Fast, Realistic Lighting and Material Design using Nonlinear Cut Approximation

(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 interactive rendering system for realistic lighting and material design under complex illumination with arbitrary BRDFs. Our system smoothly integrates accurate 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 build upon a clustered piecewise constant representation called *cuts* to accurately approximate both the illumination and precomputed visibility as nonlinear sparse vectors. Our key contribution is an efficient algorithm for merging multiple cuts in a single linear traversal. We use this algorithm to simultaneously interpolate visibility cuts at each pixel, and to compute the triple product integral of the illumination, interpolated visibility, and dynamic BRDF samples. The theoretical error bound of this approach can be proved using statistical interpretations of cuts. Our algorithm extends naturally to computation with many cuts; moreover, it maps easily to modern GPUs, resulting in a significant performance speedup over existing methods based on nonlinear wavelet approximations. 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.

Keywords: Relighting, precomputed light transport, clustering, cuts, BRDF editing, per-pixel shading, global illumination

1 Introduction

Realistic image synthesis requires incorporating scalable illumination, intricate shadowing, and finely detailed materials. This often prevents users from receiving real-time feedback on lighting and material changes due to expensive simulation costs. For instance, final-quality rendering often requires minutes to hours to complete one frame, significantly hampering user productivity.

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 in exchange for real-time frame rates. Early prototypes require fixing material properties to avoid high sampling rates and rendering costs. Subsequent work, such as [Ng et al. 2004], permits dynamically selecting materials, but only supports a small set of BRDFs due to the significant preprocessing cost for each BRDF. Most recent advances in material editing [Ben-Artzi et al. 2006; Sun et al. 2007] have provided techniques for fast feedback upon BRDF changes. However, they make substantial assumptions that require either fixing the lighting and view or representing arbitrary BRDFs with a small set of linear bases. Moreover, dynamic control of meso-scale surface details is generally missing in such systems, due to the lack of per-pixel shading.

We present an interactive system that supports accurate, realistic lighting and material design with dynamic per-pixel shading effects. Figure 1 shows several examples captured in real-time. Our system is built with the following principles in mind:

- Fully dynamic control of lighting, view point, and BRDFs;
- Per-pixel shading details using bump mapping and spatially varying BRDF parameters, all of which can be modified interactively with no precomputation required;
- Accurate soft shadows and reflections at all frequency scales;
- No precomputed BRDF data or BRDF linear basis set;
- 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 and exploited on the fly. 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].

For accurate rendering, we exploit a clustered piecewise constant representation called *cuts* [Walter et al. 2005; Akerlund et al. 2007] to efficiently approximate both the illumination and precomputed visibility functions as nonlinear sparse vectors. To achieve per-pixel shading, we interpolate visibility vectors for each pixel, then evaluate the view-dependent shading results by integrating the lighting, interpolated visibility, and dynamically computed BRDF samples. We use a deep deferred shading buffer to store all relevant shading parameters. This allows us to dynamically control meso-scale surface shading details by enabling bump mapping and spatially varying BRDF parameters.

While cuts have been shown to be accurate and flexible, no prior work has studied efficient algorithms for computation involving many cuts, such as linear interpolation and multi-product integrals. These are at the core of achieving our aforementioned goals. As the primary contribution of this work, we present an efficient GPU-friendly algorithm for computing sums and multi-product integrals of cuts in a single linear traversal. We use this algorithm to interpolate visibility cuts for each pixel in a fragment shader, while simultaneously computing their integrals with the illumination cut and dynamic BRDF samples.

Our algorithm extends trivially to computation with many cuts. We show in Section 3 that the accuracy of such computation is guaranteed if we can bound the approximation error of each individual cut node. Moreover, the algorithm maps easily to modern GPUs with dynamic branching. The combination of algorithmic and hardware developments lead to an order of magnitude speedup over existing methods based on wavelets.

Finally, we present an two-pass, data-driven approach that exploits subsampled visibility data to optimize the illumination clustering schemes. This proves to be effective at improving the efficiency of cuts, and reduces the overall data size by an average of 20$\sim$30%.

2 Related Work

Illumination from many lights. Efficient rendering from a large number of light sources has always been a central challenge in graphics. Debevec et al. [1998] show that using detailed environment lighting enables photorealistic rendering effects; most global illumination problems such as indirect lighting are fundamentally problems involving many lights. Standard algorithms entail a linear cost with increased number of lights. To reduce this cost, several methods[Ward 1994; Shirley et al. 1996; Paquette et al. 1998] have been introduced to intelligently pick important lights and save computation on unimportant ones; [Wald et al. 2003; 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 highly efficient, 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 by [Hašan et al. 2007] exploits the GPU to quickly sample and cluster many lights, improving 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 real-time visibility sampling, but typically at the cost of reduced 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 in exchange for real-time frame rates. A comprehensive overview of current PRT techniques can be found in [Lehtinen 2007]. Early work presented in [Kautz et al. 2002; Lehtinen and Kautz 2003; Liu et al. 2004; Wang et al. 2004] focuses on illumination from distant environment lighting or mid-range illumination [Sloan et al. 2002; Annen et al. 2004]. They take dense illumination samples, then exploit standard bases such as spherical harmonics (SH), Haar wavelets, or radial basis functions (RBFs) to compress light transport data and reduce rendering costs. A drawback of these bases is that they are difficult to apply to arbitrary, unstructured illumination samples, as in the case of local indirect lighting. This often limits the technique to parameterized scene models [Kontkanen et al. 2006]. A recent work by Hašan et al. [2006] introduced a scheme to hierarchically cluster a group of lighting samples into a quad-tree, which can then be compressed directly using wavelets. Most recently, Akerlund et al. [2007] presented precomputed visibility cuts, a technique inspired by Lightcuts, for accurate, piecewise constant approximation of arbitrary lighting samples. This approach has the additional advantage of incorporating BRDF samples computed on the fly, thus permitting fully dynamic material editing.

Our method builds upon this general representation using cuts. We note that clustering of illumination samples is hardly a new idea: it has been used frequently in previous work, notably in radiosity algorithms [Hanrahan et al. 1991; Drettakis and Sillion 1997; Gortler et al. 1993]. However, our main interest is in efficient algorithms for computing interactions between many cuts which are also GPU-friendly. To our knowledge this is the first study on such topic. Two related works are the wavelet triple product integral by Ng et al. [2004] and the generalized version by Sun et al. [2006]. However, wavelets impose significant limitations on handling local lighting and dynamic BRDFs; in addition, the multi-product wavelet integral algorithm is expensive to map to the GPU, and hasn't been shown in fully interactive rates.

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 realistic shadows but requires fixing material properties. By adopting a factored representation that decouples materials from scene definition, Ng et al. [2004] enable dynamic selection of BRDFs, but they only support a small selection of materials as expensive preprocessing is required for each BRDF. In general, these systems focus more on the capability of *relighting* under rich illumination but do not provide online modification of arbitrary BRDFs.

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 *bidirectional texture functions* (BTFs). Most recently, advances in BRDF editing [Ben-Artzi et al. 2006; Ben-Artzi et al. 2007; Sun et al. 2007] have enabled real-time BRDF changes under large-scale illumination; but due to the high cost in sampling view-dependent effects, they typically make substantial assumptions that require fixing the lighting and view, or require representing general BRDFs with a small linear basis set. Both of these are assumptions we try to avoid. In addition, due to the lack of per-pixel shading support, they ignore important meso-scale surface details provided by bump maps, spatially varying BRDFs or bidirectional texture functions. A recent work by [Kontkanen et al. 2007] precomputes visibility data (stored in shadow maps) and casts dynamic BRDF samples to simulate complex reflections. While conceptually similar to ours, their system is unsuitable for large-scale illumination, requiring several seconds to converge.

Sloan et al. [Sloan et al. 2003] introduced bi-scale radiance to incorporate BTFs in low-frequency PRT: a macro-scale that smoothly interpolates global effects (e.g. shadows), and a meso-scale that provides local shading details using texture functions. Their system is limited to low-frequency transport effects, and requires preprocessing of texture functions. Our approach is similar to theirs in spirit, but we handle all-frequency transport effects efficiently throughout our system; in addition, we permit fully dynamic material changes with no preprocessing of BRDF data.

3 Algorithm Overview

Assumptions. Similar to previous work, we make the following assumptions: 1) scene geometry is fixed; 2) illumination can be modeled as many diffuse point lights (this includes distant lights); 3) we only consider one bounce of illumination directly from the lighting samples. The second assumption means we do not handle glossy to glossy transfer paths.

3.1 Mathematical Framework

Given a set of point lights $\mathcal{S}$, the radiance B caused by one-bounce illumination from $\mathcal{S}$ to a surface point $\mathbf{x}_o$ in view direction ω 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 $\mathbf{x}_i \in \mathcal{S}$ is an illumination sample, L is the source radiance, f_r is the BRDF, V is the binary visibility function, and G is a geometric term – the differential solid angle subtended by $\mathbf{x}_i$. To simplify the notation, we combine the cosine term 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:

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

As $\mathcal{S}$ contains a large number of points, directly evaluating this integral is impractical. A large body of previous work has studied the use of 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 limited to distant environment lighting and require precomputing BRDF data. Refer to [Ng et al. 2004] for a comprehensive study of these approaches.

Representation using Cuts. Recently Akerlund et al. [2007] presented an alternative representation called *cuts* for accurate approximation in large-scale illumination problems. Cuts nonlinearly approximate the precomputed visibility function V as a piecewise constant function: assume that we can partition the whole set $\mathcal{S}$ into a small number of clusters $\mathcal{C}_k$, such that V is coherent within each $\mathcal{C}_k$ and can thus be approximated by the *mean* cluster value:

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

where $\langle \cdot \rangle$ denotes the mean, and $|\mathcal{C}_k|$ denotes the number of cluster samples. In addition, the center of cluster (the mean position of samples) $\langle \mathbf{x}_k \rangle = \frac{1}{|\mathcal{C}_k|} \sum \mathbf{x}_i$ can be used as a representative direction to sample dynamic BRDFs, assuming that the BRDF changes smoothly. In order to efficiently compute the cluster values, a global *light tree* is constructed which hierarchically partitions lighting samples. The light tree is a binary tree where the leaf nodes represent individual lighting samples and the interior nodes represent clusters. A *cut* through the tree is selected to represent a partitioning of lighting samples into disjoint sets of clusters. Figure 2 shows an example. Cuts are selected in order to minimize both error and cut size. In [Akerlund et al. 2007], visibility functions are preprocessed using cuts, which are called *precomputed visibility cuts*; the rendering algorithm then takes one sample for L and f_r per visibility cluster, and computes the illumination integral by summing up the contribution from all clusters. This approach creates clusters based entirely on visibility and does not account for potential errors in the illumination and BRDF. Moreover, as in most other PRT systems, they are restricted to per-vertex lighting, disabling important per-pixel shading effects.

In general we can use cuts to approximate arbitrary functions up to some predefined accuracy. For example, we can create an *illumination cut* for L, and similarly, *BRDF cuts* for f_r. The question is how

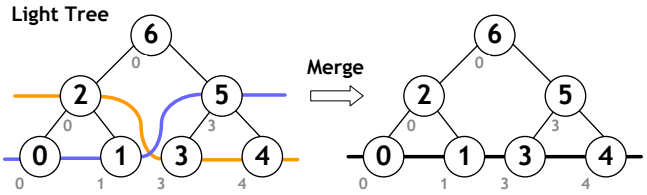


Figure 2: *A light tree and two example cuts. Leaf nodes are individual light samples and interior nodes are clusters. Each tree node is indexed by the postorder traversal index (the center number); it also stores the index to its leftmost child leaf (the lower-left number). A cut (colored lines) represents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

to efficiently combine these cuts and use them to quickly evaluate Eq. 2. As presented in the next section, a simple algorithm exists to quickly evaluate the triple product integral of the three cuts; in addition, the same algorithm can simultaneously interpolate several cuts, making it possible to enable per-pixel lighting by interpolating visibility cuts at each pixel. The result of this computation is effectively a new, *merged cut* that combines the subclusters from all source cuts. With the merged cut, Eq. 2 becomes:

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

where $\mathcal{C}'_k$ denotes a new cluster on the merged cut 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 merged cut retains the accuracy in each source cut, the result of this approximation has guaranteed error bound. In Section 4 we show a theoretical analysis of this error bound.

Unfortunately computing cuts for the BRDF is impractical as it involves significant preprocessing that prevents us from modifying them dynamically. Therefore, in practice we never compute BRDF cuts, but instead evaluate BRDFs dynamically using each cluster's center direction. This effectively changes the computation to:

$$B = \textstyle\sum_k |\mathcal{C}'_k|\, \langle l_k \rangle\, \langle v_k \rangle\, f_r(\langle \mathbf{x}_k \rangle) \quad (5)$$

We will return to this issue in Section 4. For now assume that a BRDF cut does exist, and the following algorithm and theoretical analysis remain valid for the most general case.

3.2 Efficient Algorithm for Merging Cuts

Properties of Cuts. We first define some notations to simplify the discussion. As shown in Figure 2, each tree node on the light tree is indexed by its *postorder traversal index*; at 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 have:

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 non-overlapping cluster on the left side of* $\mathbf{p_2}$ *in the tree.*

In addition, the following properties hold for cuts in general:

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 children nodes in all cuts; in other words, it represents a new clustering that merges the subclusters of all cuts.*

Algorithm for Merging Cuts. Our goal is to efficiently compute sums and product integrals of several cuts. From Figure 3, it is easy

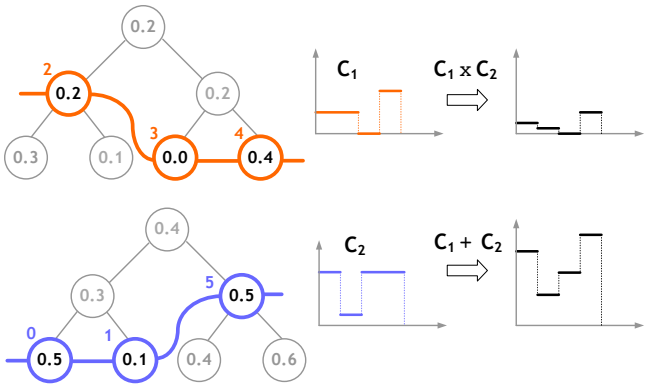


Figure 3: *Merging cuts. The left column shows two cuts C_1, C_2. The node value is the cluster mean (average); the upper-left colored number is the cluster index. The middle column shows the effective cut approximations. The right column shows the results of summing and multiplying the two cuts.*

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 purple cut. Shaded nodes are the current nodes in execution (the $\mathbf{p}_j$'s); underscore marks the smallest current index; checkmarks indicate whether all current nodes overlap.*

to see that both operations result in a new piecewise constant representation of the merged cut. Therefore we use a unified algorithm called *merging cuts* to compute both.

We note from Properties 3 and 4 that merging cuts very much resembles merging sorted arrays or sorted sparse vectors. The standard algorithm works as follow: start by setting pointers to the first elements of the arrays; begin looping, finding the smallest elements (or elements with the smallest index, in case an index-value pair is used) from all current pointers, merge the smallest values into a result buffer, and increment the pointers of those with the smallest elements; the loop continues until it has come to 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 on each array, and the traversal is performed simultaneously for all arrays. However, to use it for merging cuts, we must consider a the case where cut nodes that do not share the same index can still overlap because 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 current index; however, the contribution of *all* current nodes will be merged together as long as the intersection of their clusters is not empty.

Nodes are overlapping if they all share one common child. According to Property 1, this can be quickly checked by intersecting the child index range of all nodes: the intersection is non-empty if and only if all nodes overlap. With this modification, we now have the complete algorithm presented below in pseudocode. The algorithm works for an arbitrary number (N) of cuts, and the computation can be a sum or product of multiple cuts, or a combination of the two:

---

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 the multi-product integral of the 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 linear interpolation weight. In Figure 3 and 4 we show a detailed example of running this algorithm.

We note that the algorithm does not rely on any particular tree structure. It works even for non-binary trees. Although we do not currently take advantage of this property it could be useful in the future for increasing compression efficiency.

Algorithm for selecting a cut. Our cut selection algorithm is similar to [Akerlund et al. 2007]. We first sample a function (e.g. the visibility V) at the leaf nodes of the light tree; these values are then clustered and propagated to all the interior nodes. At each node we store the cluster mean $\langle v_k \rangle$, along with the cluster variance $\mathrm{var}(v_k)$, which represents the *average* L^2 approximation error caused by substituting the entire cluster with the mean. To select a cut, we start with a trivial cut that contains only the root node, then progressively subdivide it. At each subdivision step we choose the node from the current cut with the highest error and replace it with its two children nodes. This refinement process stops when every cut node has a total error that falls below a certain threshold:

$$|\mathcal{C}_k| \, \mathrm{var}(v_k) \leq \sigma^2, \quad \forall k \tag{6}$$

where σ is a predefined threshold representing the maximum standard deviation allowed for this approximation. We typically set this to 1.5∼3.0 in all our experiments. The resulting cut is then stored as a sparse vector containing each cut node's index and the cluster mean $\langle v_k \rangle$. To prevent cuts from growing to arbitrary lengths, we set a maximum cut size of 1000 as in [Walter et al. 2005].

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

3.3 Theoretical Analysis

Complexity analysis. Clearly the computational cost of our algorithm is linear to the size of the final merged cut. Assuming we have N cuts and the average cut size is m: in the best case when all cuts are exactly the same (essentially degenerating to linear approximation), the complexity is $O(m)$; and in the worst case when cuts are extremely incoherent, the complexity is $O(Nm)$. In typical situations the complexity falls somewhere in between.

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

Error analysis. Theoretical error bounds using our approach can be easily proved 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 the error at one cluster node, as the total error is a direct sum of per-cluster errors.

Double product integrals

Assume that functions a and b have been approximated with cuts; at any given cluster $\mathcal{C}_k$ on their merged cut, our cut selection criterion (Eq. 6) guarantees the following conditions:

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

where σ_a, σ_b are the individual cut 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 their means $|\mathcal{C}_k|\langle a_k\rangle\langle b_k\rangle$, the absolute error resulting from this approximation can be bounded by:

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

where $\langle\cdot\rangle$ denotes and mean, and cov denotes the *covariance*. Step (8) can be proved using the Cauchy-Schwarz inequality and the conditions given in Eq. 7. This implies that the double product integral error only relies on the error assumed for each individual cut. Therefore we can effectively predict the error of this computation once we have limited the cut selection threshold σ.

Triple product integrals

Unfortunately the conclusion above does not extend trivially to the cases of triple- and multi-product integrals. The error resulting from the approximation is similar, with an additional function c:

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

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

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

Similar to Eq. 8, the covariances are still bounded:

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

however, the error now additionally relies on the magnitude of the mean values $|\langle a_k\rangle|$, $|\langle b_k\rangle|$, and $|\langle c_k\rangle|$, which generally cannot be assumed to have bounds.

This problem can be understood intuitively by the fact that a large average value in any of the three functions can arbitrarily magnify the product error resulting from the other two terms. The simplest example is to set one function, say a_k, to a constant, thus $a_k = \bar{a}$. In this case, Eq. 9 reduces to the simple form:

$$\varepsilon_3 = |\mathcal{C}_k| \cdot |\bar{a}| \cdot |\operatorname{cov}(b_k, c_k)| \leq |\mathcal{C}_k| \cdot |\bar{a}| \cdot \sigma_b\sigma_c \tag{10}$$

where it's obvious that the error is unbounded if $\bar{a}$ becomes arbitrarily large. Note that double product integrals do not have this problem.

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 remove the anomalies. For instance, if we look at the three quantities involved in Eq. 2: first of all, visibility V is a binary function and is thus bounded:

$$|\langle v_k\rangle| \leq 1, \quad \forall v_k$$

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):

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

So the BRDF term contributes to limited error when 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 errors as the overall illumination level increases.

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]. Fundamentally Haar wavelets are just like cuts: both are adaptive methods that create piecewise constant representations. Therefore in theory they should have similar efficiencies. This claim is confirmed through our experiments in Section 5. Despite the similarity, cuts are more flexible and enable a substantially improved and efficient algorithm for computing sums and multi-product integrals on the GPU, resulting in a significant performance speedup over similar computation using wavelets. In addition, we've shown that the computation error using cuts can be easily bound using statistical interpretations; we are not aware of any similar analysis on wavelet-based methods.

3.4 Improving the Light Tree Construction

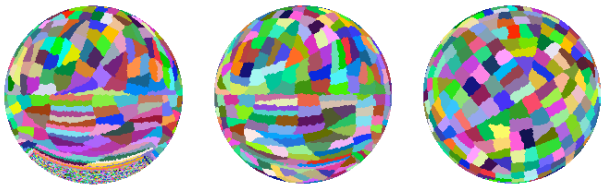
Our initial light tree construction algorithm is based on the clustering algorithm by Hašan et al. [2006]. A purely geometric distance metric d_g from a point $\mathbf{x}_s$ to cluster center $\langle\mathbf{x}_c\rangle$ is defined as

$$d_g(s, c) = \frac{K^2}{d^2}||\mathbf{x}_s - \langle\mathbf{x}_c\rangle||^2 + ||\mathbf{n}_s - \langle\mathbf{n}_c\rangle||^2 \tag{11}$$

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 always 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 of 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 illumination samples that are geometrically close to each other are likely to produce similar light transport results. While this is a good heuristic, we found that by having some knowledge of the actual functions being approximated, we can optimize the light tree construction and achieve a significant improvement in compression. This is done by 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 visibility vector from a single light sample point $\mathbf{x}_i$ to all mesh vertices $\mathbf{x}_o$. In order to subsample the visibility vector, 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 accomplish this very efficiently. We then sample the light visibility vectors, by casting shadow rays from each light $\mathbf{x}_i$ to each vertex cluster, and averaging the visibility values taken for all cluster members. With this information we can define a new, data-driven distance function d_v:



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

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

$$d_v(s,c) = \frac{(K+1)^2 A^2}{M} ||\vec{l_s} - \langle \vec{l_c} \rangle||^2 \quad (12)$$

where $\vec{l_s}$ is the light sample's visibility vector and $\vec{l_c}$ is the average visibility vector of a light cluster, K is the same as in the previous scheme, and A is the approximate solid angle subtended by each light, i.e. $4\pi/N$ where N is the 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, in some cases the visibility vectors are all zero and clustering becomes ineffective – samples with zero visibility vectors are grouped at random. The effect can be seen in Figure 5(a) for sampling directions near the $-Y$ axis. Due to a ground plane, many samples near the $-Y$ direction have zero light visibility vectors and so get clustered randomly. 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. This effect can also be seen in Figure 5(a) as the long skinny clusters in the $-Y$ direction. While this purely data driven approach significantly improves the compression rate of precomputed visibility cuts, it may lead to very long illumination cuts, thus reducing overall rendering performance. Therefore, we combine this approach with the geometric approach and create a new distance metric:

$$d(s,c) = (1-\beta)\, d_v(s,c) + \beta\, d_g(s,c) \quad (13)$$

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∼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 light sample points. For environment lighting we generate samples on the unit sphere; and for local 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. We find that 32K points is sufficient for even high frequency environment lighting. For local lighting the number of samples depends on the complexity of the scene, but we find 32K works well for all our examples.

Next we must generate the global light tree from the unorganized sample points. 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.

Once we have the light tree constructed, we simply loop through each vertex to generate visibility cuts. To sample visibility, we use a simple, unoptimized ray tracer to cast shadow rays from every vertex to each light sample. Because these rays are cast in a coherent fashion, we expect a modern, optimized ray tracer could perform at least an order of magnitude faster. We then multiply the binary visibility with the proper geometric term of the light sample, which depends on the type of lighting being used. Because our implementation guarantees the tree will be a complete binary tree we store the tree linearly in memory for efficiency. Once the sampled visibility values have been propagated up the tree, including the variance, we use the algorithm described in Section 3.2 to compute a cut. Because each vertex is independent we parallelize this process by processing different vertices in different threads.

4.2 Rendering

Basic rendering algorithm. At runtime we need to render our mesh, compute a merged visibility cut for each pixel as a linear combination of the visibility cuts at the triangle's vertices, and compute the approximate value of the lighting integral by multiplying the merged visibility cut, the light cut, and the BRDF cut:

$$B = \sum L(\mathbf{x}_i)\, f_r(\mathbf{x}_i)\ (w_1\, V_1(\mathbf{x}_i) + w_2\, V_2(\mathbf{x}_i) + w_3\, V_3(\mathbf{x}_i))$$

Although we logically perform these steps in sequence, generating intermediate merged cuts, we actually handle all cut operations in a single traversal. We construct the light cut for the current frame. Then, for each pixel, we perform a traversal on the vertex visibility cuts and the light cut at the same time. This traversal visits the deepest cluster nodes in all of these cuts. At each node we interpolate the current visibility values using barycentric weights obtained during rasterization. The value $L(\mathbf{x}_i)$ is obtained directly from the light-cut. Finally, we need to obtain the value $f_r(\mathbf{x}_i)$. Because creating BRDF cuts would be too expensive and require additional precomputation, we sample the BRDF on the fly. Note that this could be problematic with high frequency BRDFs since there is no BRDF cut to ensure we use small clusters in important and highly varying regions of the BRDF. Currently we rely on the light cut to ensure clusters are small enough in important regions to ensure sufficient sampling of the BRDF. This is similar to importance sampling and works well in practice. Finally, with all terms computed, we compute the product and add it to the sum.

Note that the algorithm can also be easily extended to account for rigid dynamic objects using the shadow fields algorithm [Zhou et al. 2005]. In this case, precomputed visibility fields are computed as the effect an object would have on the visibility of a neighboring object. At runtime, the visibility for a pixel is computed by locating nearby objects whose shadow fields contain the point $\mathbf{x}_i$, interpolating a visibility function $V_k(\mathbf{x}_i)$ caused by each neighbor object using linear interpolation from that object's shadow field cuts, and finally computing the full visibility function, which is the product of these per-object visibility functions:

$$B = \sum L(\mathbf{x}_i)\, f_r(\mathbf{x}_i)\ (V_1(\mathbf{x}_i)\, V_2(\mathbf{x}_i) \cdots V_n(\mathbf{x}_i))$$

Again, although these steps are performed in sequence logically, in practice the entire process can be implemented in a single traversal, requiring no storage of temporary merged cuts. We have not yet implemented shadow fields in our system, but it is a very promising future direction.

Figure 6: *The upper 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. The first two images in the bottom row shows editing of the puff chair model with different materials; the right two images show ward anisotropic BRDF applied on the teapot and a star specular map applied on the sphere.*

GPU implementation. Our GPU-based rendering system is a simple translation of the algorithm to the GPU. First we must map our data to textures so they can be used in shaders. Our precomputed data is a large set of sparse, variable sized vectors. We use a greedy algorithm to pack them into a 3D texture so that each vector fits into a single row of the texture. This allows shaders to treat the vectors as a 1D vector, as they truly are, and thus avoids the 1D to 2D texture coordinate mapping commonly required when mapping data to textures. 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.

We must also map our lighting values to textures. Some of this data will be static (the light sample positions and normals) and some will be dynamic (the light color). Although it is simpler to construct the light tree with a standard ordering of lights, we store the textures using postorder indices so they can be easily indexed when traversing the cut. We simply allocate a square texture large enough to hold the entire light tree. We upload the position and normal information immediately and upload the color information when it becomes available. Note that the normal information will of course be ignored for environment lighting, but is required for indirect local lighting.

At runtime we must sample lighting and perform shading. We have two ways to sample lighting. For direct environment lighting we directly sample a cubemap at the mipmap level with approximately the same number of texels as we have light samples. For local indirect lighting we use a standard shadow cube map approach. Instead of evaluating shadows per pixel as in standard rendering, we use the light textures described above and perform the shadow test for the light positions. We of course could use any approach to compute the direct diffuse radiance for the light samples - simple unshadowed point lights, shadow maps, or even another PRT approach. We then download the data to the CPU and generate the full light tree. We then upload the resulting full tree back to the lighting texture.

We also generate the light cut at this stage. This implementation is as simple as generating the precomputed visibility cuts except we select the node to split based on the magnitude of the value at each cut node instead of the error. This is similar to importance sampling where more samples (more cut nodes) are used in areas where the lighting is stronger. The light cut is uploaded to a 1D texture using a similar format as the visibility cut data.

Shading a fragment is expensive, so our runtime algorithm uses deferred shading to process only the visible fragments. In the first pass we render to a deep deferred shading buffer. We use a geometry shader to generate vertex indices and barycentric coordinates to be interpolated and passed to the fragment shader. The fragment shader then stores the barycentric coordinates, material ID, position, normal, and per-pixel BRDF information. Note that this step isn't strictly necessary, we could have simply primed the z-buffer and performed shading in the second pass if that provided performance benefits. Our implementation allows the BRDF to use 8 floating point values. This was sufficient for the BRDFs we implemented but this could easily be extended to 16 or more values on current hardware. Note that all our BRDF inputs in this shader are actually textures and so trivially allow spatially varying BRDF parameters. Constant BRDF parameters are simply represented as a 1x1 texture. We have also included normal perturbation by a bump or normal map in this step. Other similar techniques could easily be added.

The second pass performs the final shading calculations. This is essentially a direct implementation of the approach described in Section 3.2, interpolating 3 visibility cuts and multiplying the result by the light cut and dynamically sampled BRDF values. We load the data from the deferred shading buffer and use the vertex indices to lookup the cut coordinates in the data pointer texture. We also initialize the light cut coordinates trivially to 0. If we're calculating local lighting we compute the direct lighting value first. Then we start the loop, finding the current cut node with the deepest postorder index and the maximum leftleaf value. If the maximum leftleaf value is less than the deepest index then the nodes are all in

Figure 7: *The first two images on the upper row compares images with direct illumination only and the addition of one bounce indirect illumination. The third and fourth images show realistic material design of this bedroom model. The bottom row shows the table scene (the scene geometry is courtesy of [Ng et al. 2004]) rendered with dynamically changing material and lighting.*

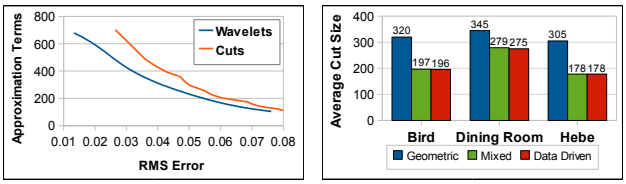
the same subtree and we compute the interpolated visibility value and sample the BRDF at the cluster center. We add the resulting value to the current pixel color. Regardless of the alignment of the nodes, each cut's current node is advanced if its current node had the deepest index. Finally, we check if any cuts have advanced past their end and terminate if they have, writing the final color sum to the output.

BRDFs are sampled entirely on the fly and no BRDF cuts are ever constructed. This gives great flexibility with BRDFs and enables BRDFs and their parameters to be selected on a per-pixel basis. However, this also imposes a few limitations. First, without a BRDF cut we cannot ensure that the BRDF is sampled finely enough in the right areas. For instance, it is possible for highly specular lobes to be missed entirely because the lighting and visibility cuts were relatively high in the light tree, and the BRDF sample taken for the node containing the lobe was relatively small. This is only a problem for high frequency BRDFs. This can result in the loss of specular highlights. However, because we construct our light cut based on light intensity, we sample our BRDF approximately as we would using importance sampling in an offline raytracer. In practice this seems to be sufficient in most cases. However, we are interested in solutions to this problem. We believe that for certain classes of BRDFs indications of the frequency of the BRDF 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.

5 Results

We performed tests on a variety of scenes: Bedroom (Figure 1(c)), Dining room (Figure 6), Hebe (Figure 6), Motobike (Figure 1(a)), Table (Figure 1(b)), and Teapot. The table scene is courtesy of [Ng et al. 2004] All performance statistics are gathered on a quad-core 2.0GHz computer with an NVIDIA 8800GTX graphics card.

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 usually selected by minimizing the total L^2 error of the entire function, or for a specified number of terms. Our experiments show that they are both very efficient at nonlinear approximation. However, cuts are more flexible in that they can operate on unstructured sample sets whereas wavelets already require parametrization. Therefore we cannot easily implement a wavelet version of our system for complete comparison.



(a) (b)

Figure 8: *(a) Comparison of compression rates using wavelets vs. cuts. (b) Comparison of different clustering schemes for light tree construction.*

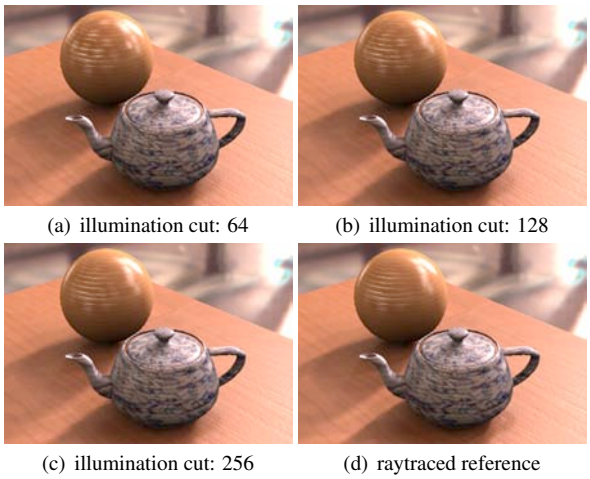
In order to compare the two we only focus on their ability to efficiently represent visibility functions as defined over a cubemap. For wavelets we specify a variety of L^2 errors and find the number of wavelet coefficients required to approximate the original cubemap with at most the specified error. For cuts, we specify a variety of per node variance thresholds and generate the cuts as described in Section 3.2. However, in both cases we plot the root mean squared error of the resulting approximation by the number of terms saved, both averaged over all the vertices in a test scene. The resulting graph using a 6x128x128 cubemap is presented in Figure 8(a).

As shown, cuts do in fact require a relatively small increase of terms to represent the same cubemap function to equal accuracy. For the same error, cuts require about 10$\sim$15% more terms than wavelets. However, note that since our comparison criteria is the RMS error, a cut selection algorithm that minimizes the total error over all clusters would likely to perform better than our current one which limits the per-cluster error. We believe the loss of compression is made up for by the cut representation's simplicity, efficient computational algorithms and flexibility of application. In addition, we do find that cuts have the nice property that they will never inflate sparse high-frequency data as wavelets can; and for very highly accurate approximation, they tend to reach a perfect representation with fewer nodes, even with high frequency data.

Data-driven construction of light tree. Figure 8(b) shows the average cut size for the different clustering schemes we tested for light tree construction. We show average cut size for three values of β: 1.0, representing pure geometric clustering; 0.5, representing the mixed mode; and 0.0, representing pure data-driven clus-

				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	Local	20min	-	-	-	114 MB	643	0.13	-	1.4 / 1.2 / 1.1
Teapot	86K	Local	16min	-	-	-	78 MB	261	0.058	-	7 / 6 / 4
Bedroom	105K	Local	30min	-	-	-	219 MB	662	0.13	-	2 / 2 / 1.5

Figure 9: *Detailed statistics of our algorithm. From left to right, the columns present the vertex count of each model, illumination type (environment or local), precomputation time, and precomputation profiles for the simple geometric based light tree construction and our proposed data-driven construction of light tree. The data-driven approach requires twice precomputation time, but reduces overall data size by an average of 20∼30%. Note that the cut size for the local lighting case is typically much larger than the environment lighting case. The last column presents the rendering rates using a selection of illumination cut size: 128/256/512.*



(a) illumination cut: 64 (b) illumination cut: 128

(c) illumination cut: 256 (d) raytraced reference

Figure 10: *Image quality comparison with ray traced reference by varying the illumination cut size. Note the artifacts on the reflection with 64 and 128 cut size; as we increase the cut size, the image quality quickly converges to reference.*

tering. Data-driven clustering is clearly a significant improvement over pure geometrical cluster, by as much as 40% and by 20∼30% on average. The mixed mode only causes very small increases in average cut size and improves rendering by allowing more efficient approximation of the lighting.

Precomputation. The precomputation stage is fairly efficient. The majority of the time during precomputation is spent performing visibility sampling. Since we use a simple, unoptimized raytracer to perform visibility sampling we believe precomputation time could be reduced significantly by an improved packet based raytracer or performing visibility sampling on the GPU.

Statistics for all the test scenes presented are given in Figure 9. These statistics include the compressed file size, average cut size, and the RMS error of the resulting data. Note that our storage requirements are less than other PRT systems because we only precompute visibility which can be compressed effectively. Note also that the data-driven clustering helps reduce the storage size signficantly at the cost of longer precomputation. We also observe that the visibility cuts computed for local lighting are longer than those for environment lighting. We expect this because for environment lighting visibility is a binary function but for local indirect lighting it is continuous.

Rendering. performance for each of the scenes we tested is given in Figure 9. Frame rates are reported for illumination cut sizes of 128, 256, and 512 nodes. Clearly increasing this size will better account for important error in illumination and improve accuracy. However, we found in general 256∼512 works for all our test scenes with no noticeable artifacts. Run time frame rates are interactive for simple scenes and 2∼4 fps for more complicated ones. Note that local lighting tends to be slower on average because the visibility cuts produced are significantly longer on average.

Figure 6 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. Figure 7(e) shows the table scene rendered using only direct lighting using shadow mapping. Figures 7(f), 7(g), and 7(h) show the same scene with the same lighting using direct and indirect local lighting. The images are much more convincing with indirect lighting enabled. Figure 6 demonstrates a variety of per-pixel shading effects. Bump mapping provides fine texture, a specular map controls the BRDF to create patterns, and texturing is used on the floor and dresser. All these effects are generated entirely at run time and require no additional preprocessing.

Varying illumination cut size. Finally, we present results regarding the error of our images with respect to the ground truth renderings obtained using an offline raytracer. Figure 10 shows renderings from our system using various light cut sizes and a reference raytraced image. The errors are small. With very small lightcuts shadows are blurred due to subsampling. Even with longer lightcuts, some reflections are inaccurate because the lighting is approximated. However, the error is barely noticeable.

6 Conclusions and Future Work

We have presented an interactive rendering system for realistic lighting and material design under complex illumination with arbitrary BRDFs. The key contribution which makes this possible is an efficient algorithm for computing sums and products of an arbitrary number of piecewise constant approximations called *cuts*. We prove an error bound on multi-product integrals computed using our algorithm. We suggest a two-pass data-driven clustering algorithm which improves compression by 20∼30%. Our system precomputes only visibility and demonstrates our efficient cut algorithm by interpolating visibility and computing the light integral at each pixel. Because of this we are also able to perform dynamic per-pixel effects such as bump mapping and spatially varying BRDF parameters, techniques not possible in most other PRT systems.

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 tessellating meshes may inflate data sizes unnecessarily. Therefore we are interested in studying both nonlinear interpolation schemes and performing efficient spatial compression of visibility data. Moreover, we plan to study better ways to guide BRDF sampling on the fly. Finally, we believe that cuts are a gen-

eral and effective approximation and, combined with our efficient computational algorithm, could find use on problems besides PRT.

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.

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., DURAND, F., AND RAMAMOORTHI, R. 2007. A precomputed polynomial representation for interactive brdf editing with global illumination. *ACM Trans. Graph. -*, -, –.

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. 1998. Rendering synthetic objects into real scenes. In *Proceedings of SIGGRAPH '98*, ACM, New York, NY, USA, 189–198.

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

DRETTAKIS, G., AND SILLION, F. X. 1997. Interactive update of global illumination using a line-space hierarchy. In *Proceedings of SIGGRAPH '97*, ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, 57–64.

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.

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

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

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.

KONTKANEN, J., TURQUIN, E., HOLZSCHUCH, N., AND SILLION, F. 2006. Wavelet radiance transport for interactive indirect lighting. In *Proceedings of Eurographics Symposium on Rendering*, 161–171.

KONTKANEN, J., TURQUIN, E., HOLZSCHUCH, N., AND SILLION, F. 2007. Interactive illumination with coherent shadow maps. In *Proceedings of Eurographics Symposium on Rendering*, 61–72.

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.

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

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

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.

WALD, I., BENTHIN, C., AND SLUSALLEK, P. 2003. Interactive Global Illumination in Complex and Highly Occluded Environments. In *Proceedings of the 14th Eurographics Workshop on Rendering*.

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.