GORTLER, S. J., SCHRÖDER, P., COHEN, M. F., AND HANRAHAN, P. 1993. Wavelet radiosity. In *Proceedings of ACM SIGGRAPH 93*, 221–230.

HANRAHAN, P., SALZMAN, D., AND AUPPERLE, L. 1991. A rapid hierarchical radiosity algorithm. *Proceedings of ACM SIGGRAPH 91*, 197–206.

JENSEN, H. W. 1996. Global illumination using photon maps. In *Proceedings of the Eurographics workshop on Rendering techniques '96*, 21–30.

KAUTZ, J., SLOAN, P.-P., AND SNYDER, J. 2002. Fast, arbitrary brdf shading for low-frequency lighting using spherical harmonics. In *EGRW '02: Proceedings of the 13th Eurographics workshop on Rendering*, 291–296.

KRISTENSEN, A. W., AKENINE-MÖLLER, T., AND JENSEN, H. W. 2005. Precomputed local radiance transfer for real-time lighting design. *Proceedings of ACM SIGGRAPH 2005*, 1208–1215.

LIU, X., SLOAN, P.-P. J., SHUM, H.-Y., AND SNYDER, J. 2004. All-frequency precomputed radiance transfer for glossy objects. In *Proceedings of Eurographics Symposium on Rendering*, 337–344.

NG, R., RAMAMOORTHI, R., AND HANRAHAN, P. 2003. All-frequency shadows using non-linear wavelet lighting approximation. *Proceedings of ACM SIGGRAPH 2003*, 376–381.

NG, R., RAMAMOORTHI, R., AND HANRAHAN, P. 2004. Triple product wavelet integrals for all-frequency relighting. *Proceedings of ACM SIGGRAPH 2004*, 477–487.

PELLACINI, F., VIDIMČE, K., LEFOHN, A., MOHR, A., LEONE, M., AND WARREN, J. 2005. Lpics: a hybrid hardware-accelerated relighting engine for computer cinematography. *Proceedings of ACM SIGGRAPH 2005*, 464–470.

SAITO, T., AND TAKAHASHI, T. 1990. Comprehensible rendering of 3-d shapes. In *Proceedings of ACM SIGGRAPH 90*, 197–206.

SÉQUIN, C. H., AND SMYRL, E. K. 1989. Parameterized ray tracing. In *Proceedings of ACM SIGGRAPH 89*, 307–314.

SLOAN, P.-P., KAUTZ, J., AND SNYDER, J. 2002. Precomputed radiance transfer for real-time rendering in dynamic, low-frequency lighting environments. In *Proceedings of ACM SIGGRAPH 2002*, 527–536.

SMITS, B. E., ARVO, J. R., AND SALESIN, D. H. 1992. An importance-driven radiosity algorithm. In *Proceedings of ACM SIGGRAPH 92*, 273–282.

TABELLION, E., AND LAMORLETTE, A. 2004. An approximate global illumination system for computer generated films. *Proceedings of ACM SIGGRAPH 2005*, 469–476.

TOLE, P., PELLACINI, F., WALTER, B., AND GREENBERG, D. P. 2002. Interactive global illumination in dynamic scenes. *Proceedings of ACM SIGGRAPH 2002*, 537–546.

WALD, I., KOLLIG, T., BENTHIN, C., KELLER, A., AND SLUSALLEK, P. 2002. Interactive Global Illumination. In *13th Eurographics Workshop on Rendering*, 15–24.

WALTER, B., DRETTAKIS, G., AND PARKER, S. 1999. Interactive rendering using the Render Cache. In *10th Eurographics Workshop on Rendering*, 19–30.

WALTER, B., DRETTAKIS, G., AND GREENBERG, D. 2002. Enhancing and optimizing the Render Cache. In *13th Eurographics Workshop on Rendering*, 37–42.

WALTER, B., FERNANDEZ, S., ARBREE, A., BALA, K., DONIKIAN, M., AND GREENBERG, D. P. 2005. Lightcuts: A scalable approach to illumination. In *Proceedings of ACM SIGGRAPH 2005*, 1098–1107.

WANG, R., TRAN, J., AND LUEBKE, D. P. 2004. All-frequency relighting of non-diffuse objects using separable brdf approximation. In *Proceedings of Eurographics Symposium on Rendering*, 345–354.

WANG, R., TRAN, J., AND LUEBKE, D. 2005. All-frequency interactive relighting of translucent objects with single and multiple scattering. *Proceedings of ACM SIGGRAPH 2005*, 1202–1207.

WARD, G., AND SIMMONS, M. 1999. The holodeck ray cache: an interactive rendering system for global illumination in nondiffuse environments. *ACM Transactions on Graphics 18*, 4, 361–368.

WARD, G. J., RUBINSTEIN, F. M., AND CLEAR, R. D. 1988. A ray tracing solution for diffuse interreflection. In *Proceedings of ACM SIGGRAPH 88*, 85–92.