Appears in SIGGRAPH 2007.

Matrix Row-Column Sampling for the Many-Light Problem

Miloš Hašan* Cornell University
Fabio Pellacini Dartmouth College
Kavita Bala Cornell University

2.2m triangles: 300 rows, 900 columns, 16.9 s | 388k triangles: 432 rows, 864 columns, 13.5 s | 869k triangles: 100 rows, 200 columns, 3.8 s

Figure 1: *In the above images, over 1.9 million surface samples are shaded from over 100 thousand point lights in a few seconds. This is achieved by sampling a few hundred rows and columns from the large unknown matrix of surface-light interactions.*

Abstract

Rendering complex scenes with indirect illumination, high dynamic range environment lighting, and many direct light sources remains a challenging problem. Prior work has shown that all these effects can be approximated by many point lights. This paper presents a scalable solution to the many-light problem suitable for a GPU implementation. We view the problem as a large matrix of sample-light interactions; the ideal final image is the sum of the matrix columns. We propose an algorithm for approximating this sum by sampling entire rows and columns of the matrix on the GPU using shadow mapping. The key observation is that the inherent structure of the transfer matrix can be revealed by sampling just a small number of rows and columns. Our prototype implementation can compute the light transfer within a few seconds for scenes with indirect and environment illumination, area lights, complex geometry and arbitrary shaders. We believe this approach can be very useful for rapid previewing in applications like cinematic and architectural lighting design.

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

Keywords: many lights, GPU, global illumination, sampling

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

1 Introduction

Rendering complex scenes with indirect illumination, high dynamic range environment lighting, and many direct light sources remains a challenging problem. Previous research [Keller 1997; Walter et al. 2005] has shown how all these sources of illumination can be converted into many point lights and treated as a single many-light problem. However, a brute-force rendering with many thousands of point lights is itself prohibitively expensive. The recently introduced *lightcuts* framework [Walter et al. 2005; Walter et al. 2006] presents a scalable solution for the many-light problem that produces accurate results in a few minutes of computation, using a CPU-based raytracer as its basic visibility algorithm. On the other hand, recent work on real-time lighting and material design (e.g. [Hašan et al. 2006; Ben-Artzi et al. 2006]) has highlighted the importance of providing high-quality, fast user feedback for changes to lighting and materials in design tasks. These approaches have been shown to scale to complex scenes and appearance models by relying on precomputation to amortize rendering costs, but this limits their applicability to static scenes and fixed appearance models (i.e., a user can either change the lights or the materials, but not both). Our goal is complementary to that of prior work. We seek an efficient solution to the many-light problem that provides fast user feedback without requiring precomputation, thus supporting any scene changes. We explicitly target execution on GPUs to take advantage of their computing power. Furthermore, we seek a solution that supports arbitrary appearance models, expressed as programmable shaders.

To take advantage of GPU capabilities, we formulate the many-light problem as a large matrix of sample-light interactions, where each column represents all samples lit by an individual light, while each row represents an individual sample shaded by all lights. A brute-force solution can be computed by accumulating all columns of such a matrix, thus summing the contribution of each light for all pixels. We show how this sum can be efficiently approximated on the GPU. Our key observation is that this matrix is often close to low rank, so its structure can be revealed by sampling a small number of rows and columns. The main benefit of this approach is that GPUs can compute entire rows and columns very efficiently using shadow mapping.

This paper makes the following contributions:

- We introduce an algorithm that efficiently uses both rows and columns of the matrix; in particular, we use the the rows to select a good set of columns. We demonstrate how row sampling provides valuable information about the structure of the matrix.
- To analyze the information provided by the computed rows, we introduce a novel clustering metric to minimize the expected error of the algorithm. We further show a practical approach to solve the resulting large-scale discrete optimization problem.
- Since our algorithm relies on entire rows and columns that can be evaluated using shadow mapping, we believe our framework to be the first to effectively map the many-light problem to GPUs.
- By viewing the problem as an abstract matrix with no additional information, we present a flexible technique that can handle any light and material type, physically-based or not.
- We present a prototype implementation showing transfer of illumination from over 100 thousand lights to over 1.9 million surface samples that achieves high accuracy in a few seconds.

We believe our algorithm would be very useful in many applications that require high-quality previewing of complex illumination, including cinematic or architectural appearance design. Our rendering times of a few seconds per image achieve a middle ground between interactive techniques that compromise quality for performance, and high-quality off-line renderers.

2 Related Work

Many lights: Several CPU-based many-light algorithms have been published. [Ward 1994] sorts the lights to determine which lights to prune at each pixel. In a Monte Carlo setting, [Shirley et al. 1996] voxelize the scene to pick important lights, and sparsely sample unimportant lights. [Paquette et al. 1998] construct hierarchical trees of lights, but ignore visibility and shadowing.

Lightcuts [Walter et al. 2005; Walter et al. 2006] introduce a hierarchical, error-bounded rendering algorithm for many lights, based on a ray-tracer to compute visibility. As in our work, they treat complex illumination as a problem of computing the interactions between many sample-light pairs. In our matrix terminology, this approach subdivides the matrix into blocks, and uses a perceptual threshold to find blocks that can be efficiently approximated by a single interaction.

Some interactive CPU-based approaches use ray tracing to sample visibility in many lights scenes. [Fernandez et al. 2002] cache and reuse visible lights and blockers, but can require large amounts of memory. [Wald et al. 2003] importance sample lights to achieve interactive performance, assuming highly occluded environments.

CPU-based indirect/environment illumination: A closely related problem to our matrix of light-surface interactions is the matrix of form-factors studied by radiosity algorithms. Similarly to us, the goal is to come up with a scalable algorithm that is asymptotically better than brute-force. A large body of research on hierarchical radiosity (HR) was started by the seminal paper [Hanrahan et al. 1991]. These approaches essentially approximate blocks of the matrix with a constant, and use error estimation oracles to decide whether to subdivide or approximate. Importance [Smits et al. 1992] can be used to speed up the convergence of these techniques. Hemicube approaches might be thought of as sampling rows and columns of the form-factor matrix. Radiosity algorithms usually work with diffuse surfaces; supporting other materials is possible but not trivial [Christensen et al. 1997]. Often, an expensive final gather step is necessary to produce high-quality output from a radiosity algorithm. [Scheel et al. 2001; Scheel et al. 2002] present techniques to speed up this process.

Irradiance caching [Ward et al. 1988] is a widely used technique to accelerate Monte Carlo ray-tracing by exploiting the smoothness of illumination; however, it works less well in non-smooth cases, e.g. detailed geometry, high-frequency environment maps or indirect shadows from strong secondary sources. Photon mapping [Jensen 2001] is commonly used in several different forms: with and without final gather, and with final gather using irradiance caching. Without final gather, it can provide fast previews of indirect or environment illumination, but with some blurry artifacts. Several systems cache sparse global illumination samples, allowing for interactive camera and object movement [Walter et al. 1999; Ward and Simmons 1999; Tole et al. 2002; Bala et al. 2003].

Environment lighting is similar to indirect illumination in that it also involves hemisphere integration. On the other hand, there is a large amount of work dealing specifically with environment maps. [Agarwal et al. 2003] is similar to our work in that it is also looking for a clustering (or stratification) that minimizes some expected error measure. A fast approach to convert an environment to a small number of lights is given in [Ostromoukhov et al. 2004].

GPU algorithms: Instant radiosity [Keller 1997] presents a solution to the indirect illumination problem by shooting particles from light sources and converting surface hits into *indirect lights*, rendered on graphics hardware as point lights with cosine emission. Our framework handles indirect illumination similarly, but we create a large number of indirect lights and use our row-column sampling algorithm to reduce the computation. We also compare our algorithm to instant radiosity in Section 5. [Carr et al. 2003] present a radiosity implementation on the GPU for diffuse environments. [Purcell et al. 2003] presents an implementation of the photon mapping algorithm on graphics hardware.

Radiance cache (splatting) [Gautron et al. 2005; Křivánek et al. 2006] presents an algorithm for one-bounce global illumination that takes advantage of illumination coherence by subsampling it at a sparse set of locations. Temporal radiance caching [Gautron et al. 2006] accelerates computation of global illumination for image sequences by reusing samples between frames. Reflective shadow maps [Dachsbacher and Stamminger 2005] and Splatting indirect illumination [Dachsbacher and Stamminger 2006] provide interactive solutions for one-bounce global illumination but neglect shadowing effects in the indirect bounces.

PRT algorithms: Finally, several precomputed transfer techniques have been shown to support interactive previewing of lighting and material changes in static scenes, e.g. [Ng et al. 2004; Sloan et al. 2002; Hašan et al. 2006; Ben-Artzi et al. 2006]. They do so at the price of several minutes to hours of precomputation and (in some cases) fixing the camera.

Theoretical work on row-column sampling: It is not difficult to see that if a matrix $\mathbf{A}$ has rank exactly k, then there exist k rows and k columns that reveal its structure entirely. In particular, let $\mathbf{C}$ be the matrix of sampled columns, $\mathbf{R}$ the matrix of sampled rows, and $\mathbf{W}$ the matrix of intersection elements. If the rows and columns were chosen so that the rank of $\mathbf{W}$ is k, then we have $\mathbf{A} = \mathbf{C}\mathbf{W}^{-1}\mathbf{R}$ – this is called a skeleton decomposition of $\mathbf{A}$. Pseudo-skeleton approximations [Goreinov et al. 1997] extend this to the case when $\mathbf{A}$ is only close to rank k, by proposing the approximation $\mathbf{A} \approx \mathbf{C}\mathbf{W}_\tau^+\mathbf{R}$, where $\mathbf{W}_\tau^+$ is the Moore-Penrose pseudo-inverse ignoring

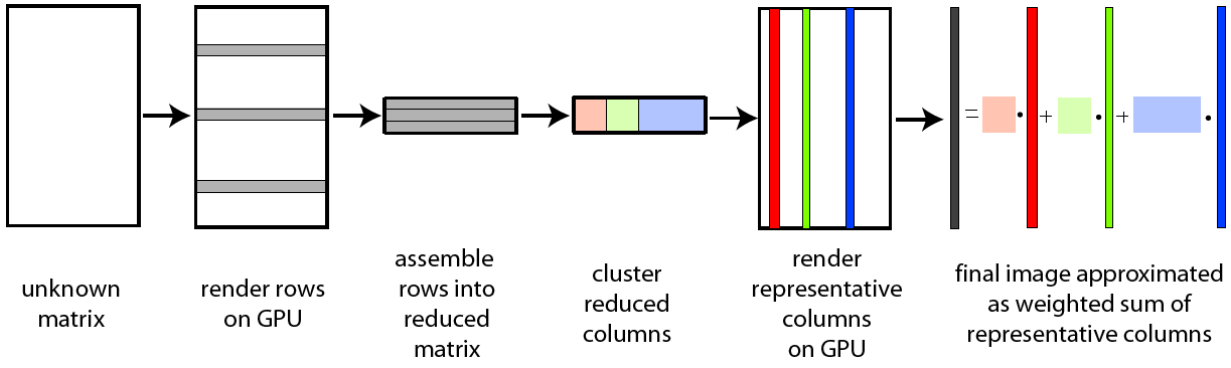

Figure 2: *Conceptual overview of our algorithm.*

Symbol	Description	Size
m	Number of surface samples	scalar
n	Number of lights	scalar
r	Number of computed rows	scalar
c	Number of computed columns	scalar
$\mathbf{A}$	Full lighting matrix	$m \times n$
$\mathbf{R}$	Matrix of computed rows	$r \times n$
$\mathbf{K}$	Matrix of cluster centers	$r \times c$
φ_j	Full column (i.e. column of $\mathbf{A}$)	$m \times 1$
ρ_j	Reduced column (i.e. column of $\mathbf{R}$)	$r \times 1$
s_i	Sum of norms of all ρ_j in cluster i	scalar

Table 1: *Summary of the notation used in the paper.*

singular values smaller than τ. However, the paper does not present a method to pick the rows and columns and the parameter τ in a robust way, without the need to compute the SVD of $\mathbf{A}$. A similar, more recent approach is presented in [Drineas et al. 2006]. Again, the running time of this algorithm is dominated by computing the SVD of $\mathbf{A}$, so it is not applicable in our domain.

3 Matrix Row-Column Sampling

This section describes our row-column sampling algorithm that presents an efficient solution to the problem of shading m surface samples from n lights. Ideally, for each sample we would like to compute the sum of the contributions from all lights. A brute-force approach would take time $O(mn)$, which is clearly not practical for the values of m and n required for high-quality rendering (thousands to millions). Our algorithm runs in $O(m+n)$ time and exploits the GPU to push down the constant factor.

3.1 A Matrix Formulation of the Many-Light Problem

We can formally describe the problem as follows: consider a matrix $\mathbf{A}$ of size $m \times n$, such that the element $\mathbf{A}_{ij}$ is the contribution of light j to sample i. Denote the j-th column of $\mathbf{A}$ by φ_j. The quantity we would like to compute is the sum of all the columns:

$$\Sigma_{\mathbf{A}} = \sum_{j=1}^{n} \varphi_j$$

However, we want to compute this without touching all elements of $\mathbf{A}$, since that would take $O(mn)$ time. Instead, we assume $\mathbf{A}$ is given as an *oracle* that computes elements on demand, and we propose an algorithm that computes an approximation to $\Sigma_{\mathbf{A}}$ in $O(m+n)$ time.

Row-column sampling. The largest cost in evaluating elements of $\mathbf{A}$ is the visibility term: if sample i is not visible from light j, then $\mathbf{A}_{ij}$ is zero. Ray casting can be used to answer such visibility queries, but remains relatively slow. Shadow mapping is a solution to the visibility problem that incurs a constant cost for rendering a depth map at point x, but amortizes it over a large number of queries between x and other points y, which have only minimal cost. This algorithm maps to the GPU naturally. Furthermore, while x is usually a light source and y a surface sample, shadow mapping can be also used to evaluate the contribution of all lights to a sample by computing the depth map centered at the sample location.

In the context of sampling matrix elements, ray casting can evaluate single elements of $\mathbf{A}$ in an arbitrary order, while shadow mapping computes whole rows and columns of $\mathbf{A}$. The per-element cost of the latter is much smaller, but the elements are not available in arbitrary order. Thus we have to design an algorithm that makes efficient use of complete rows and columns.

3.2 Proposed Algorithm

The rest of this paper presents an algorithm that computes r rows and c columns of $\mathbf{A}$ and uses the acquired information efficiently to compute an approximation to $\Sigma_{\mathbf{A}}$. We proceed by partitioning the columns of $\mathbf{A}$ into c clusters, picking a representative column in each cluster with a suitable probability distribution, scaling it appropriately to represent the energy of the entire cluster, and accumulating such representatives. We will see that this is an unbiased Monte Carlo estimator, i.e. the expected value of the computed result is exactly $\Sigma_{\mathbf{A}}$. Determining a good clustering becomes the essential step to increase the performance of the algorithm, which we address by using the rendered rows to drive our cluster selection.

Let $\mathbf{R}$ be the $r \times n$ matrix of randomly picked rows of $\mathbf{A}$, and let ρ_j be the columns of $\mathbf{R}$. (Note: we convert elements of $\mathbf{R}$ from RGB to scalars by taking 2-norms of the RGB triples.) We call ρ_j the *reduced columns*, because they can be viewed as down-sampled versions of the full columns, φ_j. Essentially, $\mathbf{R}$ is the complete matrix for a smaller version of the image. As long as r is large enough, the reduced columns should still preserve enough of the structure of their full counterpart, such that looking for an optimal clustering of the reduced columns should yield a good clustering of the full columns. One might view this idea as a combination of *exploration* (row sampling) and *exploitation* (column sampling), similar to problems arising in machine learning applications.

To summarize, our algorithm (illustrated in Figure 2) consists of the following phases:

- Sample r randomly selected rows using shadow maps (GPU).
- Partition the reduced columns into c clusters (CPU).
- Pick a scaled representative in each cluster (CPU).
- Accumulate these representatives using shadow maps (GPU).

3.3 Interpretation as a Monte Carlo Estimator

The notation used in the rest of the paper is summarized in Table 1. To formalize the above discussion, let's suppose we are already given a clustering, i.e. a partition of the n columns into k clusters $C_1, ..., C_k$. Later we will show how to find this clustering. We will use the norms of the reduced columns, $\|\rho_j\|$, as a measure of the contribution of light j to the image. Moreover, let's denote $s_i := \sum_{j \in C_i} \|\rho_j\|$; s_i can be viewed as a measure of the total energy of the cluster C_i. Our goal is to define a Monte Carlo estimator $X_{\mathbf{A}}$ so that $E[X_{\mathbf{A}}] = \Sigma_{\mathbf{A}}$, the value we are looking for. We do this by defining estimators $X^i_{\mathbf{A}}$ for each cluster C_i, and taking their sum:

$$X_{\mathbf{A}} := \sum_{i=1}^{c} X^i_{\mathbf{A}} \text{ where } X^i_{\mathbf{A}} = \frac{s_i}{\|\rho_j\|}\varphi_j \text{ with prob. } \frac{\|\rho_j\|}{s_i} \text{ for } j \in C_i$$

In other words, we are picking a representative column in each cluster with probability proportional to the reduced column norms. If we assume that all $\|\rho_j\| > 0$, we can see that $X_{\mathbf{A}}$ is indeed an unbiased estimator for $\Sigma_{\mathbf{A}}$:

$$E[X_{\mathbf{A}}] = \sum_{i=1}^{c} E[X^i_{\mathbf{A}}] = \sum_{i=1}^{c} \sum_{j \in C_i} \frac{\|\rho_j\|}{s_i} \frac{s_i}{\|\rho_j\|}\varphi_j = \sum_{i=1}^{c} \sum_{j \in C_i} \varphi_j = \Sigma_{\mathbf{A}}$$

One should note that having reduced columns with $\|\rho_j\| = 0$ does not guarantee that their contribution to the image is indeed zero. The correct (unbiased) solution to this problem would be to put these columns into a special cluster and use a different probability distribution (e.g. uniform, power or similar). In practice we simply ignore these columns, since they are negligible as long as r is large enough.

Our approach to clustering and representative selection is somewhat similar to lightcuts [Walter et al. 2005], with two key differences. First, our reduced columns contain more information than a light's parameters (position, intensity, etc.) – they approximate the actual contribution of the light to the image, including visibility. Second, lightcuts use an adaptive clustering, compared to our global clustering.

3.4 Clustering Objective

Given our Monte Carlo formulation, our clustering attempts to minimize the expected error of $X_{\mathbf{A}}$. We could try a least-squares approach and minimize $E[\|X_{\mathbf{A}} - \Sigma_{\mathbf{A}}\|^2]$. Unfortunately, this would require processing the whole matrix $\mathbf{A}$, which we want to avoid. Instead, we optimize the clustering on the reduced columns and later use the same cluster assignment on the full columns.

To do this, we define *reduced estimators* in an analogous way to the original ones: $X_{\mathbf{R}} := \sum_{i=1}^{c} X^i_{\mathbf{R}}$, where $X^i_{\mathbf{R}}$ takes value $\rho_j s_i / \|\rho_j\|$ with probability $\|\rho_j\|/s_i$ for $j \in C_i$. Denote $\Sigma_{\mathbf{R}} := \sum_{j=1}^{n} \rho_j$. Then we clearly have $E[X_{\mathbf{R}}] = \Sigma_{\mathbf{R}}$, i.e. $X_{\mathbf{R}}$ is an unbiased estimator for $\Sigma_{\mathbf{R}}$.

We are looking for a clustering that minimizes the expected error of $X_{\mathbf{R}}$, that is, $E[\|X_{\mathbf{R}} - \Sigma_{\mathbf{R}}\|^2]$. In Appendix A we prove that this can be writtten as:

$$E[\|X_{\mathbf{R}} - \Sigma_{\mathbf{R}}\|^2] = \frac{1}{2} \sum_{k=1}^{c} \sum_{i,j \in C_k} \|\rho_i\| \cdot \|\rho_j\| \cdot \|\bar{\rho}_i - \bar{\rho}_j\|^2$$

where we denote by $\bar{x}$ the normalized version of a vector x, i.e. $x/\|x\|$. To make this result intuitive, we define the *distance* between two vectors x and y as $d(x,y) = \frac{1}{2}\|x\| \cdot \|y\| \cdot \|\bar{x} - \bar{y}\|^2$ Take a complete graph with n vertices corresponding to the reduced columns, and with edge costs between i and j equal to $d(\rho_i, \rho_j)$. We can now view the problem as follows: partition the graph into c components, such that the sum of the edge costs within the components is minimized. The distance d can be viewed as a measure of how much two lights dislike to be in the same cluster. Note that d is not a metric since it does not satisfy triangle inequality. Yet more intuition can be gained by rewriting d as $d(x,y) = \|x\| \cdot \|y\| \cdot (1 - \cos(x,y))$ where $\cos(x,y) = \bar{x}^T \bar{y}$ is the cosine of the angle between x and y. In other words, the amount of "disagreement" between two lights is proportional to their energy and to the difference between the "kind" of their contribution to the image, which is measured by the angle.

3.5 Minimizing the Clustering Objective

Our objective function is similar to weighted min-sum clustering (also called weighted k-cluster problem), where the input is points $x_1, ..., x_n$ with associated weights $w_1, ..., w_n$ and a number k, and the goal is to partition the points into k clusters, minimizing

$$\sum_{p=1}^{k} \sum_{i,j \in C_p} w_i \cdot w_j \cdot \|x_i - x_j\|^2$$

Our problem is a special case of this one, which can be seen by setting $k := c$, $x_i := \bar{\rho}_i$ and $w_i := \|\rho_i\|$. [Schulman 1999] gives a $(1+\varepsilon)$-pseudo-approximation algorithm which is unfortunately not practical. Instead, we present an approach which works well in practice, even for the large values of n, r and c needed for our application.

Clustering by sampling. A simple algorithm can be derived by randomly picking c points as cluster centers and assigning all other points to the nearest (with respect to d) cluster center. While a naive implementation would provide poor clusterings, we use a modified formulation inspired by [Schulman 1999]. We define α_i to be the cost of all edges in the graph incident to vertex i, i.e.

$$\alpha_i = \sum_{j=1}^{n} d(\rho_i, \rho_j)$$

We make the probability p_i of choosing point i proportional to α_i, intuitively making points that are far away from others more likely to be picked. Let a multi-set be a set with corresponding positive weights for each element. When a point i is picked, we add it to the multi-set with weight $1/p_i$. If the element is already there, we just increase its weight by $1/p_i$. We iterate this process until we have exactly c elements in the multi-set. Finally, we scale the selected cluster centers by their weights from the multi-set, and assign each point to the nearest cluster based on d.

Clustering by top-down splitting. One can derive a different approach that starts with all points in the same cluster, and keeps splitting the cluster with highest cost, until c clusters are reached. Defining how to split is crucial, and we found the following to give

good results: let C be the cluster we are splitting. Project the points $\bar{\rho}_i, i \in C$ onto a random line (in r-dimensional space) and find the best position to cut the line into two pieces. There are only $|C|-1$ possibilities, which can be checked quickly.

Combining the algorithms. We found that running the splitting algorithm on the result of the sampling algorithm is both faster and produces better clusterings than either of the algorithms alone. We produce $\frac{2}{3}c$ clusters by sampling, and then run splitting until c clusters are reached. Intuitively, the sampling algorithm might leave some areas under-sampled, so some clusters could be too large, but the splitting algorithm fixes these problems.

3.6 Low Rank Interpretation

In realistic lighting situations, lights can often be approximated by linear combinations of other lights. Intuitively, this means that $\mathbf{A}$ is close to *low rank*. This intuition can be verified by computing singular values of $\mathbf{A}$ (see Figure 3). Our algorithm benefits from this, since it can be interpreted as producing a rank-c approximation to $\mathbf{A}$ (where each cluster can be viewed as a rank-1 block).

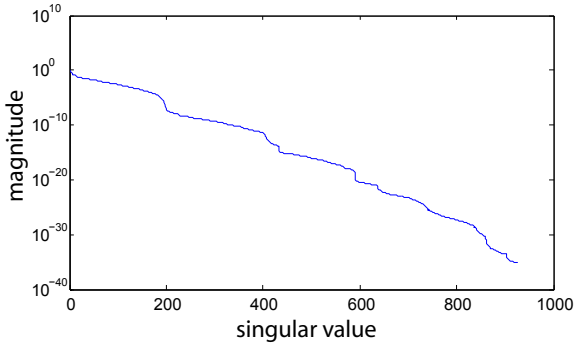

Figure 3: *A logarithmic plot of the singular values of a smaller, 1200 × 1001 version of the matrix corresponding to the temple scene, confirming the intuition that the matrix is close to low rank.*

4 Implementation Details

Row analysis. Rows are selected by stratified uniform sampling: we divide the image into blocks and pick a row in each block. Rows are packed into a matrix $\mathbf{R}$ and analyzed using the clustering algorithm given in Section 3.4. The most expensive operation of the sampling phase of the clustering algorithm is assigning each point to its closest cluster center. We speed this up by noting that d can be rewritten as $d(x,y) = \|x\| \cdot \|y\| - x^T y$. Let $\mathbf{K}$ be an $r \times c$ matrix of cluster centers (as columns). Let $n_{\mathbf{R}}$ be the row vector of column norms of $\mathbf{R}$, and let $n_{\mathbf{K}}$ be the row vector of column norms of $\mathbf{K}$. Distances from every point to every cluster center can be computed all at once by evaluating $n_{\mathbf{K}}^T n_{\mathbf{R}} - \mathbf{K}^T\mathbf{R}$. To further improve performance, we use the optimized BLAS implementation distributed in the Intel MKL. We similarly compute α_i by expressing the distance from every point to every other point as the matrix $n_{\mathbf{R}}^T n_{\mathbf{R}} - \mathbf{R}^T\mathbf{R}$. Since we are only interested in the row sums of this matrix, we can multiply it by a vector of ones and use the associativity of matrix multiplication to simplify the expression.

Random projection. Another speed-up can be obtained by randomly projecting the reduced columns into an r'-dimensional space before running the clustering. This increases performance at the expense of very slight decrease of clustering efficiency. We use $r' = 50$ in our results. For an introduction to random projection techniques see [Vempala 2004].

Representative selection and color averaging. As discussed in Section 3, we can choose a representative of a given cluster i with a probability proportional to the reduced column norm, $\|\rho_j\|$, and then scale the column by $s_i/\|\rho_j\|$. We use a slight practical improvement, which gives the representative the "total color" of the cluster. Define ρ_j^c be the "colored" version of ρ_j and let the 2-norm be defined per-channel. We scale the representative by $(\sum_{j \in C_i} \|\rho_j^c\|)/\|\rho_j^c\|$.

GPU implementation. Programmable shaders are used to evaluate surface and light models, while visibility is computed using shadow maps. In particular, we use cube shadow maps for indirect and omni-directional lights, while standard shadow maps are used for directional and spot lights. Row visibility is evaluated by computing a cube shadow map at the sample location. Rows are rendered one-by-one and read back to the CPU, while columns are accumulated directly without read-back.

Computing surface samples and lights. In our prototype, surface samples are computed using ray-tracing, which takes roughly 10-15 seconds for an 800×600 image with 2×2 anti-aliasing, using an unoptimized Java implementation. However, this sampling could be also implemented on the GPU with execution time comparable to one shadow map. Area lights and environment lighting are handled by stratified uniform sampling, while the sun is converted into a single directional light. Indirect lights are computed using particle tracing as in [Keller 1997]. The results in this paper use about 100,000 lights, which takes roughly one second to sample. For a faster turnaround we could adopt a GPU technique such as [Dachsbacher and Stamminger 2005] to determine the indirect lights, which would only capture one-bounce indirect illumination, but it would eliminate ray-tracing altogether.

5 Results and Discussion

This section presents our results for various scenes rendered using our system. We report the times as measured on a Core 2 Duo 2.6 GHz with an Nvidia GeForce 8800 GPU. (The timings do not include computation of surface samples and lights; we assume these are already given as input.) For comparison we provide reference images computed by accumulating all the lights, which is to say by computing the whole matrix. We also point out some image differences by white arrows; however, it should be kept in mind that randomized algorithms will produce different artifacts in different runs.

Performance. Table 2 presents scene statistics and shows that our algorithm supports the large geometric and lighting complexity required for high-quality previewing. Execution times are dominated by row and column rendering, where shadow mapping constitutes the largest component.

Complex lighting conditions. Figure 4 shows three scenes rendered with our algorithm, where we specifically choose difficult realistic illumination to demonstrate the robustness of our approach. For each scene, we show a fast preview image together with a slower high-quality image displaying the trade-off between time and quality of the converged solution. The *sponza* and *temple* scenes use a sun-sky direct illumination model, where most of the visible scene is lit by indirect and sky illumination only. The *kitchen*

scene	triangles	lights	total (s)		row render (s)		clustering (s)		col. render (s)		shadow map (s)	
			100/100	300/900	100/100	300/900	100/100	300/900	100/100	300/900	100/100	300/900
sponza	66,454	100,001	1.6	7.9	0.9	2.9	0.2	0.6	0.5	4.4	0.6	3.1
trees	328,126	100,002	2.2	9.9	1.2	3.4	0.5	4.9	0.6	4.9	0.9	4.7
kitchen	388,454	101,089	2.3	12.0	1.0	3.0	0.3	1.3	0.9	7.8	0.9	5.7
bunny	869,483	100,489	3.2	12.6	2.1	6.2	0.5	2.1	0.5	4.1	2.1	7.9
grand central	1,526,555	100,836	3.2	16.1	1.5	4.2	0.6	1.9	1.2	10.0	0.8	4.0
temple	2,214,021	100,001	3.2	17.0	1.7	4.9	0.2	0.6	1.3	11.4	2.0	12.0

Table 2: *Detailed statistics for scenes rendered using our algorithm at a resolution of 800 × 600 with 2 × 2 supersampled antialiasing. We report geometry size and total number of lights. For timings we present total time in seconds for different numbers of rows and columns. We further give detailed timing breakdowns for rendering rows, clustering, and rendering columns. We also give total time for shadow mapping.*

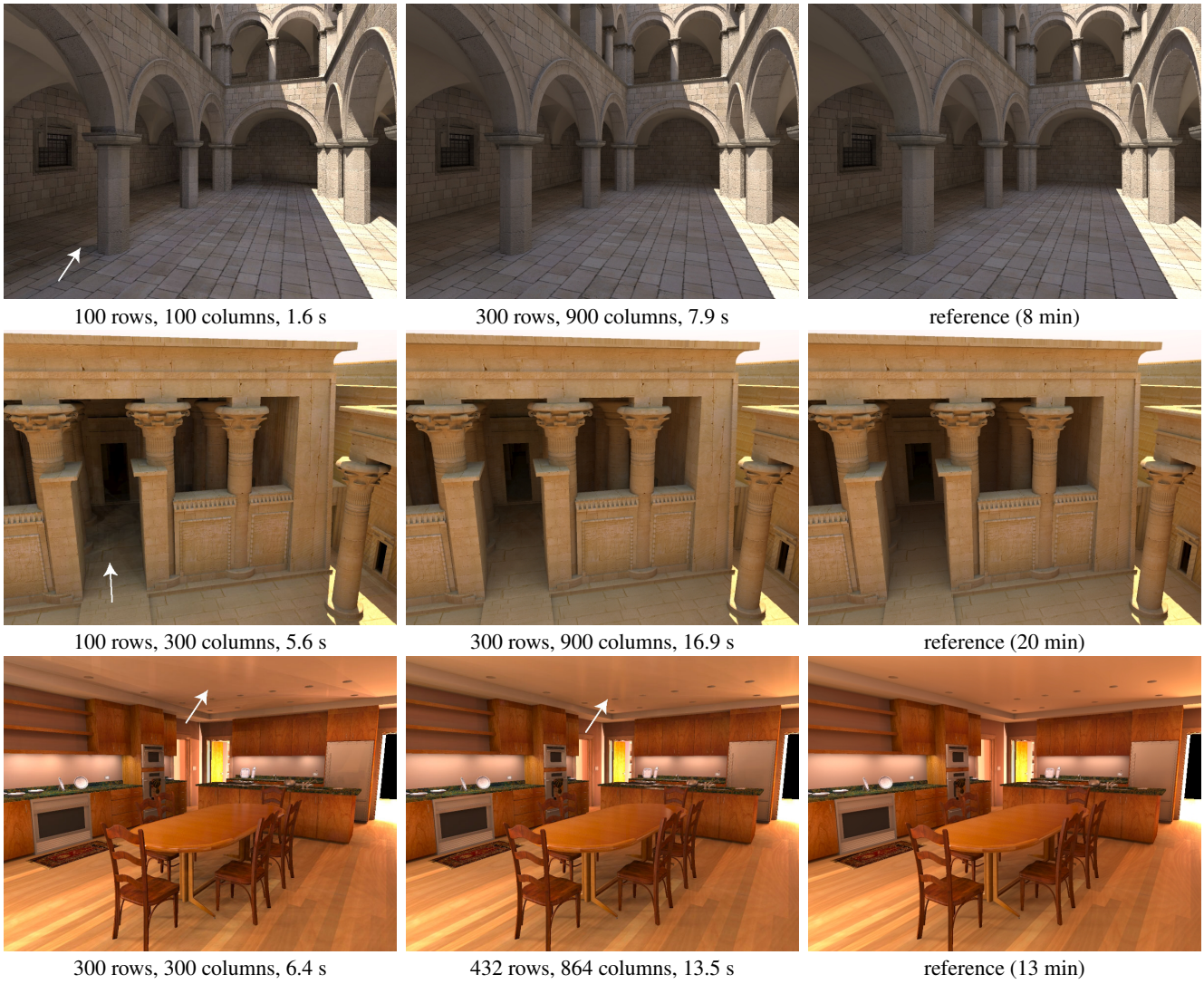

Figure 4: *Images rendered with our system with difficult lighting configurations for the Sponza, temple and kitchen scenes. We show two different settings, and a reference image computed by accumulating all the lights. We report the number of rows and columns used, and the rendering time. Note that these timings do not include computation of surface samples and lights, which we treat as input to the algorithm.*

scene is lit by area lights above the counters and by strong omni-directional lights behind the corners, making the image dominated by indirect illumination. Furthermore, most of the materials have a glossy component (Ward model). Note how indirect shadows are handled correctly, e.g. on the kitchen floor, and on the detailed geometry of the temple pylons.

Exploration vs. exploitation. Our framework is built on the assumption that exploration (in the form of row computation) is worthwhile, even though it may take considerable time. An alternative approach would be to skip exploration altogether and dedicate resources to render more columns. To prove the effectiveness of our row sampling strategy, we compare our algorithm to:

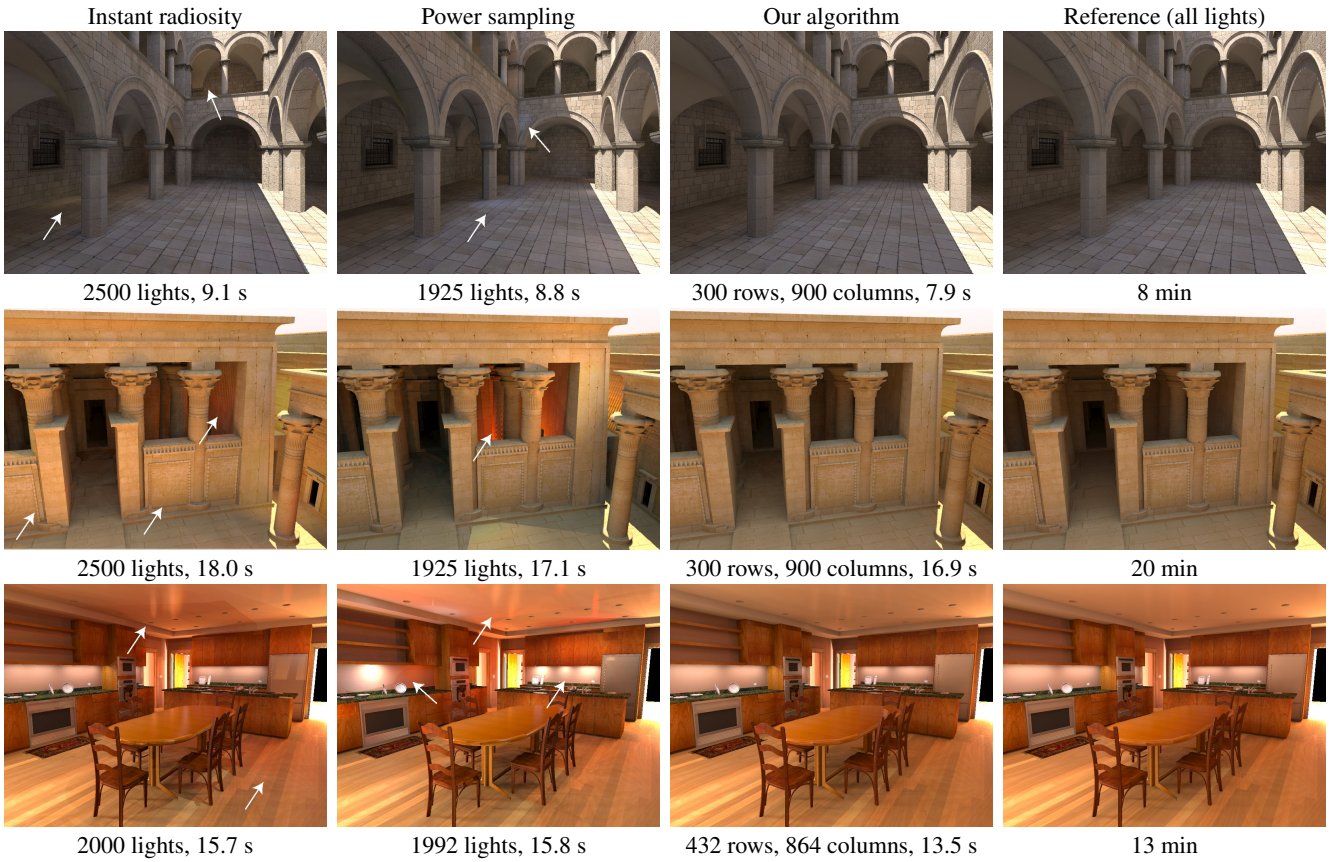

Figure 5: *Equal-time image comparison of instant radiosity, power sampling and our algorithm. The last column shows reference images.*

- an "instant radiosity"-type algorithm (IR), where we generate a smaller number of lights but render them exhaustively;
- a "power sampling" algorithm (PS), which solves the many-light problem by picking lights with probability proportional to their power (which can be thought of as sampling the columns of the matrix, without any information from the rows).

An equal time comparison of the results of these algorithms is shown in Figure 5, showing the benefit of employing row-sampling.

Figure 6 plots the error of the three algorithms as a function of execution time. For our algorithm we report four result curves corresponding to different ratios between rows and columns. The reported error is computed as the 2-norm of the difference from the reference image, normalized by the reference 2-norm. The *temple* and *sponza* graphs show that our algorithm consistently computes better solutions than either of the two prior ones. The graphs for the IR and PS algorithms do not decrease monotonically, showing the high variance of the error. On the other hand, our technique shows lower variance together with lower error.

The *kitchen* graph shows a fundamental property of our algorithm: the error drops rapidly when more resources are allocated to row sampling. This behavior is logical because a certain threshold number of rows is needed to find all interesting lighting features present in the image. Determining that number automatically without any prior knowledge of the matrix is not trivial. In practice we run most of our test cases with either a 1:2 or 1:3 ratios between rows and columns. This problem of determining the right trade-off between parameters is common in many rendering algorithms; e.g. determining the number of photons in photon mapping.

Coherence vs. low rank. Figure 7 shows two scenes which illustrate complex incoherent geometry, showing that our algorithm performs well even for scenes that do not show image-space coherence (high-frequency rows) or lighting-space coherence (high-frequency columns). Most algorithms based on interpolation of illumination would not necessarily perform well on these scenes. Our algorithm instead exploits the low-rank nature of the lighting matrix, which is not equivalent to coherence in geometry or lighting.

Low-rank assumption. The *Grand Central* scene (Figure 8) is an example of a scene that does not match our low-rank matrix assumption as well as other scenes. This is because of the omni-directional point lights positioned in small recesses between stone blocks. Columns corresponding to these lights cannot be well approximated as linear combinations of other nearby ones. However, the reduced columns corresponding to these lights are very similar, so the algorithm is eager to cluster them. While, as expected, our algorithm will require a higher number of rows and columns, it will eventually converge to the correct solution, showing that we can successfully capture even these adversarial cases.

Comparison with Lightcuts. The *Lightcuts* [Walter et al. 2005; Walter et al. 2006] framework presents a scalable solution to the many-light problem that uses a CPU raytracer to evaluate visibility. Our matrix sampling approach and Lightcuts solve the same mathematical problem, but operate at different points of the performance-quality tradeoff. An important difference is that Lightcuts adapt the clustering to the sample being shaded, while we use one global clustering to make the algorithm GPU-friendly. While directly compar-

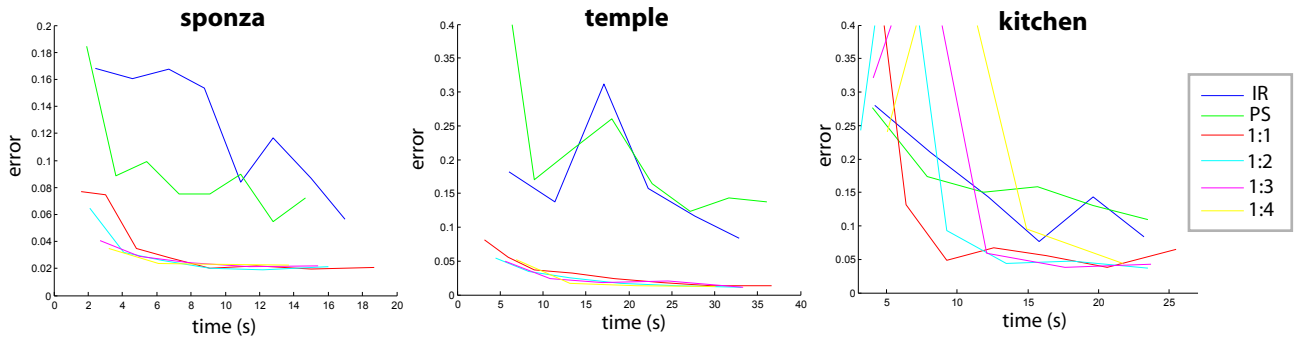

Figure 6: *Comparison of error as a function of execution time for instant radiosity (IR), power sampling (PS) and 4 versions of our algorithm with different row-column ratios.*

100 rows, 200 columns, 2.9 s | reference (14 min) | 100 rows, 200 columns, 3.8 s | reference (10 min)

Figure 7: *Images rendered with our system showing high-frequency lighting and geometric details captured efficiently by our algorithm for the trees and bunny scenes. The bunny scene shows a shader implementing the Kajiya-Kay hair shading model.*

ing the execution times of different CPU vs. GPU based systems is tricky, we have run the reconstruction cut algorithm (which exploits image space coherence) from Lightcuts for the scenes in Figure 4 on the same machine used for our results. We obtain speed improvements in the range between 20x and 30x for our high-quality images (middle column of Figure 4).

Discussion and Limitations. One of the drawbacks of our approach is that shadow mapping artifacts may be present. In particular shadow bias is an issue, since there might not exist a single bias setting that works for all 100 thousand automatically generated lights. Moreover, the conversion of indirect illumination to point lights requires clamping, common to all similar approaches [Keller 1997; Walter et al. 2005]. This leads to slight darkening of indirect lighting, especially in corners. Our framework does not specifically address this limitation of the many-light formulation, treating indirect lights like any other programmable shader. Furthermore, while our algorithm is mostly designed for previewing single frames, we would like to explore rendering animations. Currently, slight temporal artifacts might be seen due to the Monte Carlo nature of the algorithm, and can be remedied by increasing the number of samples; however, we are interested in developing a more temporally stable version of the algorithm. Finally, it would be interesting to investigate automatic selection of r and c, the number of rows and columns. For columns, this is not conceptually difficult: instead of minimizing expected error for a given c, we can search for the minimum c that satisfies an error threshold. For rows, this is harder to do; however, just as rows contain information about which columns to pick, columns can suggest which rows to pick. Therefore, we are investigating a promising variation of the algorithm that alternates row and column sampling.

6 Conclusion

We have presented an algorithm to compute fast and high-quality solutions to the many-light problem, which we treat as the problem of approximating the sum of all columns of an unknown matrix. We explore the matrix structure by sampling a small set of rows, and reconstruct the image by rendering a small set of representative columns. Our framework explicitly takes advantage of GPU acceleration and requires no precomputation. Since complex and arbitrary object appearance can be expressed in the context of the many-light problem, we believe our approach could have compelling applications in cinematic and architectural lighting design.

Acknowledgments

We thank Veronica Sundstedt and Patrick Ledda for the temple model, and Bruce Walter and the Program of Computer Graphics for support. This work was supported by NSF CAREER 0644175 and Affinito-Stewart Award.

Appendix A

We can rewrite $E[\|X_{\mathbf{R}} - \Sigma_{\mathbf{R}}\|^2]$ by noting that the $X^i_{\mathbf{R}}$ are independent, from which it follows that the expected error of $X_{\mathbf{R}}$ is the sum of the expected errors of the $X^i_{\mathbf{R}}$. Moreover, we note that for a random variable X and its independent, identically distributed "clone"

300 rows, 600 columns, 12.7 s | 588 rows, 1176 columns, 24.2 s | reference (44 min)

Figure 8: *The Grand Central scene is an example of a scene that does not match our low-rank assumption. While our algorithm will require more samples, it continues to provide a good quality preview and will converge to the solution even in this adversarial case.*

X', we have

$$E[\|X - E[X]\|^2] = \frac{1}{2} E[\|X - X'\|^2]$$

We can thus write our clustering metric as:

$$\begin{aligned} E[\|X_{\mathbf{R}} - \Sigma_{\mathbf{R}}\|^2] &= \sum_{k=1}^{c} E[\|X_{\mathbf{R}}^k - E[X_{\mathbf{R}}^k]\|^2] \\ &= \frac{1}{2} \sum_{k=1}^{c} E[\|X_{\mathbf{R}}^k - X_{\mathbf{R}}^{k\prime}\|^2] \\ &= \frac{1}{2} \sum_{k=1}^{c} \sum_{i,j \in C_k} \frac{\|\rho_i\|}{s_k} \cdot \frac{\|\rho_j\|}{s_k} \cdot \| \frac{s_k}{\|\rho_i\|} \rho_i - \frac{s_k}{\|\rho_j\|} \rho_j \|^2 \\ &= \frac{1}{2} \sum_{k=1}^{c} \sum_{i,j \in C_k} \|\rho_i\| \cdot \|\rho_j\| \cdot \|\bar{\rho}_i - \bar{\rho}_j\|^2 \end{aligned}$$

References

AGARWAL, S., RAMAMOORTHI, R., BELONGIE, S., AND JENSEN, H. W. 2003. Structured importance sampling of environment maps. *ACM Transactions on Graphics 22*, 3 (July), 605–612.

BALA, K., WALTER, B. J., AND GREENBERG, D. P. 2003. Combining edges and points for interactive high-quality rendering. *ACM Transactions on Graphics 22*, 3 (July), 631–640.

BEN-ARTZI, A., OVERBECK, R., AND RAMAMOORTHI, R. 2006. Real-time brdf editing in complex lighting. *ACM Transactions on Graphics 25*, 3 (July), 945–954.

CARR, N. A., HALL, J. D., AND HART, J. C. 2003. Gpu algorithms for radiosity and subsurface scattering. In *Graphics Hardware 2003*, 51–59.

CHRISTENSEN, P. H., LISCHINSKI, D., STOLLNITZ, E. J., AND SALESIN, D. H. 1997. Clustering for glossy global illumination. *ACM Transactions on Graphics 16*, 1, 3–33.

DACHSBACHER, C., AND STAMMINGER, M. 2005. Reflective shadow maps. In *SI3D '05: Proceedings of the 2005 symposium on Interactive 3D graphics and games*, 203–231.

DACHSBACHER, C., AND STAMMINGER, M. 2006. Splatting of indirect illumination. In *2006 ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*.

DRINEAS, P., MAHONEY, M. W., AND MUTHUKRISHNAN, S. 2006. Subspace sampling and relative-error matrix approximation: Column-row-based methods. In *Algorithms - ESA 2006*, 304–314.

FERNANDEZ, S., BALA, K., AND GREENBERG, D. P. 2002. Local illumination environments for direct lighting acceleration. In *Rendering Techniques 2002: 13th Eurographics Workshop on Rendering*, 7–14.

GAUTRON, P., KRIVÁNEK, J., BOUATOUCH, K., AND PATTANAIK, S. 2005. Radiance cache splatting: A gpu-friendly global illumination algorithm. In *Rendering Techniques*, 55–64.

GAUTRON, P., BOUATOUCH, K., AND PATTANAIK, S. 2006. Temporal radiance caching. In *SIGGRAPH 2006 Sketches*.

GOREINOV, S. A., TYRTYSHNIKOV, E. E., AND ZAMARASHKIN, N. 1997. A theory of pseudoskeleton approximations. *Linear Algebra and its Applications 261*.

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

HAŠAN, M., PELLACINI, F., AND BALA, K. 2006. Direct-to-indirect transfer for cinematic relighting. *ACM Transactions on Graphics 25*, 3 (July), 1089–1097.

JENSEN, H. W. 2001. *Realistic image synthesis using photon mapping*. A. K. Peters, Ltd., Natick, MA, USA.

KELLER, A. 1997. Instant radiosity. In *Proceedings of SIGGRAPH 97*, Computer Graphics Proceedings, Annual Conference Series, 49–56.

KŘIVÁNEK, J., BOUATOUCH, K., PATTANAIK, S. N., AND ŽÁRA, J. 2006. Making radiance and irradiance caching practical: Adaptive caching and neighbor clamping. In *Eurographics Symposium on Rendering*.

NG, R., RAMAMOORTHI, R., AND HANRAHAN, P. 2004. Triple product wavelet integrals for all-frequency relighting. *ACM Transactions on Graphics* (July).

OSTROMOUKHOV, V., DONOHUE, C., AND JODOIN, P.-M. 2004. Fast hierarchical importance sampling with blue noise properties. *ACM Transactions on Graphics 23*, 3, 488–495.

PAQUETTE, E., POULIN, P., AND DRETTAKIS, G. 1998. A light hierarchy for fast rendering of scenes with many lights. *Computer Graphics Forum 17*, 3, 63–74.

PURCELL, T. J., DONNER, C., CAMMARANO, M., JENSEN, H. W., AND HANRAHAN, P. 2003. Photon mapping on programmable graphics hardware. In *Sypmosium on Graphics Hardware 2003*.

SCHEEL, A., STAMMINGER, M., AND SEIDEL, H.-P. 2001. Thrifty final gather radiosity. *Proceedings of the 12th Eurographics Workshop on Rendering*.

SCHEEL, A., STAMMINGER, M., AND SEIDEL, H. 2002. Grid based final gather for radiosity on complex clustered scenes. *Computer Graphics Forum, 21(3)*.

SCHULMAN, L. J. 1999. Clustering for edge-cost minimization. *Electronic Colloquium on Computational Complexity (ECCC) 6*, 035.

SHIRLEY, P., WANG, C., AND ZIMMERMAN, K. 1996. Monte carlo techniques for direct lighting calculations. *ACM Transactions on Graphics 15*, 1 (Jan.), 1–36.

SLOAN, P.-P., KAUTZ, J., AND SNYDER, J. 2002. Precomputed radiance transfer for real-time rendering in dynamic, low-frequency lighting environments. *ACM Transactions on Graphics 21*, 3 (July), 527–536.

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

TOLE, P., PELLACINI, F., WALTER, B., AND GREENBERG, D. P. 2002. Interactive global illumination in dynamic scenes. *ACM Transactions on Graphics 21*, 3 (July), 537–546.

VEMPALA, S. 2004. *The Random Projection Method*. DIMACS Series in Discrete Mathematics and Theoretical Computer Science.

WALD, I., BENTHIN, C., AND SLUSALLEK, P. 2003. Interactive global illumination in complex and highly occluded environments. In *14th Eurographics Workshop on Rendering*, 74–81.

WALTER, B., DRETTAKIS, G., AND PARKER, S. 1999. Interactive rendering using the render cache. In *Eurographics Rendering Workshop 1999*.

WALTER, B., FERNANDEZ, S., ARBREE, A., BALA, K., DONIKIAN, M., AND GREENBERG, D. P. 2005. Lightcuts: a scalable approach to illumination. *ACM Transactions on Graphics 24*, 3 (Aug.), 1098–1107.

WALTER, B., ARBREE, A., BALA, K., AND GREENBERG, D. P. 2006. Multidimensional lightcuts. *ACM Transactions on Graphics 25*, 3 (July), 1081–1088.

WARD, G., AND SIMMONS, M. 1999. The holodeck ray cache: An interactive rendering system for global illumination in non-diffuse environments. *ACM Transactions on Graphics 18*, 4 (Oct.), 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.

WARD, G. 1994. Adaptive shadow testing for ray tracing. In *Proceedings of the Second Eurographics Workshop on Rendering*, 11–20.