Eurographics Symposium on Rendering (2004)
H. W. Jensen, A. Keller (Editors)

All-Frequency Precomputed Radiance Transfer for Glossy Objects

Xinguo Liu[1], Peter-Pike Sloan[2], Heung-Yeung Shum[1], and John Snyder[3]

[1] Microsoft Research Asia [2] Microsoft Corporation [3] Microsoft Research

Abstract

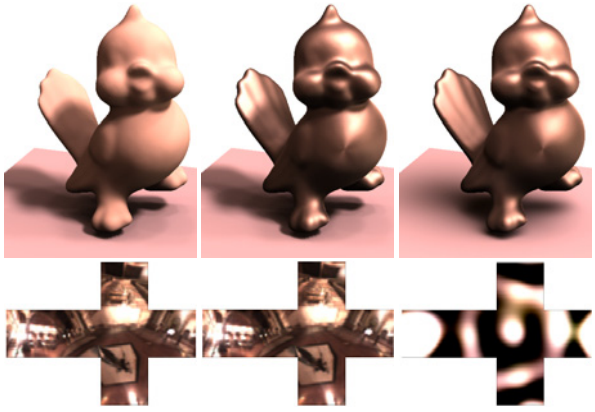
We introduce a method based on precomputed radiance transfer (PRT) that allows interactive rendering of glossy surfaces and includes shadowing effects from dynamic, "all-frequency" lighting. Specifically, source lighting is represented by a cube map at resolution $n_L = 6 \times 32 \times 32$. We present a novel PRT formulation which factors glossy BRDFs into purely view-dependent and light-dependent parts, achieving reasonable accuracy with only m=10 dimensional factors. We then tabulate an $m \times n_L$ transfer matrix at each surface vertex as a preprocess, representing the object's response to this lighting. Because this surface signal is so high-dimensional, reducing m is crucial for making practical both the preprocessing and run-time. To compress the transfer matrices, we divide the cube map into 24 lighting segments and apply the Haar wavelet basis in each segment to provide sensible quantization. We also apply clustered principal component analysis (CPCA) to each PRT segment to approximate it as a linear combination of a few (n=16) representative transfer matrices within a small set of clusters over the surface. This exploits spatial coherence to compress very effectively. Most important, it maintains fast rendering rates with 2-3 orders of magnitude more lighting coefficients than previous methods, which increases accuracy and avoids temporal artifacts in high-frequency lighting environments. We demonstrate interactive performance (1-5Hz) on models having up to 50,000 vertices.

Categories and Subject Descriptors (according to ACM CCS): I.3.3 [Computer Graphics]: Picture/Image Generation

1. Introduction

Fast rendering of global light transport effects from realistic lighting environments is a difficult problem. To solve it, we apply precomputed radiance transfer (PRT) [SKS02]. The basic idea of PRT is to record over sample points on a surface a *transfer matrix* that converts source into exit radiance, and incorporates effects like shadowing, inter-reflection, and subsurface scattering from one part of the object onto another. In other words, PRT tabulates the objects's linear response to source lighting. The method can quickly render global illumination effects from distant, environmental lighting which would be too slow for on-the-fly ray tracing, and require too many rendering passes on graphics hardware.

Our method addresses limitations of previous PRT methods, as shown in Figure 1. [SKS02, KSS02, SHHS03] are limited to low-frequency lighting, producing only soft shadows as seen on the bird's tail and the ground in Figure 1(c). Instead of using a low-order spherical harmonic basis com-



(a) all-freq. diffuse [NRH03] (b) all-freq. glossy our method (c) low-freq. glossy [SHHS03]

Figure 1: *Comparison of PRT methods.*

prising only 25 lighting coefficients, we use a cube map with many more ($n_L = 6 \times 32 \times 32 = 6{,}144$) coefficients. This provides soft shadows from area lighting without precluding sharper shadows from small lights. [NRH03] handles higher-frequency lighting but uses only single-row (scalar output) transfer matrices by limiting to diffuse objects or glossy objects seen from a fixed view. We handle glossy objects with an unconstrained view, and retain more of the lighting energy, which avoids temporal artifacts when the selection of lighting coefficients to truncate changes in the presence of dynamic lighting.

To make our method practical, we combine three ideas from previous work: BRDF factorization [KM99], "all-frequency" PRT using Haar wavelets [NRH03], and CPCA for PRT compression [KL97, SHHS03].

We factor the BRDF into the sum of products of m functions depending only on light direction with m functions only of view direction, as in [KM99]. Unlike past BRDF factorization methods, we account for shadows rather than assuming that source lighting arrives entirely unoccluded. This factoring yields PRT matrices with m rows that are specialized to the particular object's surface reflectance. With a small m (10), we obtain accuracy that would require many more coefficients using unspecialized bases such as spherical harmonics or the directional basis.

Our lighting basis uses Haar wavelets over blocks, called *segments*, of a cube map. Quantizing or truncating near-zero coefficients then provides a simple way to approximate the lighting that preserves its most important content [NRH03]. Since the surface exhibits a more coherent response to lighting from the same general direction, we perform a 2×2 segmentation of the six faces of the cube map and perform a Haar transform on the 16×16 image in each of the 24 segments. A lighting segment corresponds to a *transfer segment* or set of columns of the transfer matrix that represents the surface's linear response to that lighting segment alone. This segmentation does not constrain the lighting we can handle in any way; it is merely a device to speed compression of the signal by an a priori division of it into parts that are likely to be coherent.

We then use clustered principal component analysis (CPCA) [KL97] to approximate each transfer segment as a linear combination of a few (n=16) representatives. Rather than forming a global approximation over the entire object, CPCA clusters points and computes an independent approximation in each cluster. As shown by [SHHS03], CPCA provides a highly-compressed yet accurate approximation for nonlinear PRT signals. It also accelerates rendering by performing expensive matrix/vector multiplies only per-cluster and reducing the per-vertex computation to a weighted combination of n m-dimensional vectors, followed by an m-dimensional dot product. The same benefits apply to our PRT formulation, though our transfer signal is based on different input and output bases and has nearly 100 times as many dimensions.

Our main contribution is to introduce the first interactive method for rendering glossy objects with global effects due to "all-frequency" (not just low-frequency) lighting. Though BRDF factorization is not new, our application of it to PRT is novel and greatly reduces the number of rows in our transfer matrices while maintaining accuracy for specular BRDFs. Even so, we deal with a high-dimensional ($m \times n_L$) transfer signal much larger than any from previous PRT work. We show effective compression using a light-segmented CPCA encoding whose clusters automatically adapt to sharp shadow boundaries on the surface. We also show how our compression reduces the run-time's dependence on the number of lighting coefficients, allowing fast rendering without truncating any of the lighting energy. Finally, we demonstrate high-quality results on graphics hardware.

2. Formulation of the Factored PRT Signal

In the following, we use the term *global frame* for a coordinate frame in which the source lighting is represented and which is shared by all vertices on the object, and *local frame* for a coordinate frame determined by the normal and tangent directions at each vertex p. Lower-case letters denote scalars or low-dimensional (2 or 3) vectors, capital letters denote high-dimensional vectors or matrices. Our formulation is based on [SKS02], but introduces BRDF factorization and a generic lighting basis.

Shading at point p with view vector v is given by an integral over the hemisphere $\mathbf{H}$ of lighting directions s in the local frame:

$$g_p(v) = \int_{s \in \mathbf{H}} f(v,s)\, t_p(s)\, ds \tag{1}$$

$t_p(s)$ is the *transferred incident radiance function*. It is the radiance from the source lighting that arrives at p after including transport effects like self-shadowing from the object. $t_p(s)$ also rotates the lighting from a global to to a local frame. Note that $t_p(s)$ is a linear operator on the source lighting, $l(s)$. We denote this by

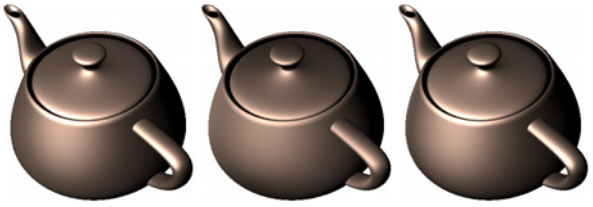
$$t_p(s) = t_p[l(s)] \tag{2}$$

The *BRDF product function* is defined as $f(v,s) = b(v,s)\, s_z$ where $b(v,s)$ is the BRDF and s_z is the cosine factor (normal component of the incident radiance direction s).

We factor f via

$$f(v,s) = G(v) \cdot F(s) \tag{3}$$

where $G(v)$ and $F(s)$ are m-dimensional vector functions purely of view direction v and light direction s respectively. The PRT surface signal is then given by the following linear



original m=5, error=10% m=10, error=2.1%

Figure 2: *BRDF factorization example. Errors are sum of squared differences divided by total sum of squares over all BRDF samples.*

operator (having m outputs) at each point p

$$M_p[l(s)] = \int_{s \in \mathbf{H}} F(s)\, t_p[l(s)]\, ds \tag{4}$$

Because t_p is a linear operator, M_p also depends linearly on the source lighting function, $l(s)$. Any linear basis for source lighting, such as the spherical harmonic basis, the Haar wavelet basis over cube map faces, or even a directional basis, represents $l(s)$ via

$$l(s) = \sum_{i=1}^{n_L} L_i\, l_i(s) \tag{5}$$

where $l_i(s)$ is the i-th lighting basis function and L_i is its coefficient.

In this lighting basis, the PRT signal becomes a $m \times n_L$ transfer matrix at each point, M_p, a component of which is given by

$$(M_p)_{ij} = \int_{s \in \mathbf{H}} F_i(s)\, t_p[l_j(s)]\, ds \tag{6}$$

where $t_p[l_j(s)]$ is a scalar spherical function representing transferred incident radiance from the j-th lighting basis function, $l_j(s)$. M_p's rows represent contribution to one component of transferred incident radiance corresponding to $F_i(s)$. Its columns represent the response to one lighting basis function, $l_j(s)$.

Combining equations (1-6), the shading result is given simply by

$$g_p(v) = G^T(v)\, M_p\, L = G(v) \cdot \left(M_p L\right) \tag{7}$$

where $G^T(v)$ is the m-dimensional row vector formed by transposing the column-vector BRDF factor $G(v)$, M_p is the transfer matrix at p, and L is the vector of lighting coefficients.

3. BRDF Factorization

To factor the BRDF product function $f(v,s)$, we apply a simple modification of the method in [KM99]. More recent BRDF factorizations reduce error [MAA01], efficiently handle (unoccluded) area lighting [LK03a, RH03] and handle 6D BTFs [SvBAD03]. But they all use products of 2D functions whose parameters mix view and light directions. When applied to PRT, such "mixed" factors prohibit our simple formulation (equation 7) and so are inappropriate. [NN95] considers separable BRDFs for radiosity, but only single term expansions. With enough terms (big enough m), [KM99] and our work can drive the approximation error to 0 for arbitrary BRDFs.

We begin by forming a matrix Q whose components are $Q_{ij} = f(v_i, s_j)$, with n_v view samples, v_i, and n_s light samples, s_j. The viewing and lighting directions are parameterized using the parabolic map [HDKS00]. We use $n_v = n_s = 32 \times 32 = 1024$ sampled directions for both view and light.

We then perform the singular value decomposition on the matrix Q and set all but the largest m singular values to zero. Then

$$Q_{ij} \approx \sum_{k=1}^{m} G_{ik}\, \sigma_k\, F_{kj} \tag{8}$$

Absorbing a square root of the diagonal matrix formed by the singular values σ_k into both the left and right factors, we obtain our two functions $G(v)$ and $F(s)$ via

$$f(v_i, s_j) \approx \sum_{k=1}^{m} G_k(v_i)\, F_k(s_j) = G(v_i) \cdot F(s_j) \tag{9}$$

Note that continuous functions result from interpolation between sample points in the parabolic parameterization space.

Generally, a specular BRDF has high values when the view direction aligns with the reflected light direction and drops off rapidly at nearby samples. Our sampled view and light directions are located on a regular grid and so can easily miss these important features. To avoid aliasing, we supersample each pair of view/light samples by a factor of 16×16. The limited sampling has the effect of smoothing highly specular BRDFs.

Unlike [KM99], we include the cosine factor in our BRDF factorization. This attenuates the function's values, making it easier to approximate. In practice, we find the nonlinear operation of clamping values of f bigger than 3 before performing the SVD provides more visual accuracy.

Figure 2 shows an example of BRDF factorization on the Cook-Torrance lighting model [CT82] with the following parameter values: facet slope (roughness)=0.4, Fresnel=0.5. Differences can be seen in the sharpness of the highlights, especially in the spout and handle. Good accuracy is obtained using factors with only m=10 dimensions. Similar accuracy is also obtained on the anisotropic lighting model of [Sch94], used in Figures 1 and 7, with parameter values roughness=0.2, isotropy=0.2, and Fresnel=0.8. Figure 3 shows how accuracy depends on the specularity of the lighting model.

Our BRDF factorization essentially chooses an output basis for the PRT signal specially adapted to the particular

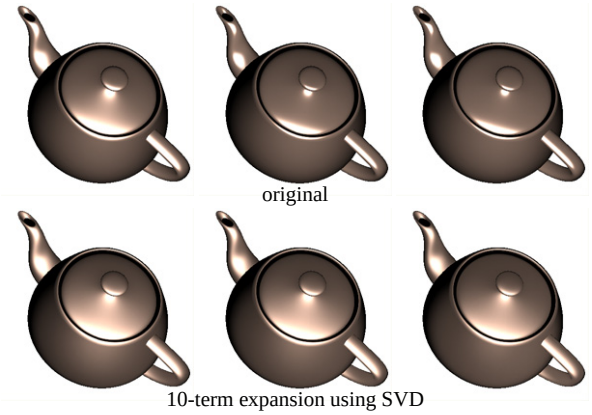


Figure 3: *Limits of specularity with 10-term BRDF factorization. The first row shows original images using the Cook-Torrance lighting model with increasing specularity. The second row shows our 10-term approximation, which exhibits visual error at roughness < 0.3.*

BRDF. In contrast, [SKS02, SHHS03] use an unspecialized output basis (spherical harmonics) having 25 rows. To compare techniques, we performed an error analysis by computing RMS and maximum pointwise error for the two lighting models above, using 1024 samples in both light direction and view direction sampled over the hemisphere. The following table summarizes results:

Method	# rows	Cook-Torrance		Schlick	
		RMS	max	RMS	max
SVD	10	0.0677	0.554	0.0389	0.298
SH (order 4)	16	0.156	1.21	0.121	0.634
SH (order 5)	25	0.102	0.967	0.0766	0.438
SH (order 6)	36	0.0708	0.727	0.0511	0.333
SH (order 7)	49	0.0538	0.532	0.038	0.267

The bottom line is that our 10 row SVD (BRDF-specialized) transfer matrix is equivalent to between a 36 and 49 row spherical harmonic (SH) matrix for both lighting models, saving more than a factor of 4 in signal dimensionality.

4. PRT Computation

We outline our precomputation of the PRT signal M_p. Currently, our implementation only handles direct shadowing effects and ignores inter-reflection.

At each mesh vertex p, we first compute a visibility map, $q_p(s_j)$, at directional samples s_j which returns 0 if p is shadowed in the direction s_j and 1 if it is unshadowed. We use a cube map in a global coordinate system to parameterize the directions s_j. Directions below the hemisphere around p's normal are ignored. We supersample q_p 4×4, yielding a $6 \times 128 \times 128$ cube map.

From the visibility map, we then compute a "raw" transfer matrix signal, $\tilde{M}_p$ which integrates against the BRDF basis functions $F_i(s)$. Unlike the final transfer matrix, it uses a directional lighting basis. This raw transfer matrix is given by the following integral over a small cone of directions in the global frame:

$$\left(\tilde{M}_p\right)_{ij} = \int_{s \in \mathbf{C}(s_j)} d_{s_j}(s)\, q_p(s)\, F_i\left(R_p(s)\right)\, ds \qquad (10)$$

where $\mathbf{C}(s_j)$ is the cone of directions within one cube map texel of the direction s_j, $d_{s_j}(s)$ is the bilinear basis function on the cube map centered at the sample s_j, and R_p rotates the direction s from the global frame to the local frame.

The raw transfer signal $\tilde{M}_p$ has $m \times n_L$= 61,440 dimensions at each vertex, requiring several gigabytes of storage for typical models in single precision. To compress it, a simple method is to extend the technique in [NRH03] to these transfer matrices by applying the Haar wavelet transform over the light dimensions, quantizing to a reasonable precision, and exploiting sparsity. We find that this reduces our data by roughly a factor of 10, but the compressed data is still impractically large.

Our method partitions the lighting basis into 24 segments, using a 2×2 subdivision of the cube map faces. Each resulting transfer segment has $n_T = m \times (n_L/24) = 2560$ dimensions, corresponding to columns in the transfer matrix that respond to that lighting segment. We then compress using CPCA over each transfer segment (see next section). We apply the Haar wavelet transform to the representative matrices (eigen-matrices) in each cluster and quantize them to 16 bits of precision, followed by a lossless sparsity coding. We also quantize the per-vertex weights to 16 bits. This provides a compression factor of 77 (1.3% of raw size) on our glossy bunny; compression results for other models can be found in Table 1.

To exploit sparsity in the quantized representative transfer matrices, we use the method in [NRH03], which applies the normalized, non-standard Haar transform, but we quantize to 16 bits rather than 8. The basic idea is to form a matrix where each row is one representative transfer matrix from the clusters. We block this matrix into groups of 256 rows, and perform a sparse-matrix encoding over columns (i.e., over single components of the representatives). In other words, we store a row index and 16-bit quantized value only for components that are non-zero.

5. PRT Compression using CPCA

CPCA Representation We apply the general technique of CPCA to 24 separate signals consisting of an n_T dimensional transfer segment at each surface vertex. We will continue to denote the signal as M_p though it is understood in the following that the signal is only a subset of the columns of the matrix in equation (6). Because CPCA encoding is quite slow

and is quadratic in the signal dimension n_T, dividing the signal into 24 independent components makes the computation faster.

CPCA approximates a transfer segment via the following linear combination in each cluster

$$M_p \approx \sum_{i=1}^{n} w_p^i M^i \tag{11}$$

where n is the number of representative matrices per cluster, w_p^i are n scalar weights that vary spatially, and M^i are n representative matrices which are constants for each cluster. (An alternative formulation is an affine combination which includes the unweighted cluster mean [SHHS03], and complicates the analysis below slightly.)

We achieve a good approximation for interactive rendering using n=16. Figure 5 compares rendering quality from various n.

CPCA Compression Analysis Only lighting segments above the normal's hemisphere contribute to shading; segments below it can be ignored. At any vertex, at least 4 out of 24 segments are below the normal's hemisphere; on average, about 8 out of 24 are. So a given transfer segment will be nonzero for only ω = 2/3 of all vertices, even without any shadowing. We take advantage of this by culling all vertices whose shading ignores a particular segment before beginning the CPCA encoding. Extensive self-shadowing on the object, such as on the buddha model, further decreases ω (see Table 1) which reduces the signal's size even more.

To analyze compression after elimination of entirely zero transfer segments, let n_C be the number of vertices in a cluster. The size of the compressed representation in a single cluster is $n_T\, n$ to store the representative transfer segments and $n_C\, n$ to store the per-point weights. The uncompressed size of the cluster data is $n_C\, n_T$. This provides a compression ratio r defined by

$$r = \frac{n_C\, n_T}{n(n_C + n_T)} \tag{12}$$

As a typical example, assume we wish to encode a single transfer segment over a model with 16,000 vertices. Assuming ω=2/3, only 10,667 of these vertices have a nonzero transfer segment. Using 64 clusters in each segment (64× 24 total clusters) yields n_C=167 average vertices per cluster, and a compression ratio estimate of r≈9.8. This is only an estimate because CPCA allows n_C to vary over clusters; we can only fix its average. The total compressed size is roughly ω/r = 6.8% of the raw signal, including both CPCA and elimination of zero transfer segments.

CPCA Encoding To compute a CPCA approximation, we use the method called "iterative CPCA" in [SHHS03], which is a simple generalization of VQ clustering [LBG80]. The method alternates between classifying a point in the cluster providing the smallest approximation error followed by updating the cluster's representatives using an eigen-analysis over all points classified to it. Figure 4 shows how well this method adapts to the object's self-shadowing.

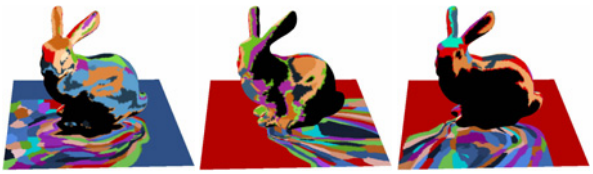


Figure 4: *CPCA clustering visualization for three different transfer segments. Areas on the surface receiving no light from the segment are colored black. Note how well CPCA adapts to the shadowing.*

We currently determine the total number of clusters by dividing the total number of nonzero transfer segments over all vertices by 200. This fixes an average n_C=200, which in turn targets a compression ratio of r≈11.6, not counting elimination of zero segments.

CPCA Rendering Besides compression, another advantage of CPCA is that the representation can be rendered directly without the need to reconstruct the entire transfer matrix [SHHS03, LK03b]. The result is a significant runtime speedup. To see this, apply our approximate operator to the lighting to get an m-dimensional vector, T_p, representing transferred incident radiance with respect to the BRDF-specialized output basis $F(s)$, via

$$T_p = M_p L \approx \sum_{i=1}^{n} w_p^i \left(M^i L\right) = \sum_{i=1}^{n} w_p^i T^i \tag{13}$$

So instead of reconstructing a transfer matrix at each p and then applying it to the lighting, we compute n matrix/vector multiplies in each cluster to obtain the T^i, and only have to perform n weighted combinations of these vectors at each vertex. From equation (7), the final shade is then given by the dot product of the m-dimensional vectors T_p and $G(v)$.

CPCA thus makes the computation fairly insensitive to the number of lighting basis functions n_L. Critically, the per-vertex computation no longer depends on n_L at all, only on m and n. The per-cluster computation (of the T^i) does depend on n_L, but there are many times fewer clusters than vertices (see Table 1).

6. PRT Rendering

Our PRT rendering method is based on [SHHS03], but is applied to transfer matrices having a different input basis (Haar wavelets over cube map segments) and output basis (m-dimensional BRDF-specialized functions $F_i(s)$), rather than spherical harmonics. It performs the following four steps:

1. Project the time-varying lighting environment onto the cube map, and then into the Haar basis over each segment to obtain the lighting vector L.

2. For each transfer segment, transform the lighting through each cluster's representatives to obtain the $T^i = M^i L$.
3. At each vertex, reconstruct the transferred radiance vector T_p using a weighted combination of the T^i in equation (13).
4. Compute the local view direction v and return the dot product of $G(v)$ with T_p.

Since the lighting vector L has three color channels, these steps are performed for each color channel. Shadowed transfer requires only single-channel transfer matrices since all colors are occluded by the object in the same way. We compute a simple 3-channel multiplication of the output of step 4 to provide surface color.

Lighting Projection When transforming the $6\times32\times32$ lighting cube map into the vector L, it is important to consider aliasing, especially with high dynamic range lighting which can contain very high frequencies. We supersample the lighting 4×4 and decimate before computing the segmented Haar transform.

Another issue is lighting *truncation* which eliminates unimportant lighting coefficients to make the vector L sparser. [NRH03] called this "non-linear" approximation and presented several strategies for it, including truncating coefficients with the lowest magnitude, magnitude weighted by spherical area, and magnitude weighted by average transfer response on the surface. Though truncation can accelerate performance, it risks temporal artifacts when the lighting changes if significant lighting energy is truncated.

In our method, such truncation strategies acclerate the per-cluster computation (next subsection). But we are less dependent on truncation to achieve reasonable performance, because per-cluster work forms only a part of the computation.

Per-Cluster Computation Computing the $T^i = M^i L$ involves a sparse matrix/vector multiply on the CPU. Sparsity in L drives the computation. We find that when we eliminate all truncation, the per-cluster and per-vertex computation times are about equal.

Per-Vertex Computation Though CPCA reduces it, per-vertex computation remains significant, requiring ω (average fraction of vertices having nonzero segment) $\times$ 24 (segments) $\times$ n=16 (representatives) $\times$ m=10 (transfer output components) $\times$ 3 (color channels) or 7680 multiplies per vertex for ω=2/3. The computation is linear in m and n so reducing them speeds things up. We also note that when only the view is changed (i.e., the light remains fixed relative to the object), then only step 4 above must be recomputed.

Unlike the per-cluster computation, the per-vertex computation processes short, contiguously-accessed vectors; i.e., it is "dense" rather than sparse. This makes it suitable for GPU implementation. We currently do all the shading computation, including the per-vertex part, on the CPU. However, we note that some of the computation could be simply transferred to the GPU, using a method similar to that in [KSS02]. The idea is to use a texture map for $G(v)$, interpolate T_p over triangles, and do the dot product in step 4 in the pixel shader. This requires streaming the m-dimensional transfer output signal T_p to the graphics hardware.

model	bunny	teapot	tweety(d)	tweety(g)	buddha(d)	buddha(g)
material	Cook	Schlick	diffuse	Schlick	diffuse	Cook
vertices	11.5k	51.3k	32.8k	32.8k	54.1k	54.1k
signal dim	2560×24	2560×24	256×24	2560×24	256×24	2560×24
raw data (Gb)	2.84	11.7	.768	7.68	1.24	12.4
ω	56.0%	61.8%	57.6%	57.6%	42.4%	42.4%
total clusters	763	3801	2260	2263	2739	2740
CPCA comp.	4.43%	5.36%	8.16%	4.85%	6.23%	3.65%
wavelet comp.	29.5%	31.2%	50.0%	30.4%	54.5%	32.2%
total comp.	1.31%	1.67%	4.17%	1.42%	3.39%	1.19%
squared error	0.20%	0.051%	0.02%	0.79%	0.11%	0.0063%
render time (fps)	5.0/6.2	1.2/1.3	8.4/14	1.7/1.8	6.6/9.4	1.1/1.2

Table 1: *Results, m=10 (glossy) or m=1 (diffuse), n=16. Render times are shading+draw/shading+no-draw using the St. Peter's HDR lighting environment with no lighting truncation.*

7. Results

Table 1 shows compression and performance statistics for various models. We obtain compression ratios ranging from 24:1 to 84:1 with low error. The performance numbers in Table 1 do not include a ground plane; timings in Figures 6 and 7 do. Because it is diffuse and so requires only single-row transfer matrices (m=1), the ground plane cost in terms of both storage and rendering time is minimal compared to the glossy objects themselves. We used a ground plane having 192x192=36864 vertices in all examples. For the bunny example, the raw data was about 715MB which compressed down to 23.6MB; compression results for other models are very similar. Our CPU shading code has been optimized to use multi-threading and SSE. Our rendering code is not optimized. All timings were performed on an dual Intel Xeon 3GHz PC with ATI Radeon 9800.

CPCA-encoding takes roughly 4 hours for the most complex (50k vertex) glossy models, about 10 minutes for the complex diffuse models, and correspondingly less for models with fewer vertices. Simulation of self-shadowing takes about 20 minutes for the most complex models.

Figure 6 compares truncation of different numbers of lighting coefficients (denoted $|L|$) on the diffuse Buddha model. For this experiment, we applied truncation using both the lighting basis and area $\times$ magnitude priority scheme from [NRH03]. (Note that [NRH03]'s lighting basis differs from ours in that it does not segment the cube map faces. We used a basis from previous work to show that the need for more coefficients does not arise from this segmentation.) We then picked the largest priority coefficients and reprojected into our segmented basis. The lighting environment consists of a collection of 40 small sources distributed over

the sphere. In such high-frequency environments, truncation causes visible artifacts unless a large fraction (≈70%) of the lighting coefficients is retained. Even worse, objectionable flickering artifacts result from animating the model or the lights, because the truncation changes as the lights move (see Video #2). Our method can render interactively without any lighting truncation at all; note how slowly the frame rate decreases as we increase the number of lighting coefficients.

Figure 7 shows example images and rendering performance for our more complicated models. The left two columns compare diffuse and glossy reflectance on the Buddha. The right column compares low-frequency (top) and all-frequency (bottom) lighting on an anisotropically glossy teapot.

8. Conclusion

Handling specular BRDFs and all-frequency lighting in PRT requires high-dimensional transfer matrics which are recorded at many points on the surface. Our method makes this practical with two key ideas. We factor BRDFs into separate view- and light-dependent parts, absorbing the light-dependent part into the transfer matrix. This greatly reduces the number of rows in our matrices by specializing transfer output to the object's particular reflectance. We then apply CPCA to the transfer signal to compress it and accelerate its rendering in a way that reduces computational dependence on the size of the lighting basis. This allows us to include a much bigger fraction (even 100%) of the lighting energy without slowing the run-time much. Our results demonstrate interactive performance that renders shadows on glossy objects from all-frequency, dynamic lighting.

In future work, we wish to include inter-reflections and subsurface scatter. One advantage of PRT is that such effects will have little impact on the run-time complexity. Nevertheless, the preprocessing is made challenging because we currently take advantage of the fact that shadowed transfer requires only a diagonal matrix when using the directional lighting basis, and thus can be stored using n_L-dimensional bit vectors, rather than n_L^2-dimensional matrices. Including inter-reflection requires simulation with a full transfer matrix. We are also interested in experimenting with better light bases, including smoother wavelets and parameterizations that sample more uniformly over the sphere. Finally, we wish to move more of the computation from the CPU to the GPU, especially by parameterizing the object and recording PRT as a texture signal to be processed in a pixel shader.

References

[CT82] COOK R., TORRANCE K.: A reflectance model for computer graphics. *ACM TOG 1*, 1 (1982), 7–24.

[HDKS00] HEIDRICH W., DAUBERT K., KAUTZ J., SEIDEL H.: Illuminating micro-geometry based on precomputed visibility. In *Proc. SIGGRAPH '00* (2000), pp. 455–464.

[KL97] KAMBHATLA N., LEEN T.: Dimension reduction by local prinicpal component analysis. *Neural Computation 9* (1997), 1493–1516.

[KM99] KAUTZ J., MCCOOL M.: Interactive rendering with arbitrary brdfs using separable approximations. *Rendering Techniques '99 (Eurographics Workshop on Rendering)* (1999), 281–292.

[KSS02] KAUTZ J., SLOAN P., SNYDER J.: Fast, arbitrary brdf shading for low-frequency lighting using spherical harmonics. *Eurographics Workshop on Rendering* (2002), 291–296.

[LBG80] LINDE Y., BUZO A., GRAY R.: An algorithm for vector quantizer design. *IEEE Transactions on Communication COM-28* (1980), 84–95.

[LK03a] LATTA L., KOLB A.: Homomorphic factorization of brdf-based lighting computation. In *Proc. of SIGGRAPH '03* (2003), pp. 509–516.

[LK03b] LEHTINEN J., KAUTZ J.: Matrix radiance transfer. *Symposium on Interactive 3D Graphics* (2003), 59–64.

[MAA01] MCCOOL M., ANG J., AHMAD A.: Homomorphic factorization of brdfs for high-performance rendering. In *Proc. of SIGGRAPH '01* (2001), pp. 171–178.

[NN95] NEUMANN L., NEUMANN A.: Radiosity and hybrid methods. *ACM TOG 14*, 3 (1995), 233–265.

[NRH03] NG R., RAMAMOORTHI R., HANRAHAN P.: All-frequency shadows using non-linear wavelet lighting approximation. In *Proc. of SIGGRAPH '03* (2003), pp. 376–381.

[RH03] RAMAMOORTHI R., HANRAHAN P.: Frequency space environment map rendering. In *Proc. of SIGGRAPH '03* (2003), pp. 517–526.

[Sch94] SCHLICK C.: An inexpensive brdf model for physically-based rendering. *Computer Graphics Forum 13*, 3 (1994), 233–246.

[SHHS03] SLOAN P., HALL J., HART J., SNYDER J.: Clustered principal components for precomputed radiance transfer. In *Proc. of SIGGRAPH '03* (2003), pp. 382–391.

[SKS02] SLOAN P., KAUTZ J., SNYDER J.: Precomputed radiance transfer for real-time rendering in dynamic, low-frequency lighting environments. In *Proc. of SIGGRAPH '02* (2002), pp. 527–536.

[SvBAD03] SUYKENS F., VOM BERGE K., ARES L., DUTRE P.: Interactive rendering with bidirectional texture functions. *Computer Graphics Forum 22*, 3 (2003), 463–472.

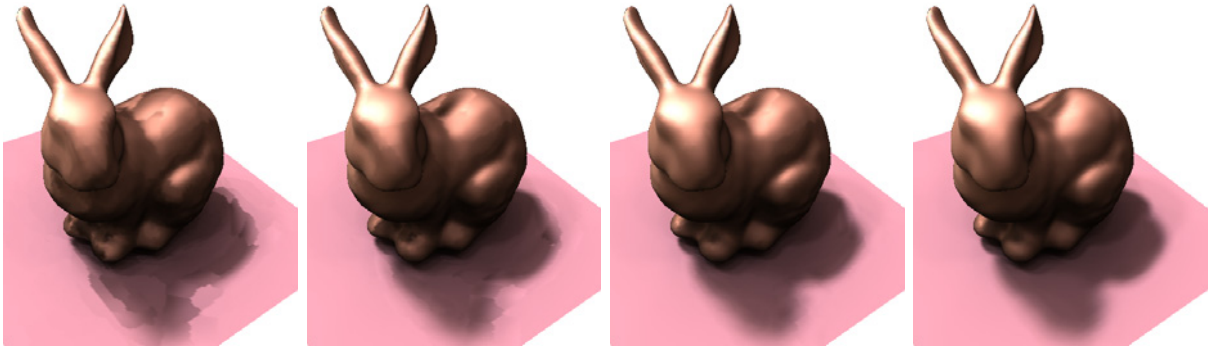


Figure 5: *Varying number of representatives, n. Note cluster artifacts especially at shadow boundaries, which are almost invisible for n=16.*

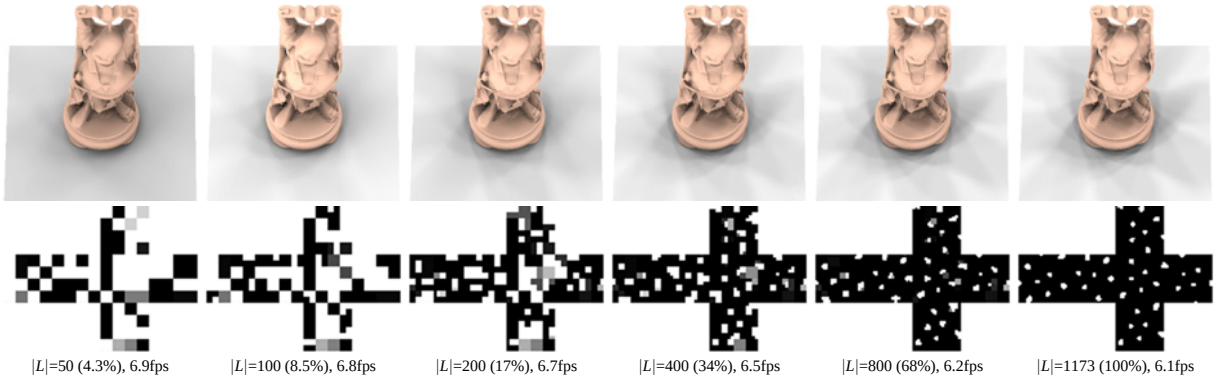


Figure 6: *Light truncation comparison. The bottom row displays the number of untruncated coefficients, the total fraction represented by that number in parentheses, and the frame rate obtained. The top row shows the rendered result from the truncated lighting environment shown in the middle row. When rotating the lighting, temporal artifacts ("flicker") appear with fewer than about 800 coefficients (68% of nonzero coefficients) in this example.*

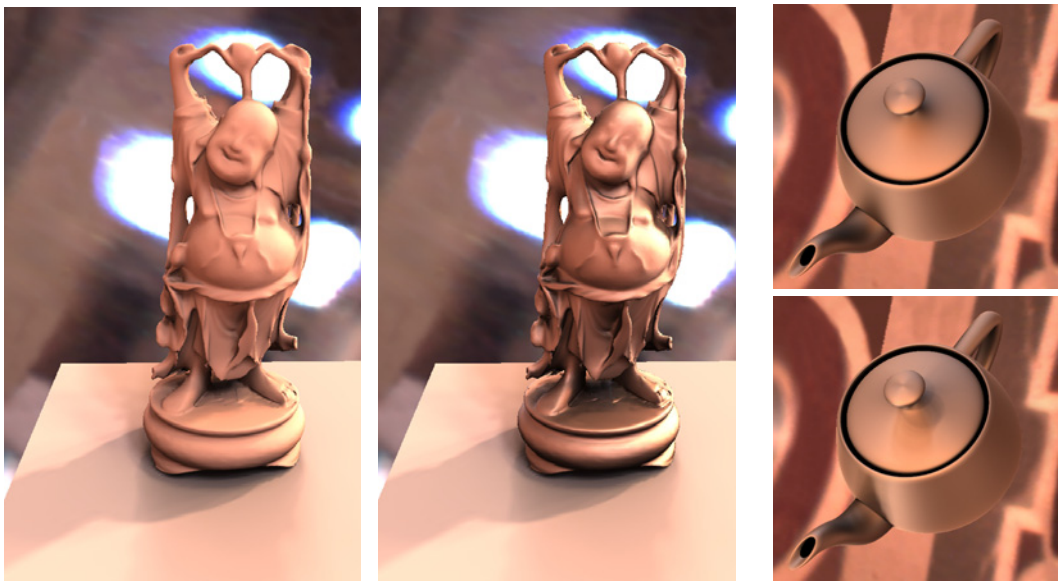


Figure 7: *Example images: diffuse buddha (left, 5.8fps), glossy buddha (middle, 1fps), low-frequency teapot (top right,1.2fps), all-frequency teapot (bottom right, 1.2fps).*