Fast, Realistic Lighting and Material Design using Nonlinear Cut Approximation

Ewen Cheslack-Postava
Stanford University

Rui Wang
UMass Amherst

Oskar Akerlund
Linkoping University

Fabio Pellacini
Dartmouth College

(a) (b) (c)

Figure 1: *Three examples of realistic lighting and material design captured using our system at 2∼4 fps. The user can dynamically modify the lighting, viewpoint, BRDF and per-pixel shading parameters.*

Abstract

We present an efficient computational algorithm for functions represented by a nonlinear piecewise constant approximation called *cuts*. Our main contribution is a single traversal algorithm for merging cuts that allows for arbitrary pointwise computation, such as addition, multiplication, linear interpolation, and multi-product integration. A theoretical error bound of this approach can be proved using a statistical interpretation of cuts. Our algorithm extends naturally to computation with many cuts and maps easily to modern GPUs, leading to significant advantages over existing methods based on wavelet approximation. We apply this technique to the problem of realistic lighting and material design under complex illumination with arbitrary BRDFs. Our system smoothly integrates all-frequency relighting of shadows and reflections with dynamic per-pixel shading effects, such as bump mapping and spatially varying BRDFs. This combination of capabilities is typically missing in current systems. We represent illumination and precomputed visibility as nonlinear sparse vectors; we then use our cut merging algorithm to simultaneously interpolate visibility cuts at each pixel, and compute the triple product integral of the illumination, interpolated visibility, and dynamic BRDF samples. Finally, we present a two-pass, data-driven approach that exploits pilot visibility samples to optimize the construction of the light tree, leading to more efficient cuts and reduced datasets.

1 Introduction

Realistic image synthesis requires incorporating scalable illumination, complex shadowing, and detailed materials. The simulation costs of such computation often prevent users from receiving real-time feedback on both lighting and material changes. Previous work [Sloan et al. 2002; Ng et al. 2003] has shown that expensive rendering costs can be amortized by precomputing illumination transport data for a static scene. Early prototypes of such algorithms focused on the capability of *relighting*, but require fixing material properties to avoid excessive sampling costs. Subsequent work, such as [Ng et al. 2004], permits dynamic selection of materials, but only from a limited library, mainly due to the cost of projecting BRDFs onto bases such as wavelets. Recent advances in material editing [Ben-Artzi et al. 2006; Sun et al. 2007] provide techniques for real-time BRDF manipulation but rely heavily on assumptions such as fixed lighting and view or the existence of a small linear basis for general BRDFs. In addition, dynamic control of meso-scale surface details is generally missing in the these systems due to the lack of support for per-pixel shading.

In order to lift these restrictions, it is necessary to efficiently interpolate visibility and evaluate the triple product integral using a basis in which dynamic BRDFs can be quickly evaluated. We build on the existing *cuts* framework [Walter et al. 2005; Akerlund et al. 2007; Arbree et al. 2008], a nonlinear piecewise constant representation. Our key contributions are:

- A general cut merging algorithm that is used to perform arbitrary pointwise operations on any number of cuts. This directly competes with and improves upon wavelet-based methods – the current state of the art in nonlinear approximation.
- A theoretical error analysis of multi-product integrals of cuts. We provide an error bound for double product integrals and an approximate error bound for triple product integrals. We are not aware of any similar analysis for wavelets.
- A specific application of our method to PRT that enables all-frequency relighting effects with dynamic per-pixel effects. Figure 1 shows several examples.
- A data-driven clustering algorithm which exploits pilot visibility samples to optimize the light tree construction and improve efficiency.

Our relighting and material editing system is built upon the cuts framework with the following goals in mind:

- Full dynamic control of lighting, view point, and BRDFs;
- Per-pixel shading details using bump mapping and spatially varying BRDF parameters, all editable at runtime with no BRDF precomputation;

- Flexible illumination sources.

Although prior work has shown subsets of the above features, the complete set is typically missing in existing systems. Our basic assumption is that scene geometry is fixed, thus visibility can be precomputed. We also assume that complex illumination can be approximated by many point lights, as suggested by recent work [Walter et al. 2005; Walter et al. 2006]. Our system supports rendering using environment lighting or one bounce diffuse interreflection from any direct lighting model which can be efficiently evaluated, similar to Hašan et al. [2006].

2 Related Work

Illumination from many lights. Efficient rendering from a large number of light sources has been a central challenge in graphics. Standard algorithms entail a linear cost with the number of lights. To reduce this cost, several methods [Ward 1994; Shirley et al. 1996] have been introduced to intelligently pick important lights; [Agarwal et al. 2003; Debevec 2005; Clarberg et al. 2005] create importance-based illumination sampling schemes or cluster many lights into a small set of representative lights. Recently, Walter et al. [2005] introduced Lightcuts – a scalable solution for handling many lights using a hierarchical algorithm. These techniques typically rely on offline renderers to sample visibility and achieve high quality. The matrix row-column sampling algorithm [Hašan et al. 2007] exploits the GPU to sample and cluster many lights, improving the speed to a few seconds per frame. In contrast, our method aims at exploiting precomputed visibility to support real-time lighting and material changes.

There is a large body of recent work on GPU-based global illumination, such as [Nijasure et al. 2005; Gautron et al. 2005; Laine et al. 2007; Dachsbacher et al. 2007]. These techniques use the GPU for fast illumination, at the cost of reduced rendering quality.

Precomputed light transport. PRT [Sloan et al. 2002; Ng et al. 2003] amortizes expensive rendering costs with many lights by precomputing illumination transport data for a static scene. A comprehensive overview of current PRT techniques can be found in [Lehtinen 2007]. Early work focused on distant environment illumination [Kautz et al. 2002; Lehtinen and Kautz 2003; Liu et al. 2004; Wang et al. 2004] or mid-range illumination [Sloan et al. 2002; Annen et al. 2004]. They take dense illumination samples and exploit bases such as spherical harmonics (SH), Haar wavelets, or radial basis functions (RBFs) to compress precomputed light transport data. A major drawback of these bases is that they are difficult to apply to an arbitrary illumination domain, such as in the case of indirect illumination where the lighting is represented by an unstructured point set. Recent work by Hašan et al. [2006] introduced a scheme to impose a hierarchical quad-tree structure on a point set for wavelet compression. Most recently, Akerlund et al. [2007] presented *precomputed visibility cuts* – a clustering-based approach that uses nonlinear piecewise constants to approximate arbitrary illumination functions. Their approach is accurate and has the advantage of permitting dynamic BRDF samples evaluated on the fly, thus enabling online material editing.

Our algorithm is based on the cuts representation. This idea has been explored in [Akerlund et al. 2007], but we are not aware of any previous study of efficient computational algorithms involving multiple cuts. Two closely related papers are the wavelet triple product integral algorithm [Ng et al. 2004] and the generalized version [Sun and Mukherjee 2006]. As noted by both, the use of wavelets makes it very challenging to incorporate local lighting and dynamic BRDFs; in addition, the multi-product wavelet integral algorithm is difficult to map to the GPU.

BRDF shading and editing. Realistic BRDF shading under large-scale illumination has been thoroughly studied. Environment mapping [Cabral et al. 1999; Ramamoorthi and Hanrahan 2002] is one popular technique that assumes no shadowing effects; early work in PRT incorporates shadows but requires fixing material properties. By adopting a factored representation that decouples materials from the scene, Ng et al. [2004] enable dynamic selection of materials, but only a small number are supported due to precomputation cost. In general, these systems focus more on the capability of *relighting* but do not provide *material editing*.

Colbert et al. [2006] developed *BRDFShop* – an intuitive BRDF editing interface; Lawrence et al. [2004] introduced the inverse shade tree for non-parametric BRDF editing; Kautz et al. [2007] presented an interactive editing system for BTFs. Most recently, advances in BRDF editing [Ben-Artzi et al. 2008; Sun et al. 2007] have enabled real-time BRDF changes under large-scale illumination; but due to the high cost of sampling view-dependent effects, they typically make heavy assumptions such as fixed lighting and view, or the existence of a small linear basis for general BRDFs. In addition, due to the lack of per-pixel shading support, they ignore meso-scale surface details provided by bump maps, spatially varying BRDFs or BTFs. Recent work by Ritschel et al. [2007] precomputes visibility data and casts dynamic BRDF samples to simulate complex reflections. Their system is unsuitable for large-scale illumination, requiring several seconds to converge. Sloan et al. [2003] introduced bi-scale radiance transfer to incorporate BTFs in SH-based PRT: a macro-scale transfer smoothly interpolates global effects, and a meso-scale transfer provides local shading details. Their system is limited to low-frequency shadow and reflection effects and requires preprocessing of texture functions. Our approach is conceptually similar to theirs, but we handle all-frequency effects throughout the system and permit dynamic material changes with no preprocessed BRDF data.

3 Algorithm Overview

Assumptions. Similar to previous work, we make the following assumptions: 1) the scene geometry is fixed; 2) the illumination can be modeled as many diffuse (isotropically radiating) point lights; 3) we only consider direct illumination from the point lights.

Our system supports two illumination modes under the second assumption. The first mode handles only direct lighting from an environment source, approximated as a set of infinitely distant point lights. The second mode uses a direct to indirect framework in which direct lighting is computed efficiently using shadow mapping, and our technique computes one diffuse bounce of indirect lighting; the final rendering includes the full contribution for direct lighting and a single indirect diffuse bounce.

3.1 Mathematical Framework

Given a set of point lights $\mathcal{S}$ (each associated with either a solid angle in the environmental lighting mode, or a small surface area with normal, in the indirect lighting mode), the reflected radiance B caused by direct illumination from $\mathcal{S}$ to a surface point $\mathbf{x}_o$ is:

$$B(\mathbf{x}_o, \omega) = \sum_{\mathcal{S}} L(\mathbf{x}_i)\, f_r(\mathbf{x}_i \rightarrow \mathbf{x}_o, \omega)\, V(\mathbf{x}_i)\, G(\mathbf{x}_i)\, \cos\theta_i \quad (1)$$

where ω is the view direction, $\mathbf{x}_i \in \mathcal{S}$ is a point light, L is the source radiance, f_r is the BRDF, V is the binary visibility function, and $G(\mathbf{x}_i)$ is the solid angle of $\mathbf{x}_i$ as observed at $\mathbf{x}_o$. To simplify the notation, we combine $\cos\theta_i$ into f_r, and combine G into V. We then focus on a fixed surface point at a fixed viewing angle, thus obtaining the simple form:

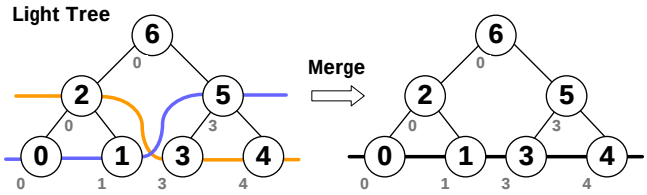


Figure 2: *A light tree and two example cuts. Leaf nodes are individual light samples and interior nodes are clusters. Each tree node is indexed by the postorder traversal index (the center number); it also stores the index to its leftmost child leaf (the lower-left number). A cut (colored lines) represents a partitioning of the leaves into clusters. Two cuts can be merged into a new cut.*

$$B = \sum_{\mathcal{S}} L(\mathbf{x}_i)\, f_r(\mathbf{x}_i)\, V(\mathbf{x}_i) \tag{2}$$

As $|\mathcal{S}|$ is large, directly evaluating this integral is impractical. A large body of previous work, summarized by [Ng et al. 2004], has studied using bases such as Haar wavelets and spherical harmonics to approximate each term and reduce the computational cost to be sublinear in the number of lights. However, these approaches are typically limited to distant environment lighting and require precomputing BRDF data.

Representation using cuts. Recently Akerlund et al. [2007] presented an alternative representation called *cuts* that uses piecewise constants to approximate functions over the domain $\mathcal{S}$. For example, V can be approximated by partitioning $\mathcal{S}$ into a small number of coherent clusters, assigning each cluster it's *mean* value:

$$\langle v_k \rangle = \frac{1}{|\mathcal{C}_k|} \sum V(\mathbf{x}_i), \quad \mathbf{x}_i \in \mathcal{C}_k \tag{3}$$

where $\langle \cdot \rangle$ denotes the mean, and $|\mathcal{C}_k|$ denotes the number of clustered samples. The cluster center $\langle \mathbf{x}_k \rangle = \frac{1}{|\mathcal{C}_k|} \sum \mathbf{x}_i$ is also computed as a representative direction. To create the clusters, a binary *light tree* is built from the sample set $\mathcal{S}$, with individual samples at leaves and clusters at interior nodes. Next, V is sampled at each leaf node, and aggregated to interior nodes. A *cut* through the tree is selected to represent a partitioning $\mathcal{S}$ into disjoint sets, as in Figure 2, resulting in a piecewise constant approximation of V.

In [Akerlund et al. 2007], only visibility functions are approximated as *precomputed visibility cuts*; the run-time algorithm then traverses each cut, summing up its illumination contribution at each node to compute the final rendering result. They create clusters purely based on visibility, ignoring potential errors in the lighting and BRDF. In addition, as the precomputation and shading are both performed at vertex level, they do not support per-pixel shading.

In fact, there is no reason why one cannot use cuts to approximate other functions. For example, we can create an *illumination cut* for L, and a *BRDF cut* for f_r. The main challenge is how to efficiently evaluate the product integral in Eq. 2. This general problem of computing the triple product integral has been thoroughly studied by [Ng et al. 2004] in the context of wavelet approximation. Unfortunately, wavelet-based methods generally lead to algorithms that are ill-suited for GPU implementation and have not been demonstrated at interactive rates.

We found that a very simple and efficient algorithm exists to quickly evaluate the triple product integral for cuts; in fact, the algorithm supports an arbitrary number of cuts. As explained in the next section, the result of this computation is effectively a new, *merged cut* that combines the finest subclusters of all source cuts. With the merged cut, Eq. 2 becomes:

$$B = \textstyle\sum_k |\mathcal{C}'_k| \, \langle l_k \rangle \, \langle \rho_k \rangle \, \langle v_k \rangle \tag{4}$$

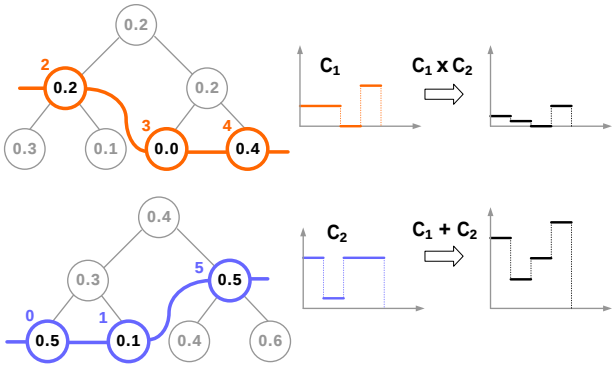


Figure 3: *Merging cuts. The left column shows two cuts C_1, C_2. The node value is the cluster mean (average); the upper-left colored number is the cluster index. The middle column shows the effective cut approximations. The right column shows the results of summing and multiplying the two cuts.*

Figure 4: *Step-by-step execution of the example given above. Cuts are shown by their node indices: 2, 3, 4 for the orange cut; 0, 1, 5 for the blue cut. Shaded nodes are the current nodes in execution; underscore marks the smallest index at each step; checkmarks indicate whether the nodes overlap or not.*

where $\mathcal{C}'_k$ denotes a merged subcluster, and $\langle l_k \rangle$, $\langle \rho_k \rangle$, $\langle v_k \rangle$ are the cluster means of L, f_r and V. Because the accuracy of source cuts is maintained the result of this approximation has a theoretical error bound, as derived in Section 4.

3.2 Efficient Algorithm for Merging Cuts

Properties of cuts. First we introduce some notation. As shown in Figure 2, each node in the light tree is indexed by its *postorder traversal index*; for each node we also keep the postorder index of its leftmost leaf child, which we call the *leftleaf*. From the definition of postorder index, we can verify the following properties:

1. The children of any tree node $\mathbf{p}$ *must have indices that range between* $[\mathbf{p}.leftleaf, \mathbf{p}.index)$*, referred to as the child index range;*

2. If $\mathbf{p_1}.index < \mathbf{p_2}.index$*, then either* $\mathbf{p_1}$ *is a child of* $\mathbf{p_2}$ *(this happens when* $\mathbf{p_1}.index \geq \mathbf{p_2}.leftleaf$*); or* $\mathbf{p_1}$ *represents a disjoint cluster on the left side of* $\mathbf{p_2}$ *in the tree.*

3. Indices of cut nodes increase monotonically from left to right;

4. The union of any number of cuts is a new cut that goes through the deepest subclusters in all cuts.

Algorithm for merging cuts. From Figure 3, it is easy to see that both sum and multiplication of two cuts result in a merged cut representing a new piecewise constant approximation. Therefore we can use the same algorithm, called *cut merging*, for both computations. From Properties 3 and 4, we observe that merging cuts very much resembles merging sorted sparse vectors. The standard sparse vector merging algorithm works as follow: start by setting a pointer to the first element of each array; begin looping, find those elements with the smallest index from all current pointers, merge their values into a result buffer, increment their pointers, and con-

tinue until it has reached the end of all arrays. Note that elements with different indices are not merged together.

This algorithm is very efficient as it only requires one linear traversal performed simultaneously on each array. However, to use it for merging cuts, we must consider the case where cut nodes that do not share the same index can overlap in space such as when one is the child of the other. In this case their values should be merged together. Given this, we modify the algorithm as follows: as before, at each step we find and advance those nodes with the smallest index; however, the contribution of *all* current nodes will be merged as long as they overlap with each other. Here, overlapping means they share at least one common leaf node. According to Property 1, we can quickly check for overlapping nodes by intersecting the child index range of all nodes: the intersection is non-zero if and only if all nodes overlap. With this modification, we now have the complete algorithm, presented below. The algorithm works for an arbitrary number (N) of cuts.

---

set $\mathbf{p}_j$ ($j \in [1, N]$) to point to the first node in each cut;
set $c = 0$;
loop
  min_index = **minimum** ($\mathbf{p}_1$.index, ... , $\mathbf{p}_N$.index);
  max_leftleaf = **maximum** ($\mathbf{p}_1$.leftleaf, ... , $\mathbf{p}_N$.leftleaf);
  if max_leftleaf $\leq$ min_index **then**
    /* all cut nodes overlap */
    perform desired computation, e.g. $c = c + \prod_j \langle \mathbf{p}_j \rangle$;
  end if
  for all j such that ($\mathbf{p}_j$.index == min_index) **do**
    advance $\mathbf{p}_j$ to its next node;
    if ($\mathbf{p}_j >$ the end of cut j) **then** quit loop;
  end for
end loop
return c;

The above example computes multi-product integral of N cuts. To perform linear interpolation instead, we can simply change the inner computation to $\sum_j w_j \langle \mathbf{p}_j \rangle$, where w_j is the interpolation weight. In Figure 3 and 4 we show a detailed example of running this algorithm. We can also change the computation to a mixture of sums and products involving the N cuts. Also note that the algorithm does not rely on any particular structure of the tree; it even works for non-binary trees. Although we do not currently make use of this property, it could be useful in the future for increasing compression efficiency.

Algorithm for computing cuts. For computing visibility cuts, we follow the algorithm presented in [Akerlund et al. 2007]. We first sample the visibility function V at the leaf nodes of the tree, then aggregate the values to interior nodes. We compute the cluster mean $\langle v_k \rangle$ for each interior node, as well as its variance $\mathrm{var}(v_k)$, which represents the average L^2 error resulting from the approximation using the cluster mean. To select a cut, we start with the root node, followed by recursive subdivision. At each subdivision step, we find a node with the highest error on the current cut, and replaced it with its two children. This refinement stops when the L^2 error of each node on the cut falls below some threshold:

$$|\mathcal{C}_k| \,\mathrm{var}(v_k) \leq \sigma^2, \quad \forall k \tag{5}$$

where σ is a predefined threshold. We typically set this to 1.5$\sim$3.0. The resulting cut is then stored as a sparse vector containing each node's index and the cluster mean $\langle v_k \rangle$.

Instead of bounding the per-cluster error, we could also bound the total L^2 error of the entire cut, analogous to how wavelet approximation is computed. However, we found that bounding the total error often results in non-uniform distribution of errors, and makes it difficult to conduct theoretical error analysis.

For computing illumination cuts, we prioritize the refinement step differently, by choosing at each step the node with the highest intensity value instead of the highest error. We found this works better for high-frequency lighting as minimizing the error in this case has lower priority than accounting for important, bright lights. In order to satisfy an error bound we apply these strategies in sequence: first generate the cut satisfying the per-cluster error bound, then use any remaining budgeted nodes to refine the cut based on intensity.

For computing the BRDF cuts, we could use similar algorithms. However, one practical issue is that the requirement for BRDF cuts generally prevents users from dynamically changing materials at run-time, a restriction that we would like to eliminate. Therefore, in practice we never precompute BRDF cuts, instead, we evaluate BRDFs on the fly using each cluster's representative direction $\langle \mathbf{x}_k \rangle$. This effectively changes the computation in Eq. 4 to:

$$B = \textstyle\sum_k |\mathcal{C}'_k| \,\langle l_k \rangle \,\langle v_k \rangle \, f_r(\langle \mathbf{x}_k \rangle) \tag{6}$$

As noted by [Akerlund et al. 2007], this approach is feasible as long as the angular size (solid angle) of each cluster is constrained, which can be easily bounded in precomputation. More precisely, it is accurate for BRDFs up to a certain frequency limit defined by the solid angle bound. We will return to this issue in Section 4. For now let's assume that a BRDF cut does exist, as the following theoretical analysis remain valid for the most general case.

3.3 Theoretical Analysis

Complexity analysis. The computational cost of our algorithm is linear to the size of the merged cut. Assuming we have N cuts and their average size is m: in the best case when all cuts share exactly the same nodes, the complexity is $O(m)$; and in the worst case when cuts are extremely incoherent, the complexity is $O(Nm)$. Typically the complexity falls in between and on the low side.

Due to the adaptive approximation nature of cuts, m is usually substantially smaller than $|\mathcal{S}|$: the total number of point lights. In fact, it is typically less than 1% of $|\mathcal{S}|$ in all our experiments. The overall complexity using our algorithm is thus strongly sublinear to the total number of lights.

Error analysis. Theoretical error bounds using our algorithm can be conveniently analyzed through statistical interpretations. In the following we focus on the analysis of multi-product integrals of cuts; interpolation of cuts can be studied similarly. For product integrals, it suffices to examine per-cluster error, since the error for all clusters are bounded in the same way.

Double product integrals

Assume that functions a and b are approximated with cuts; for any given cluster $\mathcal{C}_k$ on the merged cut, the following conditions are true by the cut selection criterion (Eq. 5):

$$|\mathcal{C}_k| \,\mathrm{var}(a_k) \leq \sigma_a^2, \quad |\mathcal{C}_k| \,\mathrm{var}(b_k) \leq \sigma_b^2 \tag{7}$$

where σ_a, σ_b are the predefined error thresholds for a and b. Now if we approximate their double product integral $\sum_k a_k b_k$ with the simple product using the mean values $|\mathcal{C}_k| \langle a_k \rangle \langle b_k \rangle$, the absolute error resulting from this approximation can be bounded by:

$$\begin{aligned} \varepsilon_2 &= |\textstyle\sum a_k b_k - |\mathcal{C}_k| \langle a_k \rangle \langle b_k \rangle| \\ &= |\mathcal{C}_k| \cdot |\langle a_k b_k \rangle - \langle a_k \rangle \langle b_k \rangle| \\ &= |\mathcal{C}_k| \cdot |\mathrm{cov}(a_k, b_k)| \\ &\leq \sqrt{|\mathcal{C}_k| \,\mathrm{var}(a_k) \cdot |\mathcal{C}_k| \,\mathrm{var}(b_k)} \leq \sigma_a \sigma_b \end{aligned} \tag{8}$$

where $\langle\cdot\rangle$ denotes and mean, and cov denotes the *covariance*. Step (8) uses the Cauchy-Schwarz inequality and the conditions given in Eq. 7. This implies that the double product integral error only depends on the error assumed at each individual cut. Therefore, once σ_a and σ_b are decided, we can easily predict the error in rendering.

Triple product integrals

We found that the conclusion above does not extend trivially to the case of triple- or multi-product integrals. In the triple case, the error is similarly defined, with an additional function c:

$$\begin{aligned} \varepsilon_3 &= |\sum a_k b_k c_k - |\mathcal{C}_k|\langle a_k\rangle\langle b_k\rangle\langle c_k\rangle| \\ &= |\mathcal{C}_k| \cdot |\langle a_k b_k c_k\rangle - \langle a_k\rangle\langle b_k\rangle\langle c_k\rangle| \end{aligned} \quad (9)$$

There is unfortunately no simple formula to relate this metric directly to our pre-conditions; however, using a close approximation by Bohrnstedt et al. [1969], we can reduce the error to:

$$\varepsilon_3 \approx |\mathcal{C}_k| \cdot |\langle a_k\rangle \mathrm{cov}(b_k, c_k) + \langle b_k\rangle \mathrm{cov}(a_k, c_k) + \langle c_k\rangle \mathrm{cov}(a_k, b_k)|$$

Now, similar to Eq. 8, the three covariances are bounded by:

$$|\mathrm{cov}(a_k, b_k)| \leq \sigma_a\sigma_b; |\mathrm{cov}(a_k, c_k)| \leq \sigma_a\sigma_c; |\mathrm{cov}(b_k, c_k)| \leq \sigma_b\sigma_c$$

However, unlike the double product integral case, the total error ε_3 now additionally relies on the mean values $\langle a_k\rangle$, $\langle b_k\rangle$, and $\langle c_k\rangle$, which may not be bounded in the general case. The fact that ε_3 may be unbounded is not surprising: since we are computing the product of three functions, a large value in any of the three can arbitrarily magnify the product error of the other two terms.

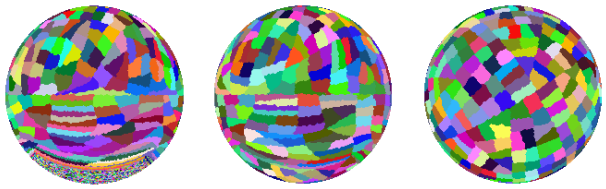
Although it seems that a meaningful analysis would be impossible given the situation, in reality the quantities involved in physics often come with additional constraints that help eliminate the anomalies. For instance, look at the three quantities involved in Eq. 2: first of all, visibility V is a binary function and thus has a trivial bound:$|\langle v_k\rangle| \leq 1$; second, a physically-based BRDF must conserve energy, therefore even though some of its particular values could be unbounded, its integral across the entire domain must be bounded (recall that we associate the cosine term with the BRDF):

$$\textstyle\sum_k |\mathcal{C}_k| \cdot |\langle \rho_k\rangle| = \int_{\mathcal{S}} f_r \leq 1$$

Therefore, overall the BRDF factor contributes to limited error when it's summed across all clusters. Finally, although we are unable to bound the illumination L in the same way, studies show that people are perceptually more tolerant to absolute errors as the overall illumination level increases.

In practice we can easily control the light and visibility cut node errors during cut computation. The average visibility error is trivially controlled as described above and a reasonable maximum lighting value can be selected to find appropriate maximum cut node errors, for example by taking the maximum of the light probes we render with. However, since our system never produces an explicit representation of the BRDF cut we cannot control the node error or average value. Therefore our system relies on the light and visibility cuts to provide a sufficient sampling of the BRDF. We believe that combined with a maximum solid angle constraint a constraint on the average value of each node could be provided for certain classes of BRDFs. We leave this investigation to future work. For practical reasons we allow the user to control the maximum cut size of all cuts in order to provide a better user experience and leave it up to the user to increase the maximum cut size when more accurate rendering is desired.

Comparison to Haar wavelets. Haar wavelets have been studied extensively in previous relighting systems [Ng et al. 2003; Liu et al. 2004; Wang et al. 2004] and are generally believed to be the most efficient representation among other choices. Fundamentally



(a) Purely Data-Driven (b) Mixture (c) Purely Geometric

Figure 5: *Clustering of point lights for the light tree construction. These graphs show environment lighting samples clustered at the 9th level, and compare the three distance metrics.*

Haar wavelets are just like cuts: both are adaptive methods and create piecewise constant approximations. Therefore in theory they should have similar efficiencies. This is confirmed through our experiments in Section 5. Despite the similarity, cuts are much more flexible and easy to analyze, and have a simple and efficient algorithm for computing sums and multi-product integrals. This results in a significant performance speedup over similar computation using wavelets. In addition, we've shown that the overall computation error using cuts can be easily analyzed, making it possible to predict rendering accuracy in a mathematically sound way. We are not aware of any similar analysis on the wavelet side [Ng et al. 2004; Sun and Mukherjee 2006].

3.4 Improving the Light Tree Construction

Our initial algorithm for constructing the binary light tree follows the algorithm by Hašan et al. [2006]. A purely geometric distance metric d_g from a point $\mathbf{x}_s$ with normal $\mathbf{n}_s$ to its cluster center $\langle\mathbf{x}_c\rangle$ with cluster average normal $\langle\mathbf{n}_c\rangle$ is defined as

$$d_g(s,c) = \frac{K^2}{d^2}||\mathbf{x}_s - \langle\mathbf{x}_c\rangle||^2 + ||\mathbf{n}_s - \langle\mathbf{n}_c\rangle||^2 \quad (10)$$

where d is the length of the scene bounding box diagonal, intended for normalization, and K is a user defined parameter for which we find a default value of 30, as used by Hašan et al. [2006], usually gives good results. Because clusters must be evenly sized to create a complete binary tree, clusters are formed by sorting the samples by the difference in the distances to the current two cluster centers, $d_g(s,c_1) - d_g(s,c_2)$ and splitting the array in the middle. The tree is built top down, recursively applying this clustering algorithm to generate a complete binary tree.

The basic assumption behind this method is that points that are geometrically close to each other are likely to have similar illumination values. While this is a good heuristic, we found that by having some knowledge of the functions being approximated, we can optimize the light tree construction and achieve a significant improvement in compression. This is accomplished using a *data-driven approach* that exploits pilot visibility samples.

Specifically, for each light sample we generate a *light visibility vector* $\vec{l_i}$ which is a subsampled version of the full visibility vector from a single light $\mathbf{x}_i$ to all mesh vertices $\mathbf{x}_o$. In order to produce $\vec{l_i}$, we must solve a similar problem and cluster the mesh vertices into M clusters, where M is the size of the light visibility vector. We use the purely geometric approach described above to cluster mesh vertices very efficiently. We then sample the light visibility vectors, by casting shadow rays from each light $\mathbf{x}_i$ to each vertex $\mathbf{x}_o$, and averaging the visibility values taken for all members belonging to every vertex cluster. With this information we can define a new, data-driven distance function d_v:

$$d_v(s,c) = \frac{(K^2+1)A^2}{M}||\vec{l_s} - \langle\vec{l_c}\rangle||^2 \quad (11)$$

where both $\vec{l}_s$ and $\vec{l}_c$ are M-D vectors, the former being the light sample's visibility vector, and the latter being the average visibility vector of a cluster of lights. K is the same as in Eq. 10. A is the approximate solid angle subtended by each light. We use $4\pi/N$ as a heuristic for A, where N is the total number of lights. The constant scaling terms are intended to normalize this metric to d_g for use later and do not affect the results of this approach.

While clustering using this metric works well, we find two problems with it. First, zero visibility vectors are clustered randomly, as seen in Figure 5(a) for sampling directions near the $-Y$ axis. This creates difficulties in producing a stable illumination cut. Second, the metric is biased strongly toward approximating visibility well, and hence may create highly anisotropic clusters that are inefficient at representing the run-time illumination or BRDF. This effect can also be seen in Figure 5(a) as the long skinny clusters around the $-Y$ direction. As a result, this purely data driven approach may lead to very long illumination cuts, reducing the overall rendering performance. Therefore, we combine this approach with the purely geometric approach, and create a new distance metric:

$$d(s,c) = (1-\beta)\, d_v(s,c) + \beta\, d_g(s,c) \tag{12}$$

where β is a user defined weight controlling the relative importance of the geometric vs. data-driven metric. Because we have approximately normalized the two metrics, values of 0.4-0.5 for β work well to balance improved compression and light cut efficiency. The results are shown in Figure 5(b). For quantitative results regarding data-driven clustering, refer to Section 5.

Clearly the trade-off of this approach is increased precomputation time vs. improved compression efficiency. Our data-driven approach now requires two visibility sampling passes and therefore takes approximately twice as much time; on the other hand, we found that it typically gains 20$\sim$30% in compression rate and therefore leads to faster rendering framerates due to reduced cut size.

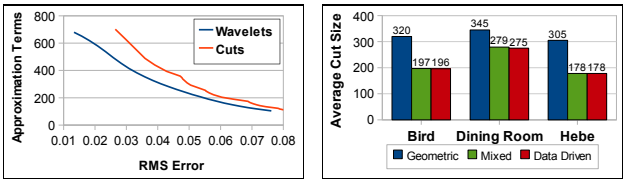
4 Implementation Details

4.1 Precomputation

The first step in precomputation is to generate point lights to sample illumination. For environment lighting we generate samples on the unit sphere; for indirect lighting we divide the samples among the objects by surface area. In both cases we choose random points and use a repulsion algorithm to distribute the points evenly. For environment lighting, we find that 32K points is sufficient for even high frequency lighting. For indirect lighting, the number of samples depends on the scene complexity, but we find 32K works well for all our examples.

Next we must generate the global light tree from the unstructured point lights. A simple implementation can use the geometric distance metric described in Section 3.4. When using the data-driven clustering based on pilot visibility samples, we must choose a light visibility vector length. We have found that 1024 vertex clusters (i.e. the light visibility vector length is 1024) work well, although this can be reduced to save space and still be effective. In our GPU implementation the light tree is represented in a texture, stored by postorder index for simple indexing in the shader.

Finally, we simply loop through each vertex to sample visibility and generate visibility cuts using the algorithm described in Section 3.2. We use a simple, unoptimized ray tracer to sample visibility, but since the shadow rays are coherent, we expect a modern packet ray tracer could perform at least an order of magnitude faster. Because each vertex is independent we parallelize this process by processing different vertices on different CPU cores. For the next step of GPU



(a) (b)

Figure 6: *(a) Visibility compression rates using wavelets vs. cuts. (b) Comparing the efficiency of different clustering schemes for light tree construction.*

rendering, we pack the precomputed data into a 3D texture so that each vector fits into a single row of the texture. For each element in the vector we store the postorder index, leftleaf, and a quantized value into the RGB components of the texture. We use an unsigned 16-bit integer texture available in OpenGL Shader Model 4.0.

4.2 Rendering

Basic rendering algorithm. Since visibility cuts are computed at vertex level, at run-time we need to linearly interpolate these cuts at each pixel in order to produce per-pixel shading results. Therefore our rendering algorithm computes the following equation, which involves both visibility interpolation and triple product integral:

$$B = \sum_k |\mathcal{C}'_k| \, \langle l_k \rangle \, \langle \rho_k \rangle \, \left(w_1 \, \langle v_k^1 \rangle + w_2 \, \langle v_k^2 \rangle + w_3 \, \langle v_k^3 \rangle \right)$$

where w_j is the barycentric weight. Although we logically perform these two steps in sequence, we actually handle all cut operations in a single traversal. We start by constructing the illumination cut for the current frame. The source radiance L at each point light is generated either by sampling a cubemap for environment lighting or using a GPU shadow mapper for indirect lighting, and the rest of the light tree is generated on the CPU. For flexibility and to ensure performance we provide an option for the user to prioritize maximum cut size over error when generating the illumination cut. This cut is then stored in a 1D texture, formatted similarly to the visibility cuts.

Because the shading pass is expensive, our runtime algorithm uses deferred shading to process only the visible fragments. In the first pass we rasterize the scene to a deep frame buffer which stores the barycentric coordinates, material ID, position, normal, and per-pixel BRDF shading parameters of each fragment. Our implementation allows the BRDF to use up to 8 floating point parameters, which is sufficient for our implementation and could easily be extended on current hardware. All our BRDF parameters are modulated by textures to allow for spatially varying BRDFs. We have also included normal perturbation by bump mapping in this step. Other per-pixel shading techniques could be similarly added.

With the illumination cut computed and the shading data available, we now compute the final rendering in a fragment shader. For each pixel, we perform a traversal of the visibility cuts stored at its three defining vertices and the illumination cut at the same time. This traversal follows the cut merging algorithm described in Section 3.2. At each node, we interpolate visibility values using the barycentric weights obtained during rasterization. The value $L(\mathbf{x}_i)$ is obtained directly from the illumination cut. Finally, we need to obtain the value $\langle \rho_k \rangle$. Because creating BRDF cuts would be too expensive and require additional precomputation, we sample the BRDF on the fly. This could be problematic with high frequency BRDFs since none of the cuts guarantee we use small clusters in important and highly varying regions of the BRDF. Currently we rely on the light and visibility cuts to ensure sufficient sampling. If this is still insufficient, we can dynamically send more BRDF samples per cluster to account for high-frequency changes. This is

analogous to bidirectional importance sampling using the illumination and visibility, and works well in practice. Finally, we compute the product, add it to the sum, and display the tonemapped result.

5 Results

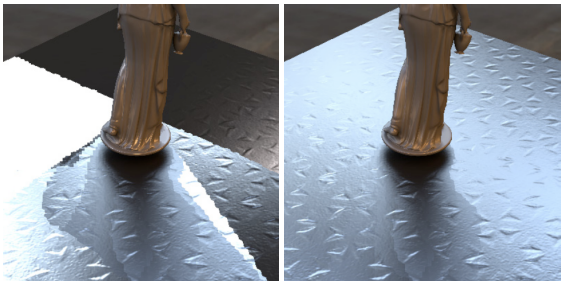
We performed tests on the following scenes: Bedroom (Fig. 1(c)), Dining room (Fig. 8), Hebe (Fig. 8), Motobike (Fig. 1(a)), and Table (Fig. 1(b)). The table scene is courtesy of Ng et al [2004]. All performance statistics are gathered at an image resolution of 640×480 on an Intel Quad-Core 2.0GHz computer with an NVIDIA 8800 GTX graphics card.

Precomputation. Statistics for all the test scenes are given in Figure 10. These data include the compressed file size, average cut size, and the RMS error of the approximation. Note that our storage requirements are reasonable because we precompute only visibility. Data-driven clustering reduces the storage size significantly at the cost of longer precomputation time. We also observe that the visibility cuts computed for indirect lighting are longer than those for environment lighting. This is expected because our test scenes for indirect lighting are more complex than environment lighting, resulting in more complicated occlusions. The majority of the time in precomputation is spent performing visibility sampling, which could be made significantly faster using an optimized ray tracer.

Rendering. Performance statistics are also given in Figure 10. Frame rates are reported for three different illumination cut sizes: 128, 256, and 512. Longer illumination cuts are more accurate for high-frequency lighting and improve final quality, but we found cut sizes of 256∼512 generally work well for all our test scenes with no noticeable artifacts. Our rendering speed is interactive for simple scenes and 2∼4 fps for more complicated ones. Note that indirect lighting tends to be slower on average because the visibility cuts are longer in that case.

Figure 8 shows an example of a scene rendered under environment lighting with a variety of materials. Notice that all effects - hard and soft shadows, low and high glossy materials - are rendered dynamically with high realism. The borrow row in Figures 9 shows the table scene rendered using only direct lighting in the first image and with indirect lighting enabled in the remainder. Figure 8 demonstrates a variety of per-pixel shading effects. Bump mapping provides fine texture, a specular map controls the spatially varying glossiness, and other BRDF parameters are also defined by textures.

Comparison of cuts and wavelets. A direct comparison of cuts and wavelets is difficult for a number of reasons: we generate cuts by limiting the per cluster maximum variance while wavelets are selected by minimizing the total L^2 error of the entire function; further, cuts work on unstructured samples while wavelets require samples to be parameterized. In order to compare the two we only focus on their ability to efficiently represent visibility functions as defined over a cubemap. For wavelets we find the number of coefficients required to approximate the cubemap to a range of maximum L^2 errors. For cuts, we generate visibility cuts as described in Section 3.2 for a range of per node variance thresholds, but then plot according to total L^2 error, averaged over all vertices, for comparison purposes. The resulting graph using a 6x128x128 cubemap is presented in Figure 6(a). Cuts are not quite as efficient as wavelets for this test, requiring about 10∼15% more terms than wavelets for the same error. However, cuts are also at a disadvantage in this comparison since our algorithm optimizes for per-node error, not total cut error. We believe the loss of compression is made up for by the cut representation's simplicity, efficient computation algorithms and flexibility of application. Further, we find that cuts have the nice property that they never inflate sparse high-frequency data as wavelets can.



(a) illumination cut: 1 (b) illumination cut: 32

(c) illumination cut: 64 (d) illumination cut: 256

Figure 7: *Image quality comparison for varying illumination cut sizes. Note the artifacts on the glossy surfaces with cut sizes 1 and 32; as we increase the cut size, the image quality quickly improves.*

Data-driven construction of light tree. Figure 6(b) shows the average cut size using different clustering schemes we tested for the light tree construction. We show average cut size for three values of β: 1.0 for pure geometric clustering; 0.5 for the mixed mode; and 0.0 for pure data-driven clustering. Data-driven clustering improves pure geometric clustering by 20∼30% on average, while the mixed mode only performs slightly worse while improving rendering performance by allowing for more efficient approximation of the lighting.

Varying illumination cut size. Figure 7 shows renderings from our system using various illumination cut sizes. With very short cut sizes glossy surfaces and shadows are shaded inaccurately due to undersampling, but the shading quickly converges to the reference ray traced image. In many cases cut sizes of 256 are sufficient, although for highly glossy surfaces longer cuts are sometimes necessary. For details on how adjusting visibility cut accuracy affects rendering refer to previous work in [Akerlund et al. 2007].

6 Conclusions and Future Work

We have presented a simple and efficient algorithm for performing computations with cuts and prove an error bound on multi-product integrals computed using our algorithm. We apply our algorithm to the problem of realistic lighting and material design under complex illumination with arbitrary spatially varying BRDFs. We demonstrated the efficiency of our algorithm by interpolating visibility and computing the lighting integral at each pixel, enabling dynamic per-pixel effects such as bump mapping and spatially varying BRDFs. In addition, we suggested a two-pass data-driven clustering algorithm which improves compression by 20∼30%.

We chose to sample BRDFs on the fly for greater flexibility, enabling all BRDF parameters to be selected on a per-pixel basis. However, this imposes a few limitations. First, without a BRDF cut we cannot ensure that the BRDF is sampled finely enough in high frequency regions such as highly specular lobes. Because we construct our light cut based on light intensity, we sample our BRDF approximately as we would using importance sampling in a ray-

Figure 8: *The top row shows realistic material design on the model Hebe with changing environment lighting. Note the rich per-pixel shading effects. We use a combination of bump map and spatially-varying BRDF parameters. On the bottom row, the first two images shows editing of the puff chair model with different materials; the right two images show the Ward anisotropic BRDF applied on the teapot and a specular map applied on the sphere.*

Figure 9: *The first two images on the top row compare direct illumination only and the addition of indirect illumination. The third and fourth images show realistic material design of this bedroom model. The bottom row shows the table scene (scene geometry courtesy of [Ng et al. 2004]) rendered with dynamic lighting and per-pixel varying materials.*

				Geometric Based Clustering			Data-driven Clustering				
Scene	Verts	Type	P. Time	Storage	avg. cutsize	RMS	Storage	avg. cutsize	RMS	Gain	FPS
Hebe	68K	Env.	13min	59 MB	260	0.074	45 MB	200	0.057	24%	8 / 6.4 / 5
Motobike	112K	Env.	40min	93 MB	190	0.059	65 MB	133	0.051	30%	4 / 3.5 / 2.5
Table	114K	Env.	30min	183 MB	402	0.099	153 MB	337	0.095	16%	2.5 / 2 / 1.5
Dining	65K	Ind.	20min	-	-	-	114 MB	643	0.13	-	1.4 / 1.2 / 1.1
Teapot	86K	Ind.	16min	-	-	-	78 MB	261	0.058	-	7 / 6 / 4
Bedroom	105K	Ind.	30min	-	-	-	219 MB	662	0.13	-	2 / 2 / 1.5

Figure 10: *Detailed statistics of our algorithm. From left to right, the columns present the vertex count of each model, illumination type (environment or indirect), precomputation time (using geometric clustering for the first three rows and data-driven clustering for the remaining three), and precomputation profiles for the simple geometric based light tree construction and our proposed data-driven light tree construction. The gain for scenes which have both geometric and data-driven statistics refers to the improvement in final compressed data size of the data-driven approach over the purely geometric clustering. The data-driven approach requires twice the precomputation time, but reduces overall data size by an average of 20∼30%. Note that due to increased scene complexity, the cut sizes for the indirect lighting examples are typically larger than environment lighting. The last column presents the rendering rates using three different illumination cut sizes: 128/256/512.*

tracer. This seems to be sufficient for our test scenes, but we are interested in solutions to this problem. We believe that for certain classes of BRDFs constraints on frequency and average value within a cluster could be found either analytically or based on a heuristic such as the gradient at the cluster center. We leave investigation of these approaches to future work.

We also note that our algorithm can be extended to account for rigid-body dynamic scenes using shadow fields [Zhou et al. 2005]. In this case, precomputed visibility fields are created as the effect of an object on the visibility of a neighboring object. At runtime, the full visibility for a pixel is computed by multiplying together the per-object shadow functions from its nearby objects, which can be interpolated from the precomputed shadow field cuts. Again, the entire process can be performed in a single traversal. We are currently implementing this feature in our system.

An implicit assumption we made is that it is appropriate to linearly interpolate visibility data. This may be true if meshes are densely tessellated, but tessellation may inflate data size unnecessarily. Therefore we are interested in studying both nonlinear interpolation schemes and performing efficient spatial compression of visibility data. Finally, we believe that cuts are a general and effective approximation and, combined with our efficient computational algorithm, could find use beyond PRT.

Acknowledgements We would like to thank SIGGRAPH reviewers for their insightful comments. This work was supported in part by NSF grant CCF-0746577 and the UMass FRG grant P1-FRG-0029. We also thank NVIDIA for their generous donations.

References

AGARWAL, S., RAMAMOORTHI, R., BELONGIE, S., AND JENSEN, H. W. 2003. Structured importance sampling of environment maps. *ACM Trans. Graph. 22*, 3, 605–612.

AKERLUND, O., UNGER, M., AND WANG, R. 2007. Precomputed visibility cuts for interactive relighting with dynamic brdfs. *Pacific Graphics 0*, 161–170.

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

ARBREE, A., WALTER, B., AND BALA, K. 2008. Single-pass scalable subsurface rendering with lightcuts. *Computer Graphics Forum 27*, 2, 507–516.

BEN-ARTZI, A., OVERBECK, R., AND RAMAMOORTHI, R. 2006. Real-time brdf editing in complex lighting. *ACM Trans. Graph. 25*, 3, 945–954.

BEN-ARTZI, A., EGAN, K., RAMAMOORTHI, R., AND DURAND, F. 2008. A precomputed polynomial representation for interactive brdf editing with global illumination. *ACM Trans. Graph. 27*, 2, 1–13.

BOHRNSTEDT, G. W., AND GOLDBERGER, A. S. 1969. On the exact covariance of products of random variables. *Journal of American Statistical Association 64*, 1439–42.

CABRAL, B., OLANO, M., AND NEMEC, P. 1999. Reflection space image based rendering. In *Proceedings of SIGGRAPH '99*, ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, 165–170.

CLARBERG, P., JAROSZ, W., AKENINE-MÖLLER, T., AND JENSEN, H. W. 2005. Wavelet importance sampling: efficiently evaluating products of complex functions. *ACM Trans. Graph. 24*, 3, 1166–1175.

COLBERT, M., PATTANAIK, S., AND KRIVNEK, J. 2006. Brdf-shop: Creating physically correct bidirectional reflectance distribution functions. *IEEE Computer Graphics and Applications 26*, 1, 30–36.

DACHSBACHER, C., STAMMINGER, M., DRETTAKIS, G., AND DURAND, F. 2007. Implicit visibility and antiradiance for interactive global illumination. *ACM Trans. Graph. 26*, 3, 61.

DEBEVEC, P. 2005. A median cut algorithm for light probe sampling. In *ACM SIGGRAPH 2005 Posters*, ACM, New York, NY, USA, 66.

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

HAŠAN, M., PELLACINI, F., AND BALA, K. 2006. Direct-to-indirect transfer for cinematic relighting. *ACM Trans. Graph. 25*, 3, 1089–1097.

HAŠAN, M., PELLACINI, F., AND BALA, K. 2007. Matrix row-column sampling for the many-light problem. *ACM Trans. Graph. 26*, 3, 26.

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

KAUTZ, J., BOULOS, S., AND DURAND, F. 2007. Interactive editing and modeling of bidirectional texture functions. *ACM Trans. Graph. 26*, 3, 53.

LAINE, S., SARANSAARI, H., KONTKANEN, J., LEHTINEN, J., AND AILA, T. 2007. Incremental instant radiosity for real-time indirect illumination. In *Proceedings of Eurographics Symposium on Rendering 2007*, Eurographics Association, 277–286.

LAWRENCE, J., RUSINKIEWICZ, S., AND RAMAMOORTHI, R. 2004. Efficient brdf importance sampling using a factored representation. *ACM Trans. Graph. 23*, 3, 496–505.

LEHTINEN, J., AND KAUTZ, J. 2003. Matrix radiance transfer. In *ACM Symposium on Interactive 3D graphics*, 59–64.

LEHTINEN, J. 2007. A framework for precomputed and captured light transport. *ACM Trans. Graph. 26*, 4, 13.

LIU, X., SLOAN, P.-P., SHUM, H.-Y., AND SNYDER, J. 2004. All-frequency precomputed radiance transfer for glossy objects. In *Proceedings of the 15th Eurographics Symposium on Rendering*, 337–344.

NG, R., RAMAMOORTHI, R., AND HANRAHAN, P. 2003. All-frequency shadows using non-linear wavelet lighting approximation. *ACM Trans. Graph. 22*, 3, 376–381.

NG, R., RAMAMOORTHI, R., AND HANRAHAN, P. 2004. Triple product wavelet integrals for all-frequency relighting. *ACM Trans. Graph. 23*, 3, 477–487.

NIJASURE, M., PATTANAIK, S. N., AND GOEL, V. 2005. Real-time global illumination on gpus. *journal of graphics tools 10*, 2, 55–71.

RAMAMOORTHI, R., AND HANRAHAN, P. 2002. Frequency space environment map rendering. *ACM Trans. Graph. 21*, 3, 517–526.

RITSCHEL, T., GROSCH, T., KAUTZ, J., AND MÜLLER, S. 2007. Interactive illumination with coherent shadow maps. In *Proceedings of Eurographics Symposium on Rendering*, 61–72.

SHIRLEY, P., WANG, C., AND ZIMMERMAN, K. 1996. Monte carlo techniques for direct lighting calculations. *ACM Trans. Graph. 15*, 1, 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 Trans. Graph. 21*, 3, 527–536.

SLOAN, P.-P., LIU, X., SHUM, H.-Y., AND SNYDER, J. 2003. Bi-scale radiance transfer. *ACM Trans. Graph. 22*, 3, 370–375.

SUN, W., AND MUKHERJEE, A. 2006. Generalized wavelet product integral for rendering dynamic glossy objects. *ACM Trans. Graph. 25*, 3, 955–966.

SUN, X., ZHOU, K., CHEN, Y., LIN, S., SHI, J., AND GUO, B. 2007. Interactive relighting with dynamic brdfs. *ACM Trans. Graph. 26*, 3, 27.

WALTER, B., FERNANDEZ, S., ARBREE, A., BALA, K., DONIKIAN, M., AND GREENBERG, D. P. 2005. Lightcuts: a scalable approach to illumination. *ACM Trans. Graph. 24*, 3, 1098–1107.

WALTER, B., ARBREE, A., BALA, K., AND GREENBERG, D. P. 2006. Multidimensional lightcuts. *ACM Trans. Graph. 25*, 3, 1081–1088.

WANG, R., TRAN, J., AND LUEBKE, D. 2004. All-frequency relighting of non-diffuse objects using separable BRDF approximation. In *Proceedings of the 15th Eurographics Symposium on Rendering*, 345–354.

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

ZHOU, K., HU, Y., LIN, S., GUO, B., AND SHUM, H.-Y. 2005. Precomputed shadow fields for dynamic scenes. *ACM Trans. Graph. 24*, 3, 1196–1201.