All-Frequency Relighting of Glossy Objects

RUI WANG, JOHN TRAN and DAVID LUEBKE
University of Virginia

We present a technique for interactive rendering of glossy objects in complex and dynamic lighting environments that captures interreflections and all-frequency shadows. Our system is based on precomputed radiance transfer and separable BRDF approximation. We factor glossy BRDFs using a separable decomposition and keep only a few low-order approximation terms, each consisting of a purely view-dependent and a purely light-dependent component. In the precomputation step, for every vertex we sample its visibility and compute a direct illumination transport vector corresponding to each BRDF term. We use modern graphics hardware to accelerate this step, and further compress the data using a non-linear wavelet approximation. The direct illumination pass is followed by one or more interreflection passes, each of which gathers compressed transport vectors from the previous pass to produce global illumination transport vectors. To render at run time, we dynamically sample the lighting to produce a light vector, also represented in a wavelet basis. We compute the inner product of the light vector with the precomputed transport vectors, and the results are further combined with the BRDF view-dependent components to produce vertex colors. We describe acceleration of the rendering algorithm using programmable graphics hardware, and discuss the limitations and tradeoffs imposed by the hardware.

Categories and Subject Descriptors: I.3.3 [**Computer Graphics**]: Picture/Image Generation; I.3.7 [**Computer Graphics**]: Three-Dimensional Graphics and Realism; I.3.1 [**Computer Graphics**]: Hardware Architecture

General Terms: Algorithms, Performance

Additional Key Words and Phrases: BRDF, Separable Approximation, Global Illumination, Graphics Hardware, Precomputed Radiance Transfer, Haar Wavelets

1. INTRODUCTION

Realistic rendering of objects at interactive rates continues to present a great challenge in computer graphics. To achieve a high level of realism, a rendering system must be able to model physically based surface reflectance, allow for large-scale lighting environments, and produce global illumination effects such as shadowing and interreflections. Conventional image synthesis techniques such as Monte Carlo ray tracing [Kajiya 1986; Veach 1997], photon mapping [Jensen 1996], and radiosity [Cohen and Wallace 1993] simulate complex illumination effects, but are too expensive for real-time applications. Image relighting [Dorsey et al. 1991; Ashikhmin and Shirley 2002; Ng et al. 2003] and environment matting techniques [Zongker et al. 1999; Peers and Dutré 2003] faithfully reproduce realistic images under dynami-

Authors' address: Department of Computer Science, University of Virginia, 151 Engineer's Way, Charlottesville, VA 22904; email: {ruiwang, johntran, luebke} @ cs.virginia.edu.

cally changing lights; however, they assume a fixed viewpoint. Light fields [Levoy and Hanrahan 1996; Gortler et al. 1996] and surface light fields [Wood et al. 2000; Chen et al. 2002] capture realistic view-dependent appearances of physical objects at interactive rates; however, they assume a fixed lighting environment. Our goal is to build a system that allows the user to interactively manipulate both the lighting and the viewpoint, while taking into account complex illumination effects, including intricate hard and soft shadows, and glossy interreflections.

In 2002, Sloan et al. [2002] introduced the precomputed radiance transfer (PRT) technique for rendering models with low-frequency environment maps. They presented a compact representation of the light transport function in a vector form on spherical harmonic basis. Relighting then reduces to a simple inner product of a light vector, also represented in spherical harmonics basis, with the precomputed transport vectors. View-dependent rendering of glossy surfaces uses precomputed transport matrices instead of transport vectors. This technique is fast and compact; however, it is limited to low-frequency lighting environments due to the approximation using low-order (25D) spherical harmonics. Therefore, it is only accurate for very soft shadows. To alleviate this problem, Ng et al. [2003] proposed using non-linear wavelet approximation to adaptively choose the best set of wavelet bases for approximation of the lighting over a broad range of frequencies. This improved approach renders both soft and hard shadows at interactive rates, and is commonly referred to as *all-frequency* PRT. For changing viewpoint, they were limited to diffuse BRDFs. This is because glossy BRDFs require an additional sampling in the view to produce view-dependent effects, the cost of which is significantly higher. Later, Ng et al. [2004] developed wavelet triple product integrals for all-frequency relighting of glossy BRDFs. Their system provides very high quality but is not interactive, requiring a few seconds to render.

Recently Liu et al. [2004] and we [Wang et al. 2004] independently proposed a new formulation that combines separable BRDF approximation [Kautz and McCool 1999] with the wavelet-based PRT for interactive relighting of glossy objects. We apply a separable decomposition to approximate glossy BRDFs with a few low-order terms, each consisting of a purely light-dependent and a purely view-dependent component. The key idea is that the light-dependent components are baked into precomputation as part of the transport function definition, and the view-dependent components are accessed during rendering to produce view-dependent effects. However, both papers are limited to direct illumination, ignoring interreflections. In this article we show through derivation that interreflections can be efficiently simulated in the same framework without sacrificing rendering performance. The only additional cost is at precomputation, where the direct illumination pass is followed by one or more interreflection passes. Each interreflection pass gathers transport vectors from the previous pass, producing a new set of vectors carrying bounced illumination. The additional computation simulates both diffuse and glossy interreflections at the same time, and does not increase precomputed data size, allowing us to portray interreflections at the same rendering speed.

In general, any precomputation technique is a form of sampling in the 6D space of light direction, view direction and surface location, which must be densely sampled in order to resolve high-frequency illumination effects. For example, high-frequency shadows require dense sampling in the light direction; highly glossy surfaces require dense sampling in both the light direction and view direction. Due to the difficulty in precomputing, storing, and rendering such high dimensional datasets, current interactive techniques choose to dramatically reduce the sampling rates in one or several of the dimensions. For example, Sloan et al. [2002] use low-order spherical harmonics to represent the lighting and BRDFs, producing low-frequency sampling in both the light direction and view direction. By bandlimiting the illumination, their approach is only accurate for very low-frequency lighting as well as view-dependent effects, manifested by the lack of sharp shadows and highly glossy surfaces. Other approaches such as light fields or image relighting simply ignore one of the sampling dimensions, such as the light direction or the view direction. The wavelet triple product integrals technique by Ng et al. [2004] preserves high frequency sampling in all six dimensions, but their system is not interactive and requires a large amount of memory to compute. Our system approaches the sampling problem by choosing a high sampling rate for the lighting but a low sampling rate for the view. This is achieved by combining non-linear wavelet approximation for the lighting with low-order separable approximation for the BRDF. As a result, we have bandlimited high-frequency specularities; however, unlike the spherical harmonics based approach, our renderings preserve intricate all-frequency shadows while remain at interactive rates.

In low-frequency PRT, various global illumination effects have been incorporated, such as interreflections [Sloan et al. 2002] and translucency [Sloan et al. 2003]. Because of its very low sampling rate in the lighting (e.g. 25 spherical harmonics bases), practically any complex illumination effect can be precomputed using a unified approach, which illuminates the scene with each lighting basis in turn and computes the per-vertex radiance response to each basis. These radiance responses then directly form the transport functions. However, in all-frequency PRT, the required sampling rate is orders of magnitude higher (e.g. $24,576$ wavelet bases), making the same approach intractable in terms of computation time and storage size. View-dependent effects are even more difficult to handle as they require additional sampling in the view. Our system circumvents these problems in two ways. First, we densely sample the source lighting in the direct illumination pass, which is efficiently computed using graphics hardware; then each interreflection pass only sparsely samples and gathers illumination transport (in compressed form) from the previous pass. This eliminates the need to obtain a full transport function per vertex prior to wavelet approximation. Second, by projecting our transport functions onto the low-order BRDF bases (which comes from the BRDF factorization), we have bandlimited the frequency content in the view, reducing its sampling rate without sacrificing the high frequencies in the lighting (hence the sharp shadows). These design decisions allow us to precompute interreflections in a practical amount of computation time with reasonable storage requirements.

In [Wang et al. 2004] we have discussed the use of programmable graphics hardware to accelerate the precomputation step. For the rendering step, our CPU implementation is interactive for changing viewpoint, but barely interactive for changing light. In this article we have implemented the entire rendering algorithm on modern graphics hardware. The most costly computation during lighting change is a sparse vector inner product, which we have accelerated using a GPU-based algorithm. Our sparse vectors are represented on the GPU in a similar way as described by Bolz et al. [2003] and Krüger and Westermann [2003]. However, we exploit the fact that our sparse vectors have the same number of non-zero elements to better optimize the data layout and improve texture access performance. Finally, we present and discuss several different possible GPU implementations of the rendering algorithm.

2. PREVIOUS WORK

2.1 Precomputation Techniques

In computer graphics, people have become increasingly interested in illuminating models by captured environment maps. One big challenge is that this type of illumination requires computing a very expensive integration over all lighting directions, preventing it from being used directly in real-time. Several techniques exploit precomputed information to help achieve interactive rendering rates. In this section we briefly review these techniques and discuss their limitations.

Environment maps were first introduced by Blinn and Newell [1976] to approximate specular reflection of distant environments. Since then, several approaches have been proposed to simulate diffuse and glossy reflections based on preconvolution of environment maps [Greene 1986; Kautz and McCool 2000; Ramamoorthi and Hanrahan 2001; 2002]. However, they ignore visibility and therefore cannot handle self-shadowing or interreflections.

Image relighting techniques precompute global illumination solutions for a set of lighting bases, such as points [Dorsey et al. 1991], polynomials [Malzbender et al. 2001], steerable functions [Ashikhmin and Shirley 2002], compressed principal component bases [Debevec et al. 2000], and wavelets [Ng et al. 2003]. These techniques can handle dynamic lighting change and complex illumination effects; however, rendering requires a fixed view. Shadowing techniques have also been presented such as convolution textures [Soler and Sillion 1998], attenuation maps [Agrawala et al. 2000], and precomputed visibility [Heidrich et al. 2000], but they do not allow for real-time dynamic lighting or complex self-shadowing geometry.

Light fields [Levoy and Hanrahan 1996; Gortler et al. 1996] record radiance samples as they pass through a pair of viewing planes. Surface light fields [Wood et al. 2000; Chen et al. 2002] record exitant radiance in sampled directions over an object's surface. These techniques allow for view-dependent rendering of complex surface appearances, but are limited to static lighting environments.

Recently, Sloan et al. [2002] introduced precomputed radiance transfer (PRT) for interactive rendering of objects under low-frequency dynamic lighting. They precompute light transport functions, which capture the way an object shadows, scatters, and reflects light, and represent them as transport vectors in a low-order spherical harmonics (SH) basis. Rendering is then reduced to a simple inner product of a light vector, also represented in the SH basis, with the precomputed transport vector. In the case of glossy BRDFs, a transport matrix is precomputed for every vertex to allow for view-dependent rendering. Their work was later extended [Kautz et al. 2002; Lehtinen and Kautz 2003; Sloan et al. 2003] to achieve better compression rates and improve rendering performance.

Spherical harmonics have also been exploited by Sillion et al. [1991], Cabral et al. [1987] and Westin et al. [1992] as a compact representation for the BRDF and global illumination solutions. As Ng et al. [2003] point out, a low-order SH basis is only accurate when approximating very low-frequency signals, and therefore cannot reproduce high-frequency lighting and shadowing effects. Instead, they use non-linear wavelet approximation to represent the lighting and transport vectors, achieving all-frequency illumination and shadows at interactive rates. They demonstrated the technique primarily for image relighting. For changing viewpoint, it was limited to diffuse objects. In a recent paper, Ng et al. [2004] developed efficient wavelet triple product integrals for all-frequency relighting. This new method is accurate, but rendering requires a few seconds, and it is not clear how the system can extend to include global illumination effects such as interreflections.

Recently Liu et al. [2004] and we [Wang et al. 2004] independently proposed a new formulation of all-frequency PRT for relighting glossy BRDFs. The new approach combines separable BRDF approximation [Kautz and McCool 1999] with Ng's wavelet-based technique to produce all-frequency shadows and low-frequency view-dependent effects. Specifically, we use a separable decomposition to approximate glossy BRDFs with a few (K) low-order terms, each consisting of a purely light-dependent part and a purely view-dependent part. For each vertex, we precompute K transport vectors, corresponding to each BRDF term. To compress the transport vectors, we use a non-linear wavelet approximation as in [Ng et al. 2003]. Liu et al. [2004] also experimented with data compression using clustered principal component analysis (CPCA) [Sloan et al. 2003], exploiting spatial coherence. At run-time, when lighting changes, we sample the environment map and apply a wavelet transform to produce a light vector. The light vector is then multiplied with each of the K transport vectors per vertex to produce K colors, which we call the *light-dependent K-vector*. When viewpoint changes, we use the view direction at each vertex to index into all K BRDF view maps, forming a *view-dependent K-vector*. Finally, the vertex color is computed as the dot product of the view-dependent K-vector with the light-dependent K-vector.

A limitation of both papers is the assumption of direct illumination. In this paper we will elaborate on developing the formulation to incorporate interreflections in precomputation, without affecting the rendering performance.

2.2 BRDF Factorization

BRDF factorization is a dimensionality reduction technique that has been successfully applied to interactive rendering [Kautz and McCool 1999], importance sampling [Lawrence et al. 2004], and precomputed radiance transfer [Liu et al. 2004; Wang et al. 2004]. The factorization can be achieved using a separable decomposition [Kautz and McCool 1999], homomorphic factorization [McCool et al. 2001] or chained matrix factorization [Suykens et al. 2003]. They all reduce a 4D BRDF to sums and products of several 2D functions, each stored as a 2D texture map for access by graphics hardware during rendering. DeYoung and Fournier [1997; 1995] have examined properties of a BRDF that affect its separability. Rusinkiewicz [1998] showed that proper reparametrization of a BRDF can improve its separability remarkably.

The current mathematical framework of our technique (Section 3.1) requires a BRDF parameterization in incident and view directions. To factorize, we typically

$\mathbf{x}$	Surface location
$\mathbf{n}$	Surface normal
ω_i	Incident direction
ω_o	View direction
V	Visibility
f_r	Surface BRDF
L_o	Outgoing radiance in the direct illumination pass
L_o^i	Bounced outgoing radiance in the i-th interreflection pass
L	Distant environment lighting
L^i	Local lighting in the i-th interreflection pass
g_k	Light map of the k-th BRDF approximation term
h_k	View map of the k-th BRDF approximation term
T_k	The k-th transport function (corresponding to the k-th BRDF term)
T_k^i	The k-th bounced transport function in the i-th interreflection pass
$\overline{T}_k$	The k-th complement transport function
$\widetilde{\mathbf{x}}$ or $\widetilde{\mathbf{x}}(\mathbf{x}, \omega)$	The surface location that a ray originating from $\mathbf{x}$ hits in direction ω

Fig. 1. Notation used in this paper

apply a separable decomposition as in [Kautz and McCool 1999], which approximates a 4D BRDF f_r as the sum of products of 2D functions g_k's and h_k's:

$$f_r(\omega_i, \omega_o) \approx \sum_{k=1}^{K} g_k(\omega_i)\, h_k(\omega_o) \tag{1}$$

where ω_i is the incident direction, ω_o is the view direction, and K is the number of approximation terms. The accuracy of this approximation increases as more terms are used. Although a highly glossy BRDF requires a significant number of terms, a few low-order terms suffice for BRDFs with low-frequency specular components. The factorization is typically computed using singular value decomposition (SVD). In this case, the BRDF is first discretized by sampling the incident direction and view direction separately, constructing a BRDF matrix $\mathbf{M}$. Then an SVD is applied on $\mathbf{M}$, resulting in a left matrix $\mathbf{U}$ and a right matrix $\mathbf{V}$, each consisting of K column vectors. These column vectors correspond directly to g_k's and h_k's (see Section 3.3 for detail). We call g_k's the BRDF *light maps* since each of them is a 2D texture map indexed only by the incident direction. Similarly, h_k's are called the *view maps* since each of them is a 2D texture map indexed only by the view direction. Instead of SVD, one can also apply non-negative matrix factorization (NMF) [Lee and Seung 1999] similarly to the work by Lawrence et al. [2004]. NMF produces strictly positive coefficients, which is desirable in certain situations.

3. ALGORITHMS

In this section, we describe algorithms for precomputing transport vectors using separable BRDF decomposition and non-linear wavelet approximation. As in other PRT work, we only consider distant illumination and ignore near-field illumination. Figure 1 lists notation used in this paper.

3.1 Direct Illumination

We first consider the rendering equation [Kajiya 1986] for direct illumination, which describes outgoing radiance L_o at a surface location $\mathbf{x}$ in view direction ω_o as:

$$L_o(\mathbf{x}, \omega_o) = \int_\Omega L(\omega_i)\, f_r(\mathbf{x}, \omega_i, \omega_o)\, V(\mathbf{x}, \omega_i)\, (\mathbf{n} \cdot \omega_i)\, d\omega_i \tag{2}$$

where L is source lighting, ω_i is incident direction, f_r is surface BRDF, $\mathbf{n}$ is surface normal, and V is visibility. The domain of integration Ω is the sphere of incoming directions in the global coordinate system, and is parameterized the same way as L. For simplicity, we only consider spatially uniform BRDF. Since the precomputation occurs per vertex, it is straightforward to handle spatially varying BRDFs that are linear combinations of several basis BRDFs or modulated by texture maps. Using separable BRDF decomposition (Eq 1), we approximate $L_o(\mathbf{x}, \omega_o)$ as:

$$\begin{aligned} L_o(\mathbf{x}, \omega_o) \ &\approx \ \int_\Omega L(\omega_i) \left(\sum_{k=1}^{K} g_k(\omega_i)\, h_k(\omega_o) \right) V(\mathbf{x}, \omega_i)\, (\mathbf{n} \cdot \omega_i)\, d\omega_i \\ &= \sum_{k=1}^{K} \left(h_k(\omega_o) \int_\Omega L(\omega_i)\, g_k(\omega_i)\, V(\mathbf{x}, \omega_i)\, (\mathbf{n} \cdot \omega_i)\, d\omega_i \right) \end{aligned}$$

For each BRDF term, we define its associated transport function as:

$$T_k(\mathbf{x}, \omega_i) = g_k(\omega_i)\, V(\mathbf{x}, \omega_i)\, (\mathbf{n} \cdot \omega_i) \tag{3}$$

which can be precomputed and allow us to further express L_o as:

$$L_o(\mathbf{x}, \omega_o) = \sum_{k=1}^{K} \left(h_k(\omega_o) \int_\Omega L(\omega_i)\, T_k(\mathbf{x}, \omega_i)\, d\omega_i \right) \tag{4}$$

Note that due to the BRDF factorization, the integrand is no longer dependent on ω_o. Using numerical cubature we evaluate the integral as:

$$L_o(\mathbf{x}, \omega_o) = \sum_{k=1}^{K} \left(h_k(\omega_o) \sum_j L(\omega_j)\, T_k(\mathbf{x}, \omega_j) \right)$$

where ω_j is the discretized incident direction. If we fix a surface location $\mathbf{x}$ and its view direction ω_o, the view-dependent color is:

$$L_o = \sum_{k=1}^{K} \left(h_k(\omega_o) \sum_j L(\omega_j)\, T_k(\omega_j) \right)$$

which we put in compact matrix notation as:

$$L_o = \mathbf{h} \cdot (\mathbf{T} \times \mathbf{L}) \tag{5}$$

Here $\mathbf{L}$ is the light vector, $\times$ denotes a matrix-vector multiplication, and $\cdot$ denotes a vector inner product. $\mathbf{h}$ is called the *view-dependent K-vector*, which is produced by texture lookups of ω_o into all BRDF view maps. $\mathbf{T}$ is the precomputed transport matrix, the rows of which are the transport functions defined in Eq 3. The multiplication of T with L produces a K-vector which we call the *light-dependent*

K-vector and use symbol $\mathbf{g}$ to denote. Now the view-dependent color is simply the dot product of the two K-vectors: $L_o = \mathbf{h} \cdot \mathbf{g}$

Notice that Eq 5 still holds when $\mathbf{L}$ and the row vectors of $\mathbf{T}$ are expressed in any orthonormal basis, such as spherical harmonics or wavelets. In fact, an approximation using such orthonormal bases is very important for two reasons: first, they provide a compact representation to dramatically reduce the storage size of precomputed transport vectors; second, they allow for fast rendering by reducing the computation of the lighting integral to a low-dimensional vector inner product.

As Ng et al. [2003] point out, spherical harmonics basis localizes poorly in the spatial domain, and hence is only accurate for approximating very low-frequency lighting, such as large area sources. On the contrary, point set basis localizes poorly in the frequency domain, and hence is only accurate for approximating very high-frequency lighting such as small area sources. Instead, wavelet basis localizes well in both frequency and spatial domains, and therefore is very efficient at approximating lighting over a broad range of frequencies. They demonstrated that using non-linear wavelet approximation preserves all-frequency illumination effects, such as both hard and soft shadows. For these reasons, we use non-linear wavelet approximation to compress our precomputed datasets.

Eq 5 is similar to the matrix notation in [Ng et al. 2003]. The difference is that our transport functions are modulated by the BRDF light maps, and rendering is modulated by the BRDF view maps. Essentially we can think of the new formulation as a projection of the full 6D transport function, defined as the triple product of f_r, V and $(\mathbf{n} \cdot \omega_i)$, onto the low-order BRDF bases, which comes from the BRDF factorization. In this way, we have reduced the sampling rate in the view to K terms and effectively constrained the precomputed datasets to a manageable size for rendering. By bandlimiting the frequency content in the view, however, we are limited to low-frequency specularities and smoothly varying view-dependent effects. On the other hand, because we represent the lighting using non-linear wavelet approximation, our renderings preserve intricate hard and soft shadows, which are very difficult to achieve using other approaches.

3.2 Global Illumination

Now we consider including global illumination effects into precomputation. We use a multi-pass gathering approach that is a straightforward adaptation of the Jacobi iterations well known in radiosity. We call the direct illumination step in Section 3.1 the 0th pass, which computes a set of *direct transport vectors*. In classic radiosity, global illumination is computed one step at a time: the initial step (the 0th pass) computes outgoing radiance for every surface location due to direct illumination; each following pass computes bounced outgoing radiance for every surface location by gathering reflected radiance from all other surface locations. Our method is different in that for every surface location, we gather transport vectors, instead of radiance values, from all other surface locations. The gathering process produces a new set of transport vectors, which we call the *bounced transport vectors*. In the following we derive the transport functions for the first interreflection bounce. The superscripts in all symbols denote the iteration number.

Illumination in the first bounce comes from the surface outgoing radiance after the direct illumination pass. Now lighting depends on surface location $\mathbf{x}$ and cannot

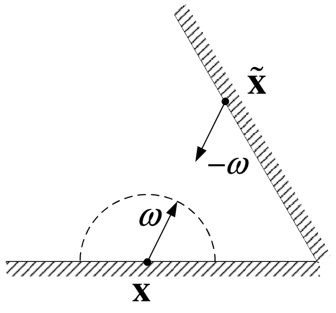


Fig. 2. An example of $L^1(\mathbf{x},\omega) = L_o(\widetilde{\mathbf{x}},-\omega)$.

be assumed distant any more. The outgoing radiance of the first bound is:

$$L_o^1(\mathbf{x},\omega_o) = \int_\Omega L^1(\mathbf{x},\omega)\, f_r(\mathbf{x},\omega,\omega_o)\,(1 - V(\mathbf{x},\omega))\,(\mathbf{n}\cdot\omega)\,d\omega$$

where L^1 is the local lighting observed by surface location $\mathbf{x}$. Similar to the direction illumination case, we substitute Eq 1 into the above equation and define a transport function associated with each BRDF term as:

$$\overline{T}_k(\mathbf{x},\omega) = g_k(\omega)\,(1 - V(\mathbf{x},\omega))\,(\mathbf{n}\cdot\omega) \tag{6}$$

Note that this definition differs with Eq 3 only by a flip in the visibility term, thus we call it the *complement transport function.* Now we can express L_o^1 as:

$$L_o^1(\mathbf{x},\omega_o) = \sum_{k=1}^{K}\left(h_k(\omega_o)\int_\Omega L^1(\mathbf{x},\omega)\,\overline{T}_k(\mathbf{x},\omega)\,d\omega\right) \tag{7}$$

L^1 is in fact the outgoing radiance L_o via the following relation:

$$L^1(\mathbf{x},\omega) = L_o(\widetilde{\mathbf{x}},-\omega)$$

Figure 2 shows the relation. Here $\widetilde{\mathbf{x}} = \widetilde{\mathbf{x}}(\mathbf{x},\omega)$ refers to the surface location that a ray originating from $\mathbf{x}$ intersects in direction ω. Intuitively, it is the closest point that $\mathbf{x}$ *sees* in direction ω. Given this, the integral in Eq 7 becomes

$$\int_\Omega L_o(\widetilde{\mathbf{x}},-\omega)\,\overline{T}_k(\mathbf{x},\omega)\,d\omega$$

We then use Eq 4 to substitute L_o in the integral and rearrange the terms:

$$\int_\Omega L(\omega_i)\left(\int_\Omega \sum_{\lambda=1}^{\widetilde{K}} h_\lambda(\widetilde{\mathbf{x}},-\omega)\,T_\lambda(\widetilde{\mathbf{x}},\omega_i)\,\overline{T}_k(\mathbf{x},\omega)\,d\omega\right)d\omega_i$$

Notice that a model can contain several BRDFs and each of them may have a different number of approximation terms, therefore the BRDF light maps and view maps both depend on the surface location, as we explicitly indicate above. Here $\widetilde{K} = K(\widetilde{\mathbf{x}})$ denotes the number of approximation terms used for the BRDF at $\widetilde{\mathbf{x}}$. The inner integral (inside the big parentheses) produces a function that depends on ω_i, and we define it as the *bounced transport function*

$$T_k^1(\mathbf{x},\omega_i) = \int_\Omega \sum_{\lambda=1}^{\widetilde{K}} h_\lambda(\widetilde{\mathbf{x}},-\omega)\,T_\lambda(\widetilde{\mathbf{x}},\omega_i)\,\overline{T}_k(\mathbf{x},\omega)\,d\omega \tag{8}$$

Intuitively, T_k^1 carries bounced illumination transport that is parameterized on the lighting, and can be explained as a gathering of direct transport functions T_λ over all points observed by $\mathbf{x}$. This gathering is valid precisely because illumination is a linear process. Now with T_k^1 we can rewrite Eq 7 by integrating over the distant source lighting L (instead of local lighting L^1):

$$L_o^1(\mathbf{x}, \omega_o) = \sum_{k=1}^{K} \left(h_k(\omega_o) \int_\Omega L(\omega_i)\, T_k^1(\mathbf{x}, \omega_i)\, d\omega_i \right) \tag{9}$$

Obviously this rendering equation has the same form as the direct illumination case (Eq 4), therefore the rendering algorithm remains unmodified. Using the same derivation, we obtain a recursive formula for the bounced transport functions in further interreflection passes:

$$T_k^{i+1}(\mathbf{x}, \omega_i) = \int_\Omega \sum_{\lambda=1}^{\widetilde{K}} h_\lambda(\widetilde{\mathbf{x}}, -\omega)\, T_\lambda^i(\widetilde{\mathbf{x}}, \omega_i)\, \overline{T}_k(\mathbf{x}, \omega)\, d\omega$$

Finally, the total outgoing radiance is the sum of all bounced radiance values:

$$L_o^{global}(\mathbf{x}, \omega_o) = L_o(\mathbf{x}, \omega_o) + L_o^1(\mathbf{x}, \omega_o) + L_o^2(\mathbf{x}, \omega_o) + ...$$

and correspondingly, the global illumination transport function is:

$$T_k^{global}(\mathbf{x}, \omega_i) = T_k(\mathbf{x}, \omega_i) + T_k^1(\mathbf{x}, \omega_i) + T_k^2(\mathbf{x}, \omega_i) + ...$$

The rendering equation again has the same form as both Eq 4 and Eq 9:

$$L_o^{global}(\mathbf{x}, \omega_o) = \sum_{k=1}^{K} \left(h_k(\omega_o) \int_\Omega L(\omega_i)\, T_k^{global}(\mathbf{x}, \omega_i)\, d\omega_i \right)$$

Note that the global illumination step simulates both diffuse and glossy interreflections consistently. In order to demonstrate distant lighting effects, we have constructed relatively *open* test scenes for all our examples, which means most regions are illuminated by rather short illumination paths. In this case, the energy redistributed at each bounce is decreasing rapidly, therefore computing a few bounces ($1 \sim 2$) usually suffices for visually pleasing results.

In low-frequency PRT [Sloan et al. 2002; Lehtinen and Kautz 2003], precomputing interreflections involves building a per-vertex incident transfer matrix that simulates the linear influence of source lighting to transferred incident radiance. The transfer matrix has properly encoded the bounced illumination transport upon arriving at a vertex; it is then convolved with the BRDF at that vertex, either on the fly or as a preprocessing, to produce the exiting radiance values. Since only the first 25 spherical harmonics bases are used for approximation, their transfer matrix is still relatively compact (up to 25×25). Applying this approach to all-frequency PRT, however, is impractical as the high dimensional sampling space would require building a transfer matrix that has a full size of $24,576 \times 24,576$. Even after using aggressive wavelet approximation in both dimensions (in fact, the data compressed this way would be very awkward and costly to render with), the matrix is still likely to be $2 \sim 3$ orders of magnitude bigger than using spherical harmonics. The problem is that such a matrix transfers source radiance to incident radiance without bandlimiting the frequency content in the output, therefore it encounters the problem of dense sampling in the 6D high dimensional space as discussed earlier.

Our approach essentially reduces sample rate by choosing a low-order output basis (from the BRDF factorization) for the transport matrix, such that when computing bounced illumination transport we only deal with signals of a much lower dimension. This means our interreflection bounces only carry low-frequency transport energy, which is reasonably accurate except for very glossy surfaces. On the other hand, we use non-linear wavelets as a different basis for the input, which efficiently preserves high-frequency content in the visibility as well as the lighting, allowing us to render sharp shadows on the fly.

3.3 Separable BRDF Approximation

3.3.1 *Factorization.* We use the SVD technique to factorize BRDFs [Kautz and McCool 1999]. We sample the BRDF and construct a BRDF matrix $\mathbf{M}$, the columns and rows of which are sampled incident directions and view directions respectively. We process the R, G, and B color channels separately. Applying SVD on $\mathbf{M}$ gives $\mathbf{M} = \mathbf{U} \times \mathbf{S} \times \mathbf{V}'$ where $'$ denotes a transpose. It can be written conveniently as:

$$\mathbf{M} = \sum_{k=1}^{K} \sigma_k \, \mathbf{u}_k \, \mathbf{v}'_k$$

where σ_k denotes the singular values and are the diagonal elements of $\mathbf{S}$; $\mathbf{u}_k$ and $\mathbf{v}_k$ denote the column vectors of $\mathbf{U}$ and $\mathbf{V}$. We factor σ_k and define $\sqrt{\sigma_k}\,\mathbf{u}_k$ and $\sqrt{\sigma_k}\,\mathbf{v}_k$ to be the BRDF light maps and view maps respectively. Since $\mathbf{u}_k$ and $\mathbf{v}_k$ are sets of orthonormal vectors, truncating the sum with the largest K singular values results in an optimal approximation of $\mathbf{M}$ in the Frobenius norm.

We usually apply equal sampling resolution on both incident and view directions. In this case, due to Helmholtz reciprocity of a physical BRDF, $\mathbf{M}$ is real symmetric. According to the Spectral Theorem, $\mathbf{M}$ is diagonalizable such that $\mathbf{M} = \mathbf{Q} \times \mathbf{D} \times \mathbf{Q}'$. The diagonal elements of $\mathbf{D}$ are eigenvalues of $\mathbf{M}$, and the column vectors of $\mathbf{Q}$ are the corresponding normalized eigenvectors. They relate to the SVD of $\mathbf{M}$ by: $\mathbf{U} = \mathbf{V} = \mathbf{Q}$, $\mathbf{S} = \mathbf{D}^2$. This implies that the k-th light map and view map will be the same if the k-th eigenvalue is positive; and they will differ only by a sign if the eigenvalue is negative.

Obviously a diffuse BRDF can be trivially factored into a single ($K = 1$) constant term, as the corresponding matrix $\mathbf{M}$ has only one non-zero singular value. In some cases, a single term approximation is also able to reproduce a reasonable amount of specularity, and can be computed efficiently using normalized decomposition [Kautz and McCool 1999]. For highly specular BRDFs, a significant number of approximation terms are necessary, since many singular values tend to become equally large as $\mathbf{M}$ approaches a diagonal matrix. With a few low-order terms, however, the approximation is accurate for low-frequency specularities. We have experimented with $K = 4$ for all our BRDFs and found this to be visually satisfactory for materials with moderate glossiness. In Section 5.2, we discuss the BRDF approximation error by varying the number K.

3.3.2 *Parametrization.* As we have shown in Section 3.1, our PRT formulation requires us to parameterize the BRDF in incident and view directions. Highly spec-

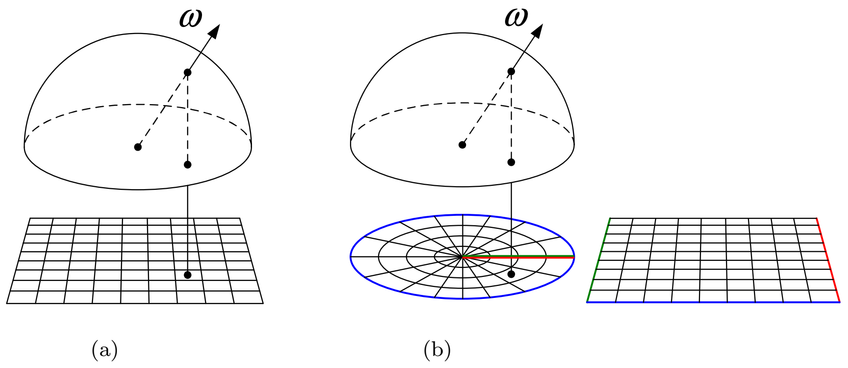


Fig. 3. (a) shows the projection of ω onto a uniform grid texture map; (b) shows the projection of ω with polar coordinates and the corresponding unwrapped texture map.

ular BRDFs are better aproximated by other parameterizations (e.g. the half-angle parametrization) [Rusinkiewicz 1998], but unfortunately we can not take advantage of them in our current framework. We store light maps and view maps from the separated BRDF as 2D textures. To access these textures, we use a parametrization that projects a hemispherical direction ω onto a uniform grid texture map as shown in Figure 3(a):

$$\mathrm{x} = (\omega \cdot \mathbf{s} + 1)/2 \qquad \mathrm{y} = (\omega \cdot \mathbf{t} + 1)/2 \tag{10}$$

where x and y are texture coordinates; $\mathbf{s}$ and $\mathbf{t}$ are tangent and binormal vectors of the local coordinate system, which are derived from the normal $\mathbf{n}$. We use stereographic projection of the unit sphere to parameterize the space of normals and derive a formula for $\mathbf{s}$ and $\mathbf{t}$ as:

$$\mathbf{s} = \text{normalize}\left[(n_y+1)^2 + n_z^2 - n_x^2,\ -2\,(n_y+1)\,n_x,\ -2\,n_x\,n_z\right] \qquad \mathbf{t} = \mathbf{n} \times \mathbf{s}$$

where $\mathbf{n} = [n_x, n_y, n_z]$ is a given normal. We favor this formula because commonly $\mathbf{s}$ and $\mathbf{t}$ are generated with two discontinuities (such as at $\mathbf{n} = [0,1,0]$ and $[0,-1,0]$); while this formula produces only one discontinuity at $\mathbf{n} = [0,-1,0]$.

Although we typically use the uniform grid projection method, we found that for some BRDFs such as the Ashikhmin-Shirley anisotropic BRDF [Ashikhmin and Shirley 2000], a significantly better single term approximation is achieved by projection with the polar coordinates, as shown in Figure 3(b). The formula is:

$$\mathrm{x} = \tan^{-1}(\omega \cdot \mathbf{s},\ \omega \cdot \mathbf{t}) \qquad \mathrm{y} = \sqrt{1 - \omega \cdot \mathbf{n}} \tag{11}$$

4. IMPLEMENTATION

In this section, we describe our implementation details and discuss ways to accelerate precomputation and rendering using modern graphics hardware.

To construct the BRDF matrix, we sample both incident and view directions on a unit disk embedded in a 2D texture map of 48 × 48 resolution. The memory required to store the tabulated BRDF is approximately 37.5 MB. Sampling at a higher resolution is possible, but requires more memory and computation time, while providing little improvement in accuracy (Section 5.2). The cosine term in the transport function (Eq 3) could be built into the definition of BRDF, as done by Liu et al. [2004]. This may result in a better approximation due to the attenuation of high BRDF values prior to factorization. Our implementation leaves the cosine term out, which enables us to preserve the symmetry of the BRDF.

4.1 Precomputation – Direct Illumination

In the initial pass we precompute K direct transport vectors T_k per vertex, forming a K-row transport matrix. Distant lighting is parameterized using a $6 \times 64 \times 64$ cubical environment map. Incident and view directions are expressed in the same global coordinate system as the lighting. For proper anti-aliasing, we start by rasterizing a higher resolution $6 \times 128 \times 128$ visibility cubemap at each vertex of the original scene model. A simplified scene model is used for visibility sampling in order to accelerate rasterization speed. We then use the direction vector of each cubemap pixel to index into the BRDF light maps, and evaluate the transport functions according to Eq 3. The result is next 2×2 downsampled to $6 \times 64 \times 64$ resolution.

4.1.1 *Wavelet Transform.* We project each cubemap face (64×64) onto a 2D Haar wavelet basis. In the case of a K-term BRDF approximation, this requires $6 \times K$ wavelet transforms. For non-linear approximation, we gather the wavelet coefficients on all 6 cubefaces and apply a linear-time algorithm to select the first N_l coefficients according to their magnitudes. Due to the orthonormality of the basis, keeping the largest coefficients (in magnitude) results in an optimal approximation in L2-norm. The K transport vectors at each vertex are approximated individually. In our experiments we found $N_l = 128$ (0.52% of the uncompressed data) achieves reasonable rendering fidelity. In Section 5.2 we illustrate the wavelet approximation error by varying N_l.

4.1.2 *Quantization and Storage.* To further reduce the precomputed data size and improve rendering performance, we uniformly quantize the wavelet coefficients. The R, G and B values are each quantized to an 8-bit signed integer. Since the wavelet approximation is non-linear, we store coefficients together with their indices, which requires an additional 15 bits to represent. For data storage efficiency, we partition the range of the index ($0 \sim 24,576$) into 96 blocks, each containing 256 continuous index values. Essentially we are splitting the index into two parts: one 8-bit block index and one 8-bit block offset. The block offset is packed together with the three color channels into a 4-byte field; and the 8-bit block indices are stored separately. We have also experimented with using more bits for quantization to provide higher accuracy. However, we found the 8-bit quantization scheme is ideal for data formats currently available on graphics hardware, and thus significantly accelerates our GPU implementation of the rendering algorithm.

4.1.3 *Hardware Acceleration.* The precomputation of direct illumination is well suited for acceleration on graphics hardware. To do this, we render the visibility cubemap into a single channel, single byte OpenGL pbuffer, and bind it as a texture for a shader that computes the transfer function based on the visibility and factored BRDF terms. This transfer function is rendered into a 16-bit floating point render target, and is bound again to do the 2×2 downsampling on the card, which we accelerate by directly taking use of 16-bit floating point hardware interpolation. We then only read data off of the card in $6\times64\times64$ blocks of memory, since we compute all six cubefaces for each vertex at once. To reduce memory bandwidth cost for data transfer between CPU and GPU, we store the simplified visibility model (for rasterization) on card as a vertex buffer object. The cubemap direction normals and factored BRDF terms are stored as 16-bit floating point textures. With the GPU implementation, we have achieved approximately $2\times$ speedup in precomputation time over an optimized CPU implementation.

4.2 Precomputation – Global Illumination

Interreflections are computed using a multi-pass method. Each interreflection pass gathers the transport vectors computed in the previous pass to produce a new set of transport vectors carrying bounced illumination. Since this step involves managing a big dataset, our implementation reflects several decisions in an effort to reduce memory consumption as well as precomputation time. First, transport vectors from the previous pass must be loaded entirely in memory, because they are accessed in a random pattern by the gathering process. In order to reduce memory usage, we decided to keep all transport data in their compressed and quantized form in memory, which incurs a slight overhead by decoding on the fly. Second, to prevent the intermediate data size from expanding, we decided to truncate the wavelet coefficients after each pass. Repeating this process, however, can lead to accumulated non-linear errors due to the early truncation of certain high-frequency coefficients. We could choose to keep an increasingly higher number of coefficients for each pass, but this would be a tradeoff between memory usage and computation accuracy. Third, because transport vectors represent radiance values as linearly influenced by the lighting (which is unknown at precomputation step), we cannot determine which surface location has the largest unshot radiance energy. This means we are not able to take advantage of more efficient techniques such as *progressive radiosity*. Finally, the gathering process does not lend itself to current graphics hardware, and we resort to a CPU implementation.

For numerical integration, the gathering process uses an approach similar to the *hemicube* approximation for computing form factors in radiosity algorithms. But instead of using a locally oriented hemicube, we construct a cubemap in the global coordinate system centered at each vertex. This helps reduce sampling artifacts due to spatial variance. We start by assigning a unique false color to each vertex in the scene model, which allows us to correctly identify any vertex by reading its color. We then rasterize a low resolution $6 \times 32 \times 32$ cubemap centered at each vertex $\mathbf{x}$, using the false colored scene model and OpenGL flat shading. Notice that this cubemap resolution is independent of the lighting resolution $(6\times64\times64)$, and can vary according to the designed accuracy. Using a very low cubemap resolution, however, could cause aliasing (banding) artifacts as shown in Figure 11. In practice, we do not observe any obvious artifacts by using the suggested resolution $(6 \times 32\times32)$. This low resolution suffices because our interreflection computations only carry low-frequency transport energy due to the bandlimiting effects of low-order BRDF approximation. Using a higher resolution will increase precomputation time considerably but provide little improvements in quality. Alternatively, one may apply Monte Carlo techniques for numerical simulation of bounce illumination;

however, we found the suggested method easy to implement and visually convincing.

In the next step, we compute bounced transport vectors according to Eq 8, where ω is simply the direction vector of each cubemap pixel, and $\widetilde{\mathbf{x}}$ is indicated by the color of that pixel. The transport vectors of $\widetilde{\mathbf{x}}$ from the previous pass are retrieved from memory, scaled by the BRDF view map values and the complement transport function values (Eq 8), and finally accumulated to a buffer containing the bounced transport vectors of $\mathbf{x}$.

4.3 Rendering

At run-time, we sample the environment map dynamically onto a $6 \times 64 \times 64$ cubemap, perform a fast 2D Haar wavelet transform and then uniformly quantize the wavelet coefficients to signed 16-bit intergers. Quantized coefficients with absolute value below a certain threshold are discarded. In order to reduce temporal artifacts (flickering) when rotating the light, we choose to keep many more lighting coefficients than we keep transport vector coefficients. As a result, our rendering speed depends primarily on the size of precomputed data and only varies slightly as we apply different lighting environments.

To relight the scene, we perform a multiplication of the sparse transport matrix with the sparse light vector, and produce the light-dependent K-vector for each vertex. The view direction at each vertex is used to index into BRDF view maps and produce the view-dependent K-vector. Vertex color is computed as the dot product of the two K-vectors. Notice that the light-dependent K-vector is recomputed every time the lighting changes, and the computation is more expensive than when only the viewpoint is changing.

4.3.1 *Hardware Acceleration.* We have accelerated the entire rendering pipeline on modern graphics hardware. We take advantages of several features available in the NVIDIA GeForce 6 series graphics cards, including 64-bit floating point texture filtering and blending, non-power of two textures, vertex/pixel buffer objects, and vertex texture fetch.

As discussed in Section 4.1.2, we use the 8-bit per channel quantization scheme for precomputed transport vectors and upload them as unsigned byte textures to the graphics card. Each transport vector contains 128 elements (non-linear wavelet coefficients) laid out as a 16×8 block. Their indices are also stored on card, with the block offsets stored as the fourth channel of the transport vector texture, and the block indices as a separate single channel floating-point texture. When lighting changes, we sample and wavelet transform the environment map dynamically on the CPU. The transformed coefficients, which forms the light vector, are laid out as a 96×256 floating point texture and uploaded to GPU with 16 bits per color channel. Unlike the CPU implementation, we do not truncate the lighting coefficients, thus only the transport vectors are sparse while the light vector is dense.

Figure 4 shows the computation diagram of our GPU-based rendering algorithm. After the lighting coefficients are uploaded, we draw a quadrilateral the size of our domain ($\mathbf{P}$ in the diagram). In a fragment shader, we perform the per-element multiplication portion of the inner product between the transport vectors and the light vector. To do so, the block index and offset of each element are combined as a 2D texture coordinate to access the proper element in the lighting texture for

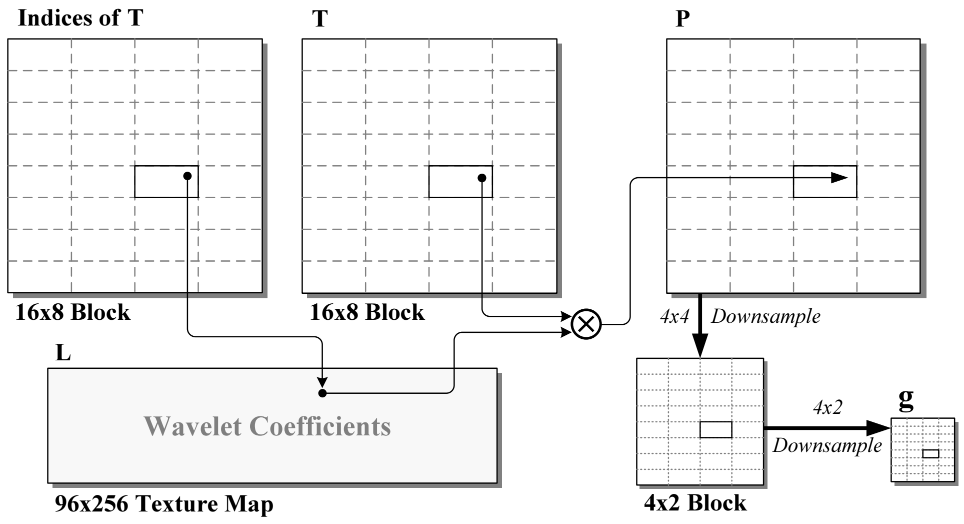


Fig. 4. Computation diagram for our GPU-based rendering algorithm. The transport vectors **T**, their indices, and the intermediate computation buffer **P** are stored in texture memory on card, which are partitioned into blocks of size 16×8 (indicated by the dotted line). Each block stores the sparse transport vector (containing 128 elements) for a single vertex. The inner product of the light vector **L** and the transport vectors **T** is calculated by first performing the per-element multiplication and storing the results in **P**. This involves one step of texture indirection using the indices of the transport vectors. **P** is then downsampled twice to sum up all the 128 elements in a block and produce the final result **g**.

multiplication. The results are rendered into an intermediate computation buffer **P**. We then repeatedly downsample this buffer taking use of hardware support for bilinear interpolation, and accumulate **P** into one final texture which holds the light-dependent color **g** for each vertex.

To fully exploit the texture cache, we choose the dimension of our domain to be as square as possible based on the block size (16×8) and the total number of vertices. As shown in the diagram, the downsampling of **P** is performed in two passes, first with a 4×4 filter, then with a 4×2 filter. This two-pass algorithm was chosen to match our current graphics hardware, and is proved to be faster than one pass by our experiments. Finally the result **g** is to be bound as either a texture or a vertex array for view-dependent rendering. Tradeoffs between these two options will be explained in Section 5.3.

When the viewpoint changes, we calculate the dot product of the light-dependent K-vector and the view-dependent K-vector in a fragment shader to produce a per-pixel color. To do so, we program a vertex shader to extract the light-dependent K-vector from a texture or a vertex attribute, depending on whether **g** is bound as a texture or a vertex array. The vertex shader also computes the view direction for each vertex, which is later accessed by the fragment shader to index into all BRDF view maps and extract the view-dependent K-vector. Finally the pixel color is computed by a multiplication of the two K-vectors.

The indexing of the view direction into BRDF textures is computed according to Eq 10 (see also Figure 3(a)). Currently our GPU implementation is limited to this

Table I. Precomputation profiles for our test cases. Each column lists the following in order: the number of vertices and triangles in each model; the number of vertices in the diffuse floor and the visibility model; 1-term and 4-term precomputation time and storage size for the direct illumination pass, and for one bounce interreflection pass (storage size generally remains the same).

	Bird	Bunny	Head	Buddha	Lucy
Number of vertices	25 K	36 K	49 K	56 K	102 K
Number of triangles	50 K	70 K	98 K	113 K	203 K
Size of floor	31 K	31 K	31 K	31 K	31 K
Size of visibility model	7.7 K	7.6 K	8.3 K	20 K	30K
Direct 1-term pre. time	5.3 min	5.6 min	7.7 min	11 min	20 min
Direct 4-term pre. time	11 min	14 min	20 min	24 min	44 min
Direct 1-term size	35 MB	36 MB	50 MB	54 MB	83 MB
Direct 4-term size	82 MB	100 MB	142 MB	160 MB	273 MB
Bounce 1-term pre. time	20 min	22 min	23 min	35 min	56 min
Bounce 4-term pre. time	39 min	46 min	51 min	109 min	122 min

Table II. Rendering frame rates (GPU/CPU) for changing viewpoint and changing light with 1-term and 4-term BRDF approximations. These experiments were done using the *St. Peter's Basilica* light probe image. Due to limitations imposed by our hardware platform, we were not able to complete some of the tests on the GPU.

	Bird	Bunny	Head	Buddha	Lucy
1-term view	230/79 fps	226/79 fps	255/56 fps	210/47 fps	173/35 fps
4-term view	123/67 fps	97/62 fps	160/46 fps	90/38 fps	101/26 fps
1-term light	31/13 fps	31/12 fps	25/10 fps	7.6/8.7 fps	–/5.9 fps
4-term light	–/5.8 fps	–/4.8 fps	–/3.6 fps	–/2.9 fps	–/1.8 fps

uniform grid projection method, because using the polar coordinates method will result in some triangles whose texture coordinates cross the periodic edge, causing texture interpolation artifacts. In Section 6, we explain a possible implementation of the polar coordinates method given the proposed future hardware specification.

5. RESULTS AND DISCUSSION

We present results using the following BRDFs:

—A Phong BRDF with a red diffuse color and $s = 50$ specular exponent

—Ashikhmin-Shirley (AS) anisotropic BRDF [Ashikhmin and Shirley 2000] with the following parameters: $k_d = 0.5$, $k_s = 0.75$, $n_u = 10$, $n_v = 100$

—A clay and a dark skin BRDF, using the Lafortune model [Lafortune et al. 1997]

Figure 5 shows the approximation results for the selected BRDFs. In each image we show the light map textures of the first 8 BRDF terms, along with the Bunny model rendered using the first 4 terms. The view map textures are not shown, as they share the same absolute values with the light map textures due to the symmetry of our BRDF matrix. We also plot the decay of cosine-weighted RMS errors as we increase the number of BRDF terms as well as the texture map resolution. These examples demonstrate that a 4-term approximation is visually adequate for materials with moderate specularities.

Figure 6 shows models rendered with diffuse and glossy BRDFs, and compares the rendering qualities of direct illumination, 1-bounce and 2-bounce interreflections. Precomputation time for each interreflection bounce is approximately the same,

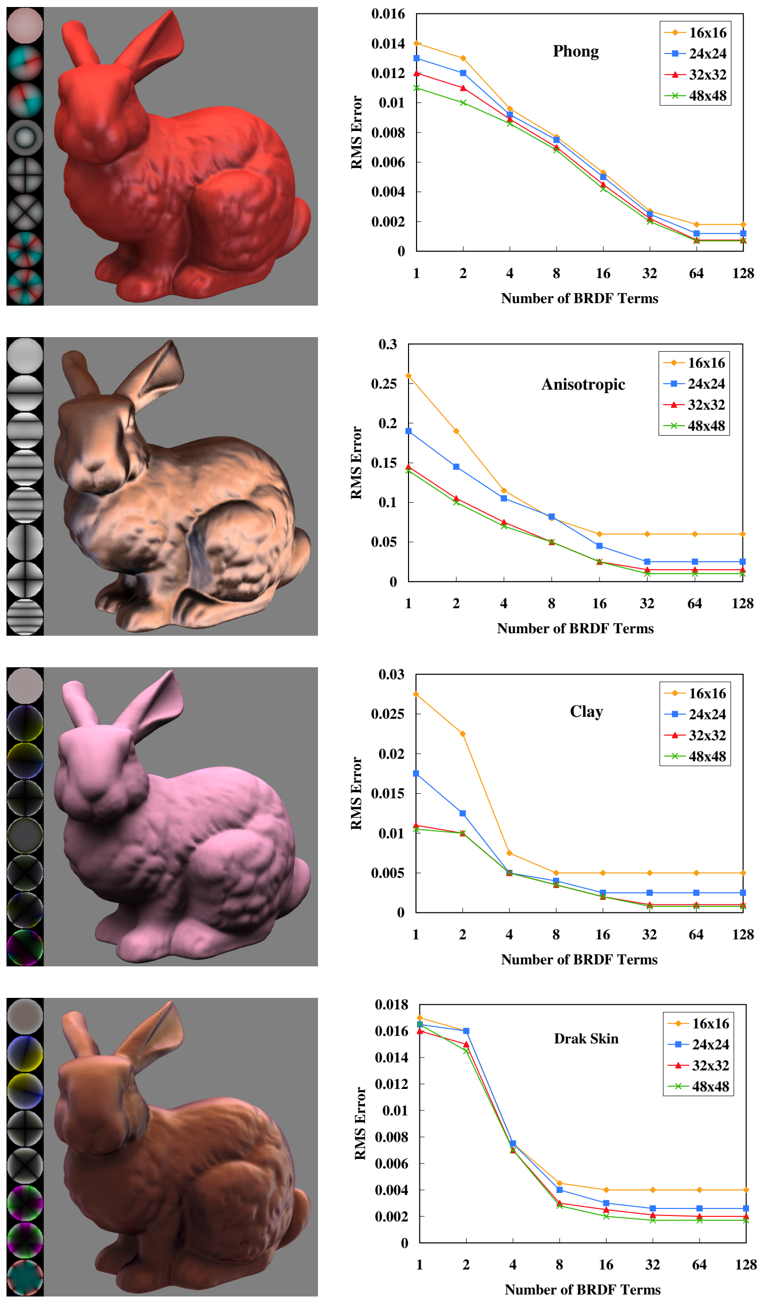


Fig. 5. BRDF approximation results. Each image on the left shows the light map textures of the first 8 BRDF terms, along with the Bunny model rendered with the first 4 terms using different environment lighting. Colors in the textures are displayed in absolute values. The graphs on the right plot the decay of cosine-weighted RMS errors as we increase the number of BRDF terms, with each curve denoting a different texture map resolution.

and rendering performance is insensitive to the additional precomputation. Note that computing one bounce already provides a good approximation to the global illumination solution. Further bounces beyond 2 provide little change overall. The clear definition of shadows on the diffuse floor demonstrates the ability of our technique to handle all-frequency lighting effects. In Figure 7 we provide two other examples of enhanced image realism by considering diffuse to glossy interreflections and vice versa. The exhibited effects, such as color bleeding and caustics, are clearly not present in the direct illumination rendering.

5.1 Performance

Our test results were acquired on an Intel Pentium 4 2.4 GHz computer with 1 GB memory and an NVIDIA GeForce 6800 graphics card. We compile our programs using the Intel Compiler 8.0, and compile our shaders using Cg 1.3 and NV40 profiles. We use the OpenGL vertex buffer objects (VBO) extension to reduce memory bandwidth cost for data transfer between the CPU and GPU. To demonstrate the ability of our technique to handle all-frequency shadows, we render a diffuse floor under each model; the reported precomputation time and rendering performance include the diffuse floor.

Table I summarizes the precomputation profiles for a number of models we experimented. All models are precomputed with up to $K = 4$ BRDF approximation terms. The direct illumination pass is computed with our GPU implementation, and the interreflection pass is computed on CPU. Since visibility sampling is expensive, we use a simplified scene model for rasterization. This approximation has no visible effect on any of our test scenes. Our non-linear wavelet approximation keeps up to $N_l = 128$ largest wavelet coefficients per transport vector. For a 4-term BRDF approximation and a 100,000-vertex model, the total storage size is approximately 240 MB with the proposed 8-bit quantization scheme. This is roughly 30% of the data size without quantization. By using graphics hardware, we have achieved 2 times speed up in precomputation time compared to a CPU implementation which uses the graphics hardware only for visibility sampling.

The precomputation for interreflections requires us to use the original model, instead of a simplified model, for visibility sampling. This is because the gathering process has to correctly identify the contributing vertices from the original model. We currently do not have a GPU implementation for this step. Precomputation with our CPU implementation is considerably slower than the direct illumination pass, and each interreflection pass takes approximately the same amount of time. After each pass the gathered non-linear transport vectors are truncated again to $N_l = 128$ largest wavelet coefficients, therefore the storage size does not grow with the additional computation, and the rendering performance remains unmodified.

Table II summarizes the rendering frames rates. Due to texture memory limits, we cannot currently perform lighting updates on the GPU for large models or for a 4-term BRDF approximation. This is primarily because we have to maintain an expensive intermediate computation buffer $\mathbf{P}$ (see Figure 4) due to the lack of an accumulation mechanism on GPU. However, for smaller models, the GPU implementation has shown considerable performance gain over a reasonably optimized CPU implementation.

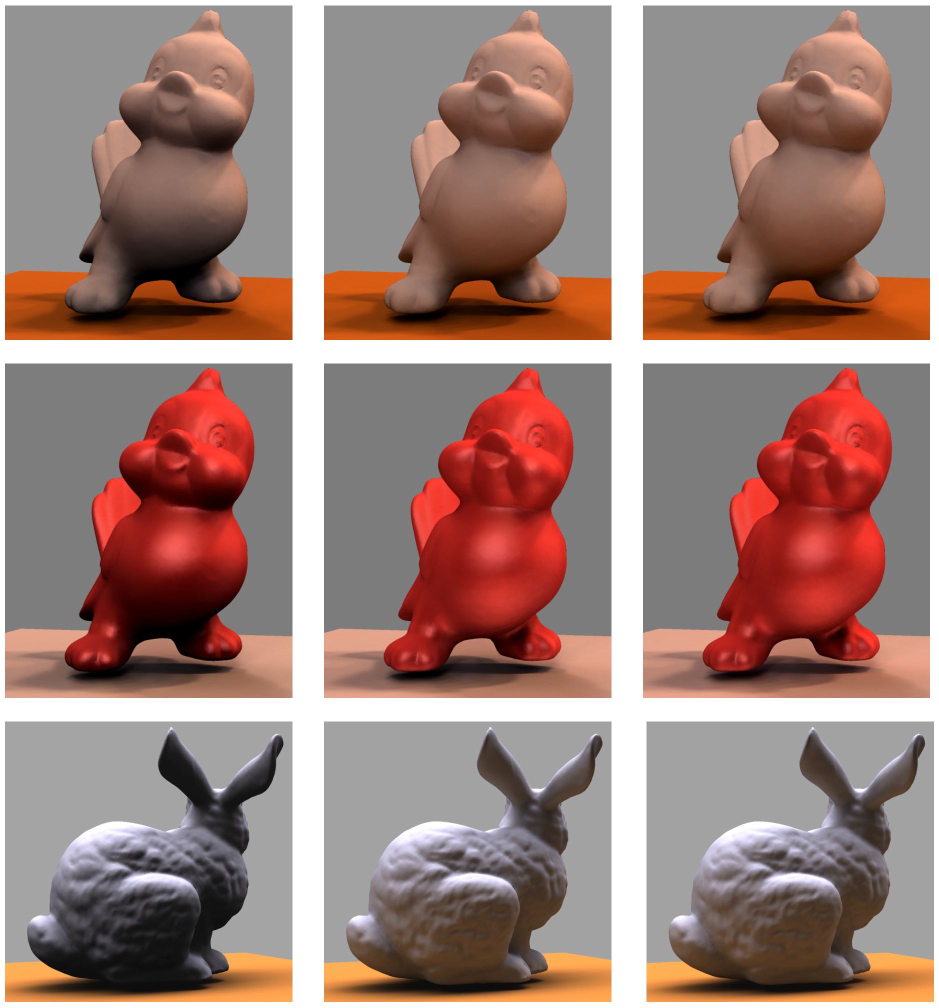

(a) direct illumination (b) 1-bounce interreflection (c) 2-bounce interreflections

Fig. 6. Simulation of interreflection effects. This example shows the Bird model rendered with a diffuse and a Phong BRDF, and the Bunny model rendered with a diffuse BRDF. Note the clear definition of shadows on the diffuse floor and the considerable added realism brought by adding interreflections. Rendering speed is insensitive to the inclusion of interreflections. Even one bounce does well at approximating the global illumination solution. Further bounces beyond 2 provide little change overall.

5.2 Accuracy

5.2.1 *BRDF Approximation.* To analyze the accuracy of our BRDF approximation, we chose 100,000 uniformly distributed samples over both the incident and view directions and evaluate the approximated BRDF values on these directions. We compute the RMS error of these values with the analytic BRDF, weighted by the incident cosine term. In Figure 5, we plot the cosine-weighted RMS error as a function of the number of approximation terms. Each curve represents a different

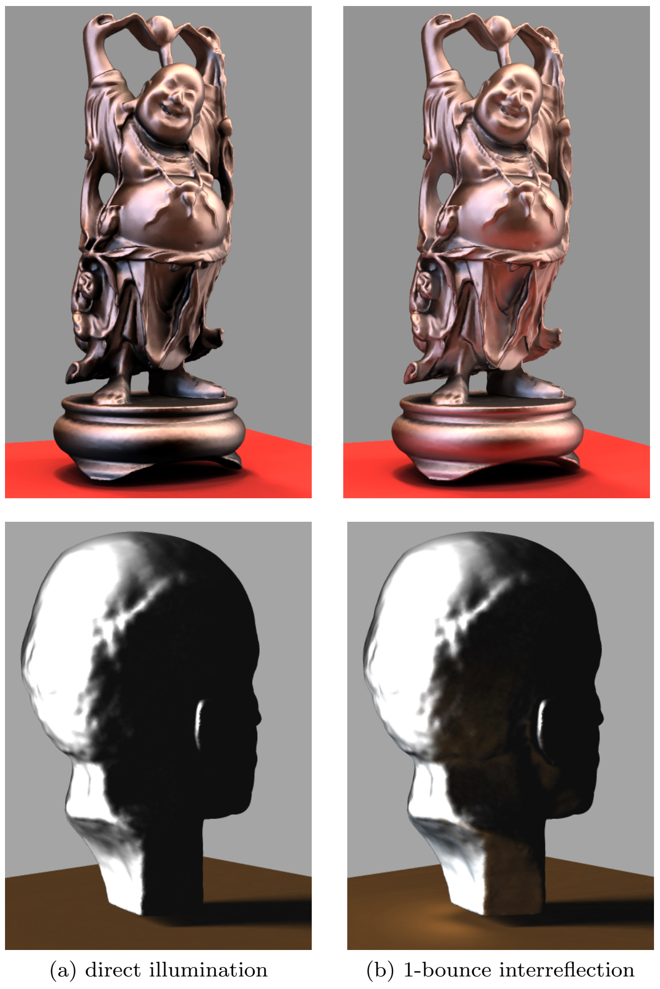


Fig. 7. The top row shows the Buddha model rendered with the AS anisotropic BRDF. Note the color bleeding from the red diffuse floor to the glossy model. The bottom row shows the head model rendered with the same BRDF. Note the caustics on the floor due to the reflection of lights from curved glossy surfaces. These effects are not present in the direct illumination renderings.

BRDF texture map resolution. The non-zero asymptotic error is due to bilinear interpolation of finite resolution texture maps. Notice that in all examples, the error curve of 48×48 texture resolution appears to converge. Sampling at a higher resolution requires considerably more memory and computation time, while providing little improvement in accuracy. Therefore we choose 48×48 resolution.

It is well-known that RMS error of the BRDF does not directly relate to perceived error in rendering. In Figure 8 we show the Buddha model rendered with different number of terms for the AS anisotropic BRDF. The model is illuminated by a single colored light and we ignore shadowing. The reference image is rendered with the analytic BRDF for comparison. From the fidelity of specular highlights, we conclude that a 4-term approximation is visually adequate. As we increase the number

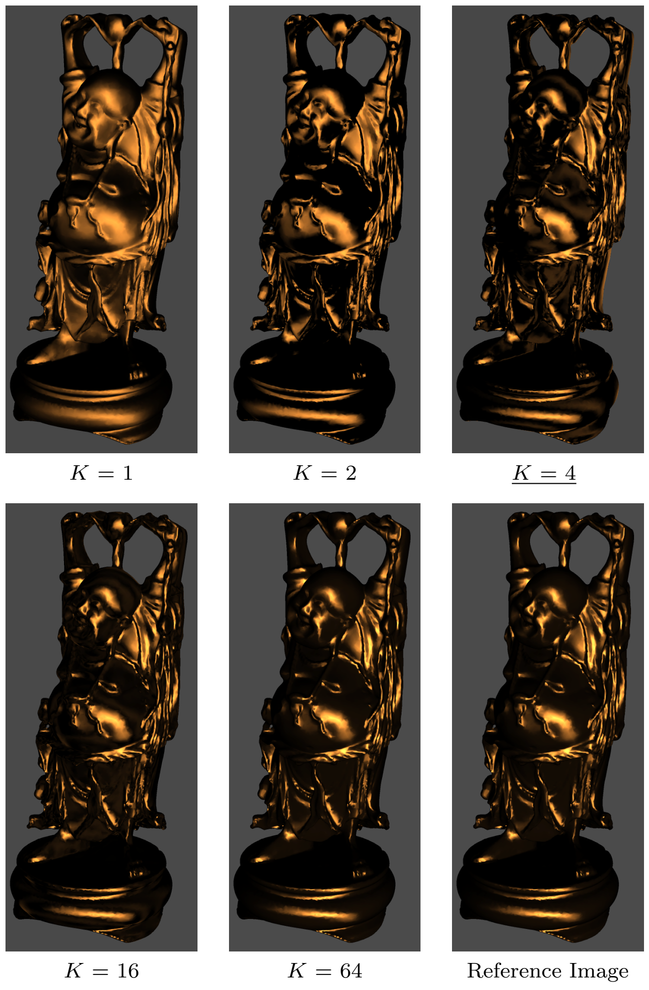


Fig. 8. The Buddha model rendered with the AS anisotropic BRDF and illuminated by a single colored light. We compare fidelity of the specular highlights by increasing the number of BRDF approximation terms K. The reference image is rendered with the analytic BRDF. We typically use $K = 4$, which is qualitatively sufficient. Some observable errors include its slightly darker appearance compared with the reference and noticeable ringing effects on the base of the model.

of terms, the precomputed data size grows linearly and rendering performance decreases linearly. Therefore we choose $K = 4$, which appears to be a good tradeoff between visual accuracy and rendering speed.

When accuracy is not the primary concern, we have found that a single term approximation can often provide reasonable specularities. Figure 9 compares the Lucy model rendered with a diffuse gray BRDF, a 1-term approximation of the AS anisotropic BRDF using projection with polar coordinates (Eq 11), and 4-term approximation of the same BRDF. Note the specular highlights present in the 1-term approximation as compared to the diffuse BRDF rendering.

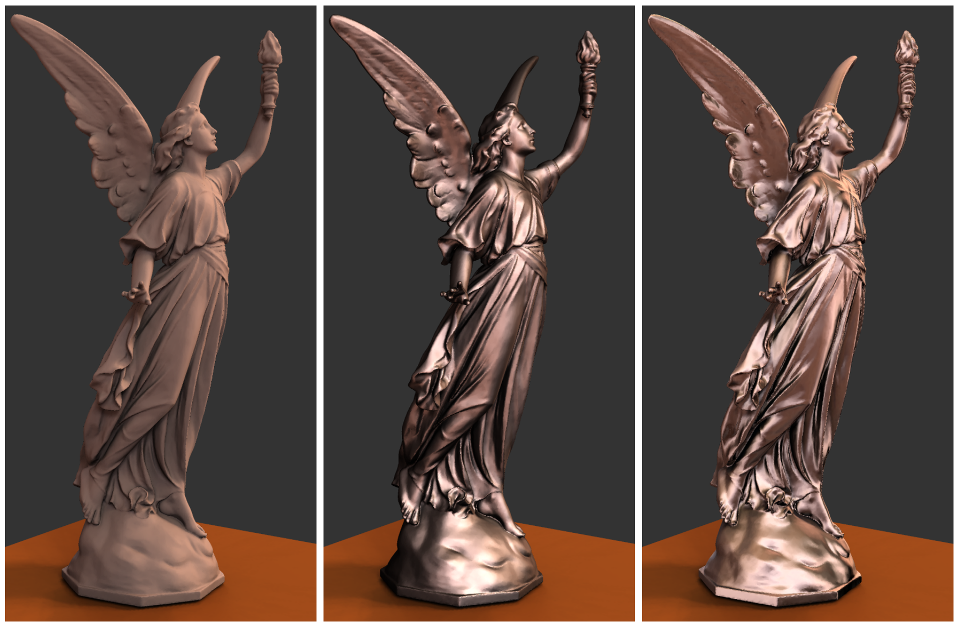


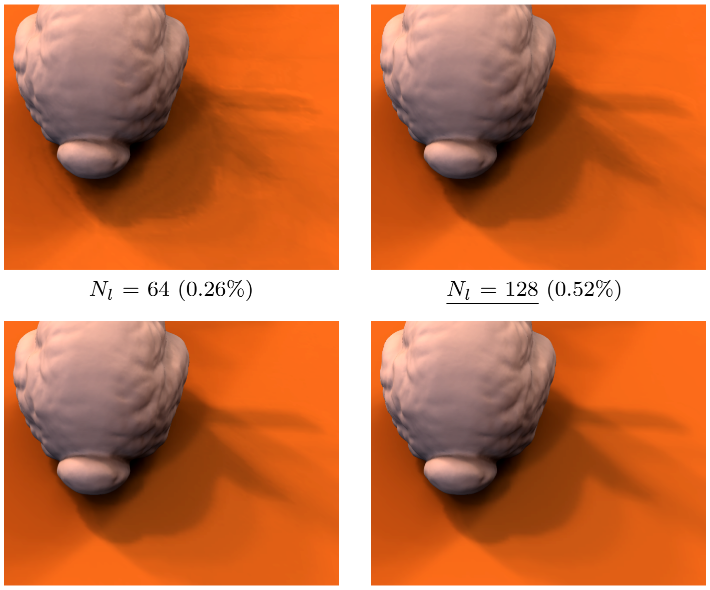
Fig. 9. This example shows the Lucy model rendered with a gray diffuse BRDF, a 1-term approximation of the AS anisotropic BRDF using projection with polar coordinates (Eq 11), and a 4-term approximation. Note that the 1-term approximation has captured reasonable specularities.

5.2.2 *Non-linear Wavelet Approximation.* Figure 10 shows several renderings of the Bunny model with high-frequency shadows on the floor as we vary the number of wavelet approximation terms (N_l). We believe that $N_l = 128$ provides visually satisfactory fidelity of shadows. In the CPU rendering algorithm, we truncate lighting wavelet coefficients, which gives slightly better frame rates. However, our rendering performance and quality is insensitive to the approximation of lighting, since we keep many more wavelet terms for the lighting than for the transport functions. A good reference for approximation of the lighting using non-linear wavelets can be found in [Ng et al. 2003]. Our GPU implementation does not truncate lighting coefficients.

5.2.3 *Interreflections.* Since we use a cubemap to sample interreflections during the gathering process, its resolution has to be sufficient to avoid aliasing, manifested as banding artifacts in the rendering. Figure 11 shows the Bird model rendered with precomputed one bounce interreflection. Note as the cubemap resolution decreases, the banding artifacts become more obvious. We choose $32 \times 32 \times 6$ resolution, which provides a good tradeoff between precomputation time and accuracy.

5.3 The GPU Rendering Algorithm

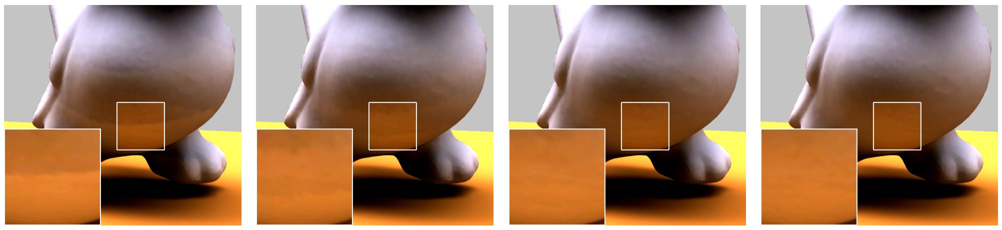
There are several possible choices to make when implementing the rendering algorithm on GPU. This section describes these choices and discusses limitations with each implementation.



$N_l = 64$ (0.26%) $\underline{N_l = 128}$ (0.52%)

$N_l = 256$ (1%) $N_l = 512$ (2.1%)

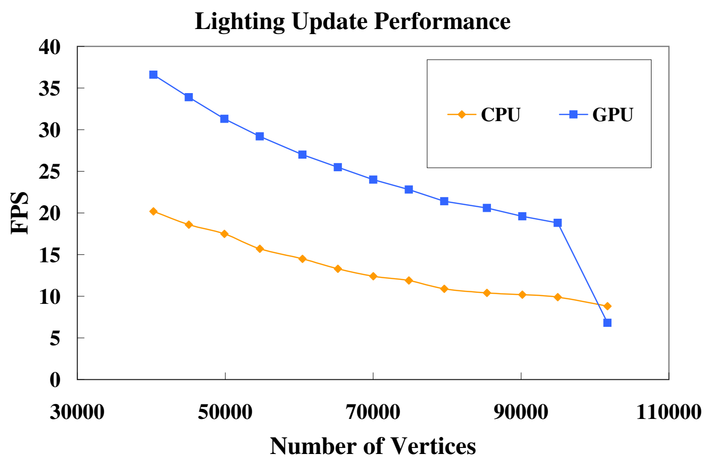
Fig. 10. The Bunny model casting sharp shadows on the diffuse floor. We compare the fidelity of shadows by increasing the number of wavelet approximation terms N_l, with the corresponding percentage of retained coefficients shown in parentheses. We typically use $N_l = 128$ terms, which provides a good tradeoff between visual accuracy and rendering speed.



$8 \times 8 \times 6$ $16 \times 16 \times 6$ $\underline{32 \times 32 \times 6}$ $64 \times 64 \times 6$

Fig. 11. This example compares the accuracy of interreflection precomputation when different cubemap resolutions are applied for the gathering process. Note that a very low sampling rate can cause severe aliasing (banding) artifacts, such as in the highlighted area. The contrast of these images have been increased to emphasize the artifacts. We typically use a $32 \times 32 \times 6$ resolution.

For convenience, we separate the rendering pipeline into two parts: *lighting update* produces the intermediate buffer $\mathbf{P}$ and downsamples the buffer to $\mathbf{g}$ which contains the inner product of the light vector with precomputed transport vectors; *view interpolation* accesses the data in $\mathbf{g}$ and the BRDF view maps to compute view-dependent colors. Lighting update is the most expensive part and performed every time the lighting changes. To communicate the data in $\mathbf{g}$ with view interpolation, we can choose between two options: 1) *vertex textures*, which binds $\mathbf{g}$ as a texture to be accessed in the vertex shader using vertex texture fetch (VTF); 2) *render-to-*



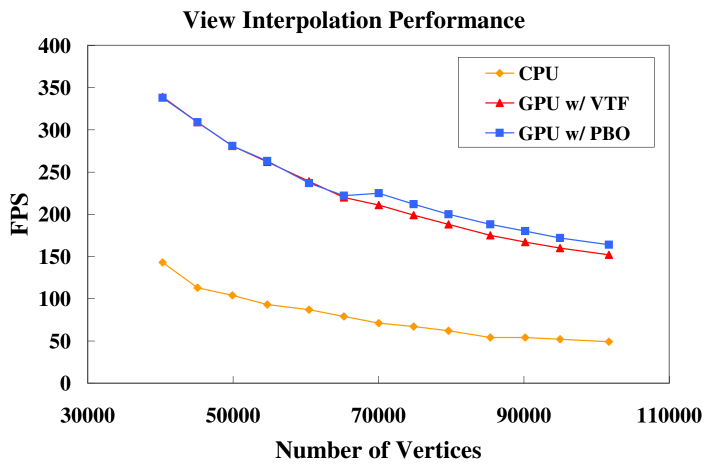
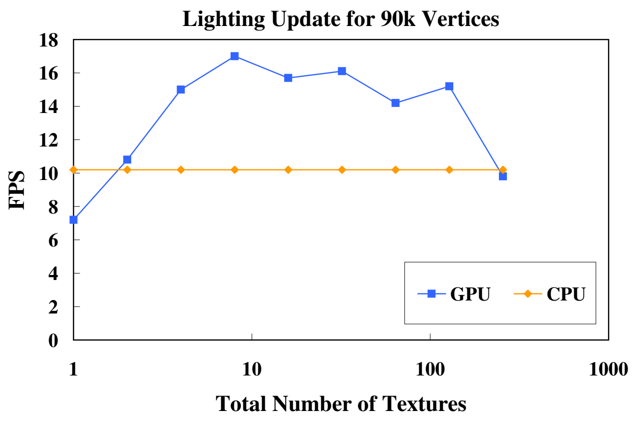


Fig. 12. The two graphs plot the comparison of CPU and GPU performance for the two parts of rendering algorithm: lighting update and view interpolation. These data were taken from the Lucy model, simplified to several levels of detail. VTF denotes a GPU implementation using vertex texture fetch, and PBO denotes using render-to-vertex-array. Both are explained in section 5.3.

vertex-array, which renders **g** into a pixel buffer object (PBO) to be copied over to a vertex buffer object (VBO) and accessed in the vertex shader as a vertex array. Both options are available in the NVIDIA hardware.

Figure 12(a) shows a comparison of the lighting update performance. The GPU implementation clearly exceeds the performance of the CPU until we get to large models, when we suspect the GPU has run out of texture memory. The memory usage on GPU is quite high due to the size of the intermediate buffer **P**, which is floating point and has the same dimension as the entire computation domain. In contrast, the CPU implementation does not need to store **P** because inner products are trivially computed via accumulation.

Figure 12(b) shows the view interpolation performance and compares the two GPU implementations, one using vertex textures (VTF) and one using render-to-



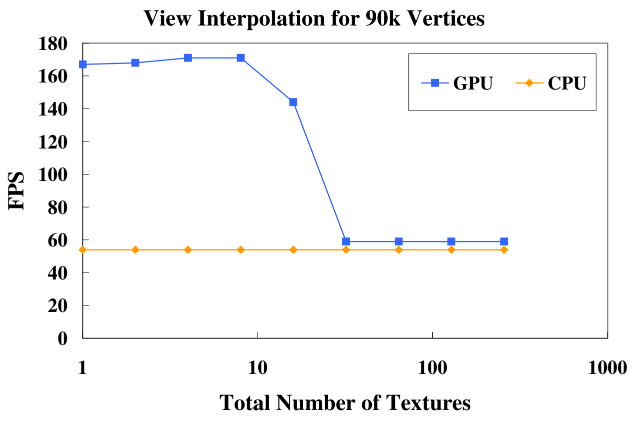


Fig. 13. Comparison of GPU rendering performance when using a different number of textures to subdivide the computational domain. These data were gathered using a 90K-vertex Lucy model.

vertex-array (PBO). The former has the advantage of not having to perform an extra copy between GPU memory (an asynchronous ReadPixels call) necessary in the current hardware, but requires the use of vertex texture fetch, which is a relatively new feature and suffers from known latency problems. Currently VTF requires the texture to be 32-bit float point per channel, twice the amount of storage we would normally use. We do at least 10 instructions within the vertex shader to hide some of the VTF latency known to exist in NV40-class hardware. For models less than 70k vertices, both implementations are virtually identical, suggesting that we have successfully hidden the latency of VTF with enough instructions in this case. However, for models larger than 70k vertices, the PBO implementation performs faster consistently. It is difficult to explain these performance differences without further knowledge of the underlying memory architecture on the GPU.

As the number of vertices in a model increases, the size of the textures required for computing lighting change becomes larger and larger. Because current graphics hardware is not well suited to handling large floating point textures (with either dimension approaching 4k), we divide these textures into several smaller textures.

We then perform all operations on these smaller sized textures and still accumulate the result into the same sized pbuffer for rendering as before. Figure 13(a) shows a log plot of performance as we vary the number of textures used to split up the large input textures. The graph shows that the lighting update performance is optimal when we use approximately 4 to 128 textures.

Contrary to intuition, Figure 13(b) suggests that as we increase the number of textures applied, the view interpolation performance in fact decreases. Since these intermediate textures are not involved in computing view interpolation, we suspect that the performance decrease is probably due to some underlying complications by having more textures on card, even through they do not directly relate to this part of the rendering algorithm.

To maximize performance in both parts of the rendering algorithm, the optimal number of textures for this 90K-vertex model is probably around 8 from the graphs. This results in a texture size of approximately 1700×800. In generally it is hard to predict performance based solely on the texture count or size.

6. CONCLUSION AND FUTURE WORK

We have shown that using non-linear wavelet approximation and separable BRDF approximation enables rendering of glossy objects under all-frequency lighting environments at interactive rates. By including interreflections into precomputation, our technique provides a higher level of realism at no additional cost in rendering performance. We have also shown acceleration of both precomputation and rendering using programmable graphics hardware.

Currently we use only the Haar wavelet approximation for its simplicity. However, compression of transport vectors by Haar wavelets may not be optimal in terms of accuracy and efficiency. In the future we plan to experiment with more sophisticated wavelet filters, in hopes of finding a better wavelet approximation that provides the same accuracy with fewer non-linear terms. To further reduce precomputed data size, we plan to exploit data coherence among neighboring vertices, such as using the clustered principal component analysis (CPCA) presented in [Sloan et al. 2003; Liu et al. 2004]. As an extension to our current precomputation algorithm on graphics hardware, we plan to compute the transfer functions of multiple vertices at the same time. By careful load balancing, we hope to offload an equal amount of work to the GPU as the CPU, allowing them to operate at full speed asynchronously.

Currently on GPU we are limited to using an orthographic projection for calculating the BRDF texture coordinates. We have found that for some BRDFs, a single term approximation is more accurately represented using the polar parametrization (Eq 11). However, this is currently not implementable on the GPU because some triangles have vertices whose texture coordinates cross the boundary of the texture, where the polar coordinates wrap periodically. We would like to be able to interpolate the texture coordinates across this boundary, but we cannot currently control interpolation on the triangle level on modern hardware. The *geometry shader*, which is suggested in the Windows Graphics Foundation (WGF) 1.0 specification, would allow us to implement a correct interpolation, since it enables access to all three vertices of a triangle and to each vertex's attributes.

7. ACKNOWLEDGEMENTS

The authors would like to thank Paul Debevec for his light probe images and NVIDIA for kindly providing the latest graphics hardware. The models used in this paper are courtesy of the Stanford University Computer Graphics Laboratory.

REFERENCES

Agrawala, M., Ramamoorthi, R., Heirich, A., and Moll, L. 2000. Efficient image-based methods for rendering soft shadows. In *Proceedings of SIGGRAPH 2000*. 375–384.

Ashikhmin, M. and Shirley, P. 2000. An anisotropic phong BRDF model. *Journal of Graphics Tools 5,* 2, 25–32.

Ashikhmin, M. and Shirley, P. 2002. Steerable illumination textures. *ACM Trans. Graphics 21,* 1, 1–19.

Blinn, J. and Newell, M. 1976. Texture and reflection in computer generated images. *Commun. ACM 19,* 10, 542–547.

Bolz, J., Farmer, I., Grinspun, E., and Schröder, P. 2003. Sparse matrix solvers on the GPU: conjugate gradients and multigrid. *ACM Trans. Graphics 22,* 3, 917–924.

Cabral, B., Max, N., and Springmeyer, R. 1987. Bidirectional reflection functions from surface bump maps. In *Proceedings of SIGGRAPH 1987*. 273–281.

Chen, W.-C., Bouguet, J.-Y., Chu, M., and Grzeszczuk, R. 2002. Light field mapping: efficient representation and hardware rendering of surface light fields. In *ACM Trans. Graphics*. Vol. 21. 447–456.

Cohen, M. and Wallace, J. 1993. *Radiosity and realistic image synthesis*. Academic Press.

Debevec, P., Hawkins, T., Tchou, C., Duiker, H.-P., Sarokin, W., and Sagar, M. 2000. Acquiring the reflectance field of a human face. In *Proceedings of SIGGRAPH 2000*. 145–156.

DeYoung, J. and Fournier, A. 1997. Properties of tabulated bidirectional reflectance distribution functions. In *Proceedings of Graphics Interface '97*. 47–55.

Dorsey, J., Sillion, F. X., and Greenberg, D. 1991. Design and simulation of opera lighting and projection effects. In *Proceedings of SIGGRAPH 1991*. 41–50.

Fournier, A. 1995. Separating reflection functions for linear radiosity. In *Proceedings of the 6th Eurographics Rendering Workshop*. 383–392.

Gortler, S., Grzeszczuk, R., Szeliski, R., and Cohen, M. F. 1996. The lumigraph. In *Proceedings of SIGGRAPH 1996*. 43–54.

Greene, N. 1986. Environment mapping and other applications of world projections. *IEEE Comput. Graph. Appl. 6,* 11, 21–29.

Heidrich, W., Daubert, K., Kautz, J., and Seidel, H.-P. July 2000. Illuminating micro geometry based on precomputed visibility. In *Proceedings of SIGGRAPH 2000*. 455–464.

Jensen, H. 1996. Global illumination using photon maps. In *Proceedings of the 7th Eurographics Rendering Workshop*. 21–30.

Kajiya, J. 1986. The rendering equation. In *Proceedings of SIGGRAPH 1986*. 143–150.

Kautz, J. and McCool, M. 1999. Interactive rendering with arbitrary BRDFs using separable approximations. In *Proceedings of the 10th Eurographics Rendering Workshop*. 281–292.

Kautz, J. and McCool, M. 2000. Approximation of glossy reflection with prefiltered environment maps. In *Proceedings of Graphics Interface 2000*. 119–126.

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 Rendering Workshop*. 291–296.

Krüger, J. and Westermann, R. 2003. Linear algebra operators for GPU implementation of numerical algorithms. *ACM Trans. Graphics 22,* 3, 908–916.

Lafortune, E., Foo, S.-C., Torrance, K., and Greenberg, D. 1997. Non-linear approximation of reflectance functions. In *Proceedings of SIGGRAPH 1997*. Vol. 31. 117–126.

Lawrence, J., Rusinkiewicz, S., and Ramamoorthi, R. 2004. Efficient BRDF importance sampling using a factored representation. *ACM Trans. Graphics 23,* 3, 496–505.

Lee, D. and Seung, H. 1999. Learning the parts of objects by non-negative matrix factorization. *Nature 401,* 6755, 788–791.

Lehtinen, J. and Kautz, J. 2003. Matrix radiance transfer. In *ACM Symposium on Interactive 3D graphics.* 59–64.

Levoy, M. and Hanrahan, P. 1996. Light field rendering. In *Proceedings of SIGGRAPH 1996.* 31–42.

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.

Malzbender, T., Gelb, D., and Wolters, H. 2001. Polynomial texture maps. In *Proceedings of SIGGRAPH 2001.* 519–528.

McCool, M., Ang, J., and Ahmad, A. 2001. Homomorphic factorization of BRDFs for high-performance rendering. In *Proceedings of SIGGRAPH 2001.* 171–178.

Ng, R., Ramamoorthi, R., and Hanrahan, P. 2003. All-frequency shadows using non-linear wavelet lighting approximation. *ACM Trans. Graphics 22,* 3, 376–381.

Ng, R., Ramamoorthi, R., and Hanrahan, P. 2004. Triple product wavelet integrals for all-frequency relighting. *ACM Trans. Graphics 23,* 3, 477–487.

Peers, P. and Dutré, P. 2003. Wavelet environment matting. In *Proceedings of the 14th Eurographics Symposium on Rendering.* 157–166.

Ramamoorthi, R. and Hanrahan, P. 2001. An efficient representation for irradiance environment maps. In *Proceedings of SIGGRAPH 2001.* 497–500.

Ramamoorthi, R. and Hanrahan, P. 2002. Frequency space environment map rendering. In *ACM Trans. Graphics.* Vol. 21. 517–526.

Rusinkiewicz, S. 1998. A new change of variables for efficient BRDF representation. In *Proceedings of the 9th Eurographics Rendering Workshop.* 11–22.

Sillion, F., Arvo, J., Westin, S., and Greenberg, D. 1991. A global illumination solution for general reflectance distributions. In *Proceedings of SIGGRAPH 1991.* 187–196.

Sloan, P.-P., Hall, J., Hart, J., and Snyder, J. 2003. Clustered principal components for precomputed radiance transfer. *ACM Trans. Graphics 23,* 3, 382–391.

Sloan, P.-P., Kautz, J., and Snyder, J. 2002. Precomputed radiance transfer for real-time rendering in dynamic, low-frequency lighting environments. In *ACM Trans. Graphics.* Vol. 21. 527–536.

Soler, C. and Sillion, F. X. 1998. Fast calculation of soft shadow textures using convolution. In *Proceedings of SIGGRAPH 1998.* 321–332.

Suykens, F., vom Berge, K., Lagae, A., and Dutré, P. 2003. Interactive rendering with bidirectional texture functions. *Eurographics 2003, Computer Graphics Forum 22,* 3.

Veach, E. 1997. Robust monte carlo methods for light transport simulation. Ph.D. thesis.

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.

Westin, S., Arvo, J., and Torrance, K. 1992. Predicting reflectance functions from complex surfaces. In *Proceedings of SIGGRAPH 1992.* 255–264.

Wood, D., Azuma, D., Aldinger, K., Curless, B., Duchamp, T., Salesin, D., and Stuetzle, W. 2000. Surface light fields for 3D photography. In *Proceedings of SIGGRAPH 2000.* 287–296.

Zongker, D., Werner, D., Curless, B., and Salesin, D. 1999. Environment matting and compositing. In *Proceedings of SIGGRAPH 1999.* 205–214.

Received Month Year; revised Month Year; accepted Month Year