Clustered Principal Components for Precomputed Radiance Transfer

Peter-Pike Sloan
Microsoft Corporation

Jesse Hall
University of Illinois

John Hart
University of Illinois

John Snyder
Microsoft Research

Abstract

We compress storage and accelerate performance of precomputed radiance transfer (PRT), which captures the way an object shadows, scatters, and reflects light. PRT records over many surface points a transfer matrix. At run-time, this matrix transforms a vector of spherical harmonic coefficients representing distant, low-frequency source lighting into exiting radiance. Per-point transfer matrices form a high-dimensional surface signal that we compress using *clustered principal component analysis* (CPCA), which partitions many samples into fewer clusters each approximating the signal as an affine subspace. CPCA thus reduces the high-dimensional transfer signal to a low-dimensional set of per-point weights on a per-cluster set of representative matrices. Rather than computing a weighted sum of representatives and applying this result to the lighting, we apply the representatives to the lighting per-cluster (on the CPU) and weight these results per-point (on the GPU). Since the output of the matrix is lower-dimensional than the matrix itself, this reduces computation. We also increase the accuracy of encoded radiance functions with a new least-squares optimal projection of spherical harmonics onto the hemisphere. We describe an implementation on graphics hardware that performs real-time rendering of glossy objects with dynamic self-shadowing and interreflection without fixing the view or light as in previous work. Our approach also allows significantly increased lighting frequency when rendering diffuse objects and includes subsurface scattering.

Keywords: Graphics Hardware, Illumination, Monte Carlo Techniques, Rendering, Shadow Algorithms.

1. Introduction

Global illumination effects challenge real-time graphics, especially in area lighting environments that require integration over many light source samples. We seek to illuminate an object from a dynamic, low-frequency lighting environment in real time, including shadowing, interreflection, subsurface scattering, and complex (anisotropic) reflectance.

These effects can be measured as radiance passing through spherical shells about the surface point p in Figure 1. *Source radiance* originates from an infinite sphere (environment map). *Transferred incident radiance* passes through an infinitesimal hemisphere, and equals the source radiance decreased by self-shadowing and increased by interreflection. *Exiting radiance* passes outward through an infinitesimal hemisphere, and results from the BRDF times the transferred incident radiance, plus subsurface scattering.

The spherical harmonic (SH) basis provides a compact, alias-avoiding representation for functions of radiance over a sphere or hemisphere [Cabral *et al.* 1987][Sillion *et al.* 1991][Westin *et al.* 1992][Ramamoorthi and Hanrahan 2001]. Low-frequency source illumination, which small vectors (e.g. N=25) of SH coefficients approximate well [Ramamoorthi and Hanrahan 2001][Sloan *et al.* 2002], is exactly the situation in which integration over the light becomes the bottleneck for traditional rendering methods.

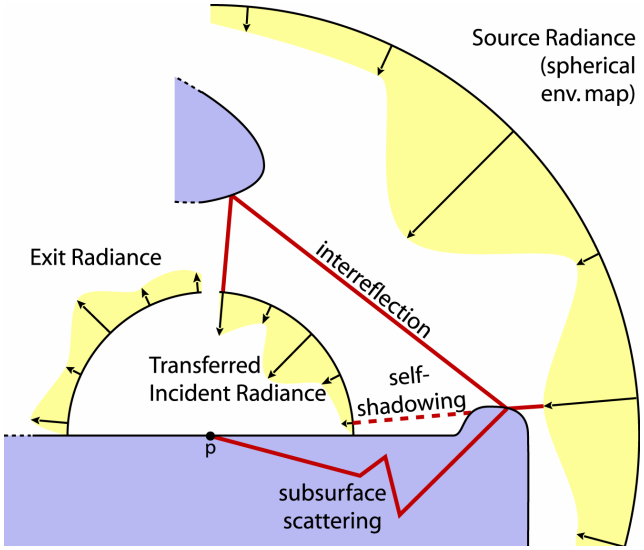


Figure 1: Radiance transfer at p from source to transferred incident to exit.

Sloan *et al.* [2002] precompute the radiance transfer of an object in terms of low-order SHs. For a diffuse object, exiting radiance results from dotting a 25-vector, representing the source radiance, with a 25-element radiance transfer vector precomputed and stored at each sample point p. By storing this transfer vector per-vertex, real-time self-shadowing and interreflection results from a simple vertex shader. For a glossy object, [Sloan *et al.* 2002] represents radiance transfer as a linear operator converting a 25D source radiance vector into a 25D transferred radiance vector, via a 625-element transfer matrix that varies for each p. This glossy transfer matrix was too big for graphics hardware. The CPU implementation ran at interactive rates (~4 Hz) and could achieve real-time frame rates only for a constant view or lighting, hampering its usefulness for applications like 3D games.

Our method lifts these restrictions, rendering the same glossy objects more than 10-20 times faster. For simpler diffuse transfer, the method allows higher frequency lighting (i.e., higher-order SH projections) for the same computational cost. As in [Lehtinen and Kautz 2003], we precompute and store per-vertex the source-to-exiting radiance transfer matrix, instead of the source-to-incident transfer [Sloan *et al.* 2002]. This also allows us to include the precomputed contribution of the object's subsurface scattering of distant environmental light.

To get real-time performance, we treat the transfer vectors or matrices stored at each vertex as a surface signal and partition them into a few (128-256) clusters. Principal component analysis (PCA) approximates the points in each cluster as a low-dimensional affine subspace (mean plus up to n'=8 PCA vectors). We call this approach *clustered principal component analysis* (CPCA). For glossy transfer, CPCA reconstructs a good approximation of its $N \times N$ matrix at each point by storing only the index of its cluster and a few ($n' \ll N \ll N^2$) scalar coordinates of

projection onto the cluster's PCA vectors. CPCA reduces not only signal storage (n' rather than N^2 scalars per point) but also the run-time computation. Instead of multiplying an $N \times N$ transfer matrix by an N-dimensional light vector at each p, we precompute this multiplication in each cluster for each of its PCA vectors and accumulate weighted versions of the n' resulting N-vectors. CPCA on diffuse transfer provides a similar savings in storage and computation.

We describe two technical contributions which may have wider applicability. The first is a very general signal approximation method using CPCA. Though used before in machine learning applications [Kambhatla and Leen 1994][Kambhatla and Leen 1997][Tipping and Bishop 1999], it is new to computer graphics. To increase spatial coherence, we augment the method by redistributing points to clusters according to an "overdraw" metric. The second contribution is the use of the optimal least-squares projection of the SH basis onto the hemisphere, which significantly reduces error compared to approaches used in the past [Sloan *et al.* 2002][Westin *et al.* 1992].

2. Related Work

Various representations encapsulate precomputed or acquired global illumination. Light fields [Gortler *et al.* 1996][Levoy and Hanrahan 1996] record radiance samples as they pass through a pair of viewing planes whereas surface light fields [Chen et al. 2002][Miller *et al.* 1998][Nishino *et al.* 1999][Wood *et al.* 2000] record 4D exiting radiance sampled over an object's surface. Both techniques support arbitrary views but fix lighting relative to the object.

Precomputed radiance transfer (PRT) [Sloan *et al.* 2002] parameterizes transferred incident radiance in terms of low-frequency source lighting, allowing changes to lighting as well as viewpoint. We build on PRT and its generalization to anisotropic BRDFs [Kautz *et al.* 2002], but speed up performance and reduce error in three ways: we record exiting radiance instead of transferred incident, use least-squares optimal projection of hemispherical functions, and compress using CPCA. We also extend PRT to include subsurface scattering. In parallel work, Lehtinen and Kautz [2003] approximate PRT using PCA. Our CPCA decoding reduces approximation error and maps well to the GPU, resulting in 2-3 times better performance.

Other illumination precomputation methods also support dynamic lighting. Matusik *et al.* [2002] handle limited, non-real-time lighting change with a surface reflectance field measured over a sparsely sampled directional light basis, stored on the visual hull of an acquired object. Hakura *et al.* [2000] support real-time lighting change with parameterized textures, but constrain viewing and lighting changes to a 2D subspace (e.g. a 1D circle of viewpoints × 1D swing angle of a hanging light source). [Sloan *et al.* 2002] compares PRT to many other precomputed approaches for global illumination.

Precomputed illumination datasets are huge, motivating compression. Light fields were compressed using vector quantization (VQ) and entropy coding [Levoy and Hanrahan 1996], and reflectance fields using block-based PCA [Matusik *et al.* 2002]. Surface light fields have been compressed with the DCT [Miller *et al.* 1998], an eigenbasis (PCA) [Nishino *et al.* 1999], and generalizations of VQ or PCA to irregular sampling patterns [Wood *et al.* 2000]. Our CPCA compression strategy improves [Wood *et al.* 2000] by hybridizing VQ and PCA in a way that reduces error better than either by itself. Unlike [Chen et al. 2002] which compresses a 4D surface light field over each 1-ring mesh neighborhood, our clustering is free to group any number of samples that can be approximated well together regardless of their surface location. Our purpose is real-time rendering with graphics hardware, not minimizing storage space. For example, we avoid entropy coding for which current graphics hardware is ill-suited.

Jensen *et al.* [2002] simulate translucent materials using a diffusion approximation of subsurface scattering accelerated by decoupling the computation of irradiance from a hierarchical evaluation of the diffusion approximation. This paper also experimentally validated when the multiple scattering term dominated. Two recent paper exploit this property and implement interactive rendering techniques based on the idea. Lensch *et al.* [2002] combine spatially varying filters in texture space with vertex-to-vertex transfer to model near and far subsurface transport. Global shadowing and interreflection effects are ignored and only ~5Hz frame rate is obtained. Hao *et al.* [2003] precompute subsurface scattering for a directional light basis. We model smooth, distant lighting environments and include a glossy term to approximate single scattering.

Like PRT, Westin *et al.* [1992] also use matrices which transform lighting into exiting radiance, both expressed in the SH basis. Their matrices encode local effects for BRDF synthesis, not spatially-varying global transport for real-time rendering. They devise a SH projection of hemispherical functions, which we improve via least-squares in the appendix.

Lensch *et al.* [2001] use a similar clustering procedure to reconstruct a spatially-varying BRDF from images. They fit parameters of a BRDF model in each cluster using nonlinear optimization and approximate using a linear combination of the resulting models, one per cluster. We use an independent affine basis per cluster.

3. Radiance Transfer Signal Representation

For diffuse surfaces, PRT encodes a transfer vector, t_p, per surface point p [Sloan *et al.* 2002]. The i-th component of this vector represents the linear contribution of source lighting basis function $y_i(s)$ to the exiting radiance of p. For glossy surfaces, we make several modifications to the glossy transfer matrix defined in [Sloan *et al.* 2002].

3.1 Transferred Incident vs. Exiting Radiance Transfer

PRT in [Sloan *et al.* 2002] represents *transferred incident* radiance (Figure 1). It is derived from a Monte Carlo simulation illuminating geometry by the SH basis functions. This decouples the way an object shadows itself from its reflectance properties, and allows different BRDFs to be substituted at run-time. Here we seek to approximate the transfer signal to reduce computation. To measure approximation error properly, we must know the BRDF. For example, a smooth BRDF weights low-frequency components of transferred radiance more than high-frequency components.

To measure signal errors properly, we include BRDF scaling by encoding the *exiting radiance* transfer matrix at p, M_p. Its component, $M_{p,ij}$, represents the linear influence of source lighting basis function j to exiting radiance basis function i. It can be numerically integrated over light directions s and view directions v over the hemisphere $\boldsymbol{H}=\{(x,y,z) \mid z \geq 0 \text{ and } x^2+y^2+z^2=1\}$ via

$$M_{p,ij} = \int_{v \in \boldsymbol{H}} \int_{s \in \boldsymbol{H}} y_i(v)\ T_p\left(s, y_j(s)\right) B(v,s)\ s_z\ ds\ dv$$

where T_p represents transport effects like shadowing, B is the BRDF, y are the SH basis functions, and s_z is the "cosine" factor (z component of s). For simple shadowing, $T_p = y_j(s)\ q_p(s)$ where $q_p(s)$ is 0 if the object occludes itself in direction s and 1 other-

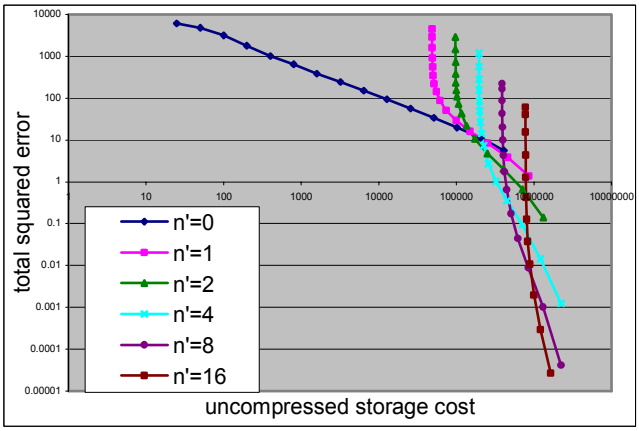


Figure 2: CPCA error analysis using static PCA. Each curve represents how squared error varies with various numbers of clusters (1, 2, 4, …, 16k) using a given number of principal components in each cluster ($n' = 0$, 1, 2, 4, 8, and 16). The signal approximated was a 25D shadowed diffuse transfer vector over a bird statue model from [Sloan *et al.* 2002] having 48668 sample points. 20 VQ iterations were used, followed by PCA in each cluster.

wise. For general transport where lighting is specified in a global frame, $M_p = B\, R_p\, T_p$ where T_p is the glossy transfer matrix defined in [Sloan *et al.* 2002], R_p is an SH rotation aligning p's normal to the z axis and its tangents to x and y, and B is the BRDF matrix

$$B_{ij} = \int_{v \in \boldsymbol{H}} \int_{s \in \boldsymbol{H}} y_i(v)\, y_j(s)\, B(v,s)\, s_z\, ds\, dv$$

R_p is a $N \times N$ rotation matrix; its computation is outlined in [Kautz *et al.* 2002].

We also add a view-independent subsurface scattering term to the transport, precomputed using the hierarchical diffusion approximation of [Jensen and Buhler 2002] but parameterized by the SH basis for lighting. The result affects only the y_0 (constant) basis function of exiting radiance.

3.2 Representing Radiance over the Hemisphere

Exit and transferred radiance at a surface point are actually functions over a hemisphere, not a sphere. For the SH basis, there is complete freedom in evaluating the function on the "opposite" hemisphere when projecting it to the SH basis. Transfer in [Sloan *et al.* 2002] and the formulas above in Section 3.1 implicitly zero the opposite hemisphere by integrating only over the hemisphere. Westin *et al.* [1992] used a reflection technique. It is also possible to use other bases such as Zernike polynomials lifted to the hemisphere [Koenderink *et al.* 1996].

Our approach uses the least-squares optimal projection of the SH basis onto the hemisphere described in the Appendix. The technique represents any SH-bandlimited spherical function restricted to the hemisphere without error. In contrast, zero-hemisphere projection incurs 35% worst-case and 20% average-case RMS error integrated over the hemisphere for all unit-power spherical signals formed by linear combinations of the 5th order SH basis. The odd reflection technique [Westin *et al.* 1992] is even worse. Beyond theoretical results, we also see visibly better accuracy on our experimental objects using optimal projection (see Figure 7).

Given a vector b which projects a hemispherical function into the SH basis by zeroing out the opposite hemisphere, the optimal hemispherical projection is simply $A^{-1}\, b$ where A is defined in the appendix. Therefore, the optimally projected exiting radiance transfer matrix is given by

$$M_p = A^{-1}\, B\, A^{-1}\, R_p\, T_p \qquad (1)$$

projecting first transferred radiance, $R_p\, T_p$, and then exiting radiance. Figure 7 compares results with and without this least-squares "boost" by A^{-1} to reduce error in transferred and exiting radiance.

3.3 Clustered PCA (CPCA) Approximation

We have an n-dimensional signal x_p sampled at points p over a surface. Each x_p represents exiting radiance as a linear operator on a light vector, and takes the form of vectors for diffuse surfaces (e.g., $n=N=25$) or matrices for glossy surfaces (e.g., $n=N^2=625$). To approximate this signal, we partition its samples into a number of *clusters* each of which is approximated by an affine subspace. More precisely, the points in a cluster are approximated by

$$x_p \approx \tilde{x}_p = x_0 + w_p^1\, x_1 + w_p^2\, x_2 + \cdots + w_p^{n'}\, x_{n'}$$

where the $n'+1$ n-vectors $x_0, x_1, \ldots, x_{n'}$ are constant over the cluster and the n' scalar weights $w_p^1, w_p^2, \cdots, w_p^{n'}$ vary for each point p on the surface. To reduce signal dimensionality, $n' \ll n$. The vector x_0 is called the *cluster mean*, and the vectors x_i, $i \geq 1$ are called the *cluster PCA vectors*. Together, the cluster's mean and PCA vectors are called its *representative vectors*.

CPCA (called "VQPCA" in [Kambhatla and Leen 1994] [Kambhatla and Leen 1997] and "local PCA" or "piecewise PCA" in the machine learning literature under the general title of "mixtures of linear subspaces") generalizes PCA (single cluster, $n' > 0$) and VQ (many clusters, $n' = 0$). VQ approximates a signal as a piecewise constant while PCA assumes it is globally linear. CPCA exploits the *local linearity* of our radiance transfer signal by breaking it down into clusters, approximating each with a separate affine subspace.

4. Compressing Surface Signals with CPCA

We review CPCA, beginning with the simplest approach and then describing several enhancements that further reduce error.

4.1 VQ Followed by Static PCA

The simplest CPCA method is to first cluster the points using VQ, and then compute a PCA fit in each of the resulting clusters [Kambhatla and Leen 1994].

VQ Clustering The LBG algorithm [Linde *et al.* 1980] performs the initial clustering. Given a desired number of clusters, the algorithm starts with clusters generated by random points from the signal and then classifies each point into the cluster having minimum distance to its representative. Each cluster representative is then updated to the mean of all its points, and the algorithm iterated until no points are switched or an iteration count is reached.

Per-Cluster PCA We first compute the cluster mean, x_0. We then compute a $m_k \times n$ matrix of residuals after subtracting the mean, $C = [x_{p1}-x_0,\, x_{p2}-x_0,\, \ldots,\, x_{pnk}-x_0]^T$, where m_k is the number of points in cluster k. Computing an SVD yields $C = U\, D\, V^T$ where U and V^T are rotation matrices and D is a diagonal matrix whose elements are sorted in decreasing order. The first n' rows of V (columns of V^T) are the cluster PCA vectors. A point p_j's projection weights (n'-vector w_{p_j}) are given by the first n' columns of row j of UD (they are also given simply by the dot product of $x_{pj}-x_0$ with each of the PCA vectors). This provides a least-squares optimal linear approximation of C from combinations of n' fixed vectors. Total squared error over all cluster points is given by

$$\sum_{j=1}^{m_k} \left\| x_{p_j} - \tilde{x}_{p_j} \right\|^2 = \sum_{i=n'+1}^{n} D_i^2 = \sum_{j=1}^{m_k} \left\| x_{p_j} - x_0 \right\|^2 - \sum_{i=1}^{n'} D_i^2$$

The SVD of C can be directly computed using the LAPACK routine `dgesvd`. To reduce matrix size and so computation, we instead convert to normal form. When $m_k \geq n$, we compute the

$n \times n$ matrix $C^T C$ and its eigenvalues (which are the squares of C's singular values) and eigenvectors (which are equal to C's right singular vectors V^T and thus the needed PCA vectors). When $m_k < n$, we compute the $m_k \times m_k$ matrix CC^T. Its eigenvalues are still the squares of C's singular values, but its eigenvectors are C's left singular vectors, U, from which the right can be derived via $V^T = U^T D^{-1} C$. The LAPACK routine `dsyevx` computes eigenpairs of symmetric matrices like $C^T C$ and CC^T, and saves computation because it can return just the n' eigenpairs having the largest eigenvalues, while `dgesvd` returns *all* singular values.

Experimental Results Figure 2 shows results for this approach on an example diffuse transfer signal (25D) over a bird statue model. Using straight VQ (n'=0), errors are only gradually reduced as storage increases. Increasing the number of PCA vectors per cluster provides an approximation that is worse at small numbers of clusters but eventually crosses below the previous curve as the number of clusters increases.

The graphs use a simple cost metric measuring total storage for the approximated signal:

$$m_p\, n' + m_c\, (n'+1)\, n$$

where m_p is the number of surface samples and m_c is the number of clusters. The first term represents the per-point weight data whereas the second represents the per-cluster representative data. This simple model correlates well with actual rendering cost.

4.2 Iterative PCA

The previous section clusters using distance to the cluster mean $\|x_p - x_0\|^2$ as the classification metric, but as observed in [Kambhatla and Leen 1997], the distance that matters is approximation error, $\|x_p - \tilde{x}_p\|^2$. Iterative PCA [Kambhatla and Leen 1997] exploits this observation by classifying points in the cluster that minimizes approximation error rather than distance to the mean. Also, after every point classification step (instead of only once at the end of the whole clustering process) it computes a PCA of the cluster's current point members to update the affine subspace model.

This approximation error at a point x_p is computed via

$$\|x_p - \tilde{x}_p\|^2 \;=\; \|x_p - x_0\|^2 - \sum_{i=1}^{n'} \left((x_p - x_0) \bullet x_i\right)^2 .$$

To avoid local minima having high error, we introduce additional PCA vectors one by one, from zero to n', and do several iterations (typically 10-20) of the generalized LBG algorithm for that number of vectors before adding another.

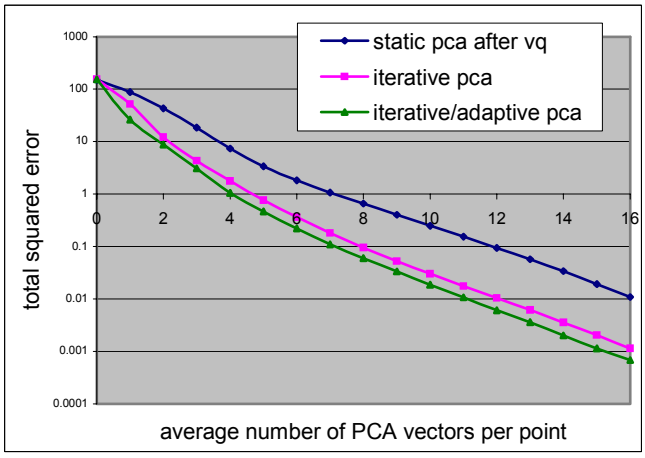


Figure 3: Comparison of error for three CPCA encoding methods. As in Figure 2, the signal encoded is 25D diffuse transfer over a bird model. 256 clusters were used.

Figure 3 and Figure 4 demonstrate the large error reduction from iterative over static PCA. Typically, iterative PCA performs as well as static having 1-4 additional PCA vectors per cluster, but the encoding cost is significantly higher.

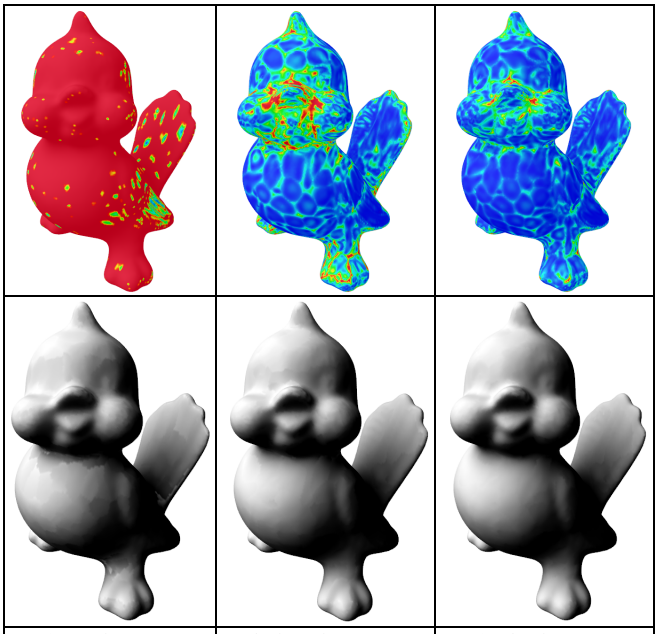
4.3 Per-Cluster Adaptation of Number of PCA Vectors

Neither static nor iterative CPCA distribute error homogenously – some clusters usually have much more error than others. Without increasing the overall amount of per-point data, we can reduce error by allowing clusters with high error to use more PCA vectors and clusters with less error to use fewer. Adaptation like this was used in [Meinicke and Ritter 2001] to avoid local overfitting.

The squared singular value D_i^2 in a cluster represents how much total squared error is reduced by the addition of PCA vector i to that cluster. But clusters do not contain the same number of points; adding an additional PCA vector in a cluster with more points is more expensive than in a cluster with fewer points because an additional weight must be stored per point. So we rank PCA vectors by D_i^2/m_k which represents the rate at which per-point squared error will be reduced by the addition of PCA vector i in cluster k containing m_k points. We sort this quantity in decreasing order over all PCA vectors in all clusters, and add PCA vectors according to this order (greatest error-reduction rate first), until it reaches its total budget of PCA vectors.

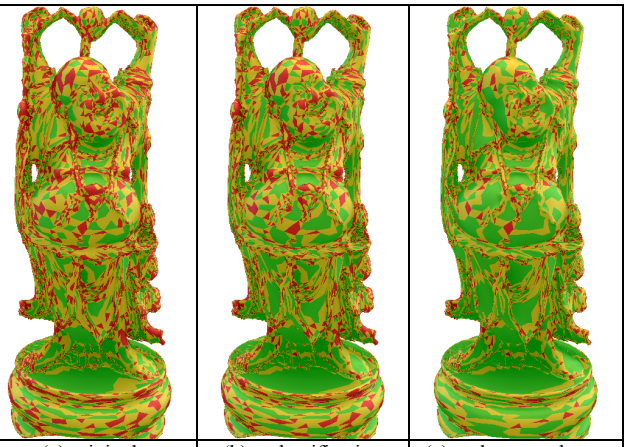
The overall algorithm starts with the CPCA result from the previous section (constant number of PCA vectors per cluster). Additional adaptation iterations then perform the following steps:

1) classify each point to the cluster having minimum approximation error, using the cluster's current n',
2) update cluster representatives using PCA (see Section 4.1),
3) redistribute the number of PCA vectors over all clusters by sorting over D_i^2/m_k and adding vectors (vector i from cluster k) in decreasing order until Σm_k reaches its budget. Record the number of PCA vectors allocated in each cluster.



(a) static [17.7] (b) iterative [4.28] (c) adaptive [2.54]

Figure 4: Per-point error distribution for three CPCA methods. A linear blue-cyan-green-yellow-red error scale is used. Rendered images are shown in the second row. The signal is that for Figure 3 with n'=3. Total squared error of the 25D signal over all 48k vertices is written in brackets.



(a) original mean overdraw: 2.03	(b) reclassification 1.79	(c) recl.+supercluster 1.60

Figure 5: Overdraw reduction using cluster coherence optimization on 256 clusters of a 625D glossy transfer signal with n'=8 and a reclassification error "budget" of 10% of the original error. Triangle color indicates overdraw: red = 3, yellow = 2, and green = 1.

As shown in Figure 3 and Figure 4, adaptation reduces error, typically producing error as low as non-adaptive PCA with an additional vector.

5. Cluster Coherence Optimization

The clusters from the previous section ignore where samples lie on the object's surface – clusters can have ragged boundaries or contain multiple components. This leads to rendering inefficiency because triangles whose vertices are in different clusters are drawn multiple times. For each triangle, this *overdraw* is defined as the number of unique clusters its vertices belong to. Overdraw summed over all mesh triangles represents the number of triangles sent to the graphics hardware for rendering (see details in Section 6). We reduce overdraw with two techniques.

The first technique seeks for each vertex a better cluster that reduces overdraw without significantly increasing approximation error. This greedy search tries reclassifying the vertex's signal in its neighboring vertices' clusters, and computes the resulting overdraw reduction and error increase. The technique then sorts all vertices by overdraw reduction divided by error increase, and reclusters each vertex in decreasing order of this quotient until reaching a given error increase budget, such as 5-10% of the initial error. Vertex reclassification requires recomputation of the quotient for the vertex and its neighbors. Figure 5(b) shows reclassification results.

The second technique, called *superclustering*, allows the graphics hardware to draw a group of clusters as a single unit, It reduces overdraw because triangles straddling clusters from the same supercluster need not be drawn more than once (see Section 6). Superclustering also ensures that primitive batches are large enough to maximize performance; the number of clusters in a supercluster is limited by the number of registers available in the graphics hardware. Unlike reclassification, superclustering does not increase approximation error.

We form superclusters greedily, initializing them to be the clusters, then repeatedly merging neighboring superclusters in order of overdraw reduction. Two superclusters neighbor each other when at least one triangle has vertices from both. Figure 5(c) demonstrates how well greedy superclustering reduces overdraw.

6. Rendering Using CPCA-Encoded Transfer

To shade a glossy surface at point p using CPCA-encoded transfer, we use a modified version of [Kautz *et al.* 2002], via

$$y^T(v_p)\ \left(B\ R_p\ T_p\right)\ l\ \ =\ \ y^T(v_p)\ \left(M_p\right)\ l$$

Here, the column-vector l results of projecting source lighting (in a global coordinate frame) into the SH basis. The matrix T_p converts this source lighting to transferred incident radiance (accounts for self-shadowing and inter-reflection). The rotation matrix R_p aligns the global coordinate system to a local frame defined by p's normal and tangent directions. The BRDF matrix B converts local incident radiance into exit. Finally, y is a column-vector (y^T is a row-vector) of SH basis functions evaluated at the view direction at p, v_p, expressed in the local frame. y and l are N-vectors and B, R, and T are $N\times N$ matrices. A fifth-order SH projection, N=25, is accurate when the lighting and BRDF are low-frequency.

One can compute the source lighting vector l in various ways [Sloan *et al.* 2002]. We can dynamically rotate a predefined environment to simulate rigid rotations of the object. Graphics hardware can sample radiance near the object which is then SH-projected. Simple light sources like circles can be projected analytically. Spatial variation in l captures local lighting effects but complicates the rendering process.

The approach in [Kautz *et al.* 2002] recorded the spatially varying signal $T'_p = R_p\ T_p$ and evaluated the matrix-vector product $f_p=T'_p\ l$ on the CPU. It then evaluated $B'(v_p) = y(v_p)\ B$ using N texture maps indexed by the view vector v_p, and finally computed a dot product of these two vectors. Texture maps B' in [Kautz *et al.* 2002] were evaluated per-vertex on the CPU because the hardware was unable to interpolate 25D vectors f_p over a triangle nor perform the 25D dot product in a pixel shader. Though the latest graphics hardware now makes it possible to interpolate such large vectors, transfer matrices remain too large to be manipulated on the GPU. Fortunately, the affine approximation used by CPCA solves this problem.

Using CPCA, we encode the entire matrix chain M_p converting source lighting to exiting radiance. This produces the approximation

$$\tilde{M}_p = M_0 + w_p^1 M_1 + w_p^2 M_2 \cdots + w_p^{n'} M_{n'}$$

Multiplying $\tilde{M}_p$ by l then yields exiting radiance projected into the SH basis, e_p, via

$$e_p = \tilde{M}_p\, l = \left(M_0\, l\right) + w_p^1 \left(M_1\, l\right) + w_p^2 \left(M_2\, l\right) + \cdots + w_p^{n'} \left(M_{n'}\, l\right)$$

We precompute the matrix/vector products for each cluster on the CPU, resulting in n'+1 fixed N-vectors, and accumulate them as a sum scaled by the per-point weights, w_p^i, on the GPU. For small $n' < N$, this reduces computation and makes the vertex data small enough for vertex shaders on current graphics cards. For example, for N=25 and n'=5, we save more than a factor of 4.

Finally, we evaluate the exiting radiance function at v_p by dotting the vector $y(v_p)$ with e_p. We evaluate y at v_p using a texture map in the same way as [Kautz *et al.* 2002] evaluated $y^T(v_p)\ B$, but we can now perform this evaluation and dot product in a pixel shader.

Diffuse surfaces simplify the computation but CPCA achieves a similar reduction. In this case, $t_p \bullet l$ computes shading where t_p is an N-dimensional transfer vector and l is the lighting's SH projection as before. Using CPCA, we encode t_p as an affine combination of per-cluster representatives and precompute in each cluster the dot product of the light with these vectors. The final shade is a weighted combination of n'+1 scalar values $t_i \bullet l$ which are constant over a cluster, via

$$\tilde{t}_p \bullet l = (t_0 \bullet l) + w_p^1 (t_1 \bullet l) + w_p^2 (t_2 \bullet l) + \cdots + w_p^{n'} (t_{n'} \bullet l)$$

This saves computation when $n' < N$. In fact, the per-vertex computation does not depend on N at all! So we can use higher-order projections (e.g., N=36 up to N=100) as long as the approximation error remains acceptable for small n' (Figure 6). Unlike [Sloan *et al.* 2002], real-time rendering is now possible with such high-order lighting, since the transfer vector is no longer stored on the GPU.

6.1 Non-Square Matrices

M_p need not be square. In an $N_r \times N_l$ matrix, more columns, N_l, provide for greater lighting frequency and thus longer, sharper shadows. More rows, N_r, provide for more specular BRDFs. Interestingly, N_l has little effect on the run-time cost with CPCA, since the transformation of source lighting is done per-cluster to produce vectors whose dimensionality only depends on N_r. Increasing N_l does increase the entropy of the transfer signal, making it harder to encode and likely to require more representatives per cluster.

Non-square transfer matrices are useful in another way. Exiting radiance is a hemispherical function, so we can use the optimal least-squares projection derived in the Appendix to represent M_p. Fifth order optimal projection of the output of M_p can be done with little error using N_r=24 basis functions – one of the 25 bases is nearly redundant (see Appendix).

6.2 Implementation Notes

We first describe the simple case of no superclustering. We decompose the mesh into chunks of geometry for each cluster, where a chunk contains all faces containing at least one vertex from that cluster. Since this also includes vertices from other clusters, we store a per-vertex bit, α_p, indicating whether the vertex p is a cluster member. Pseudocode for rendering is

```
Draw the mesh into the zbuffer only (rgb=0)
Set the blending mode to add
Foreach cluster
    Compute n'+1 per-cluster constants (M_i l or t_i • l) on CPU
    Load per-cluster constants to graphics hardware
    DrawCluster
```

DrawCluster sends the cluster's geometry to the GPU and runs a vertex shader computing the linear combination of the w_p^i with the per-cluster constants. If $\alpha_p = 0$, the w_p^i's are also set to zero so that blending vertices from other clusters does not effect the result. In other words, we blend using a linear partition of unity over each triangle face that straddles multiple clusters.

Generalizing to superclusters is not much more complicated. We compute the per-cluster constants for all clusters in the supercluster and load them into hardware registers. Every vertex in a supercluster records a cluster index, used by the vertex shader as an index register to look up its cluster's constants.

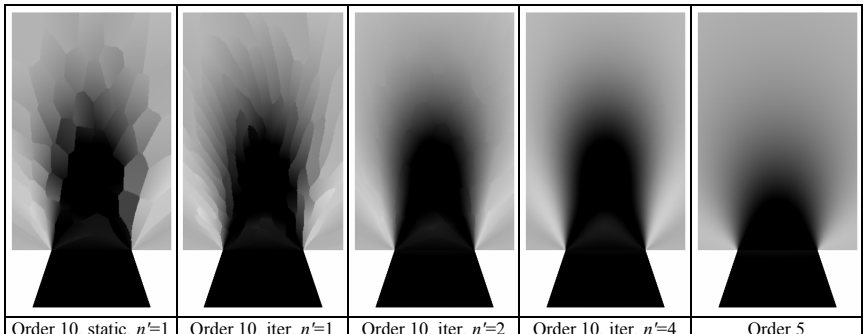
For diffuse transfer, the vertex shader produces the final shaded result. Glossy transfer is more complex – its vertex shader requires normal and tangent vectors to transform the global view vector into the local frame at p to obtain v_p. Rasterization interpolates the resulting view vector v_p and exiting radiance vector e_p over the pixels of each triangle. The pixel shader uses the local view vector to index a map of SH basis functions, $y(v_p)$, and then dots this result with e_p. We use a parabolic hemispherical parameterization [Heidrich and Seidel 1999] for the SH map, sampled at 32×32. Since e_p contains more components than current rasterization hardware can interpolate, we perform three separate passes for glossy transfer – one per color channel.

Diffuse and glossy transfer also differ in their per-cluster state. For each of the n'+1 representative vectors, the per-cluster constant is a scalar color, $t_i \bullet l$, for diffuse transfer regardless of the value of N_l. For glossy transfer, this state is a colored N_r-vector, $M_i l$. Current graphics hardware (ATI 9700, Nvidia GeForce 4) supports ~256 registers accessible by vertex shaders where each register contains 4 channels. For nonadaptive PCA, glossy transfer requires m_s (n' + 1) N_r/4 registers where m_s is the number of clusters per supercluster. This assumes three pass rendering, one per color channel, and packs 4 components of an N_r-vector into each register. Diffuse transfer requires less state: only m_s (n' + 1) registers per supercluster to compute all three color channels by packing an *rgb* color per register.

Though the programmable resources of GPUs have increased greatly, they are not yet sufficient to feasibly render adaptive PCA (Section 4.3), which requires data-dependent looping.

7. Results

Figure 10 compares rendering quality of various transfer encodings on an example bird model with a glossy anisotropic BRDF. We experimentally picked a number of clusters for VQ (n'=0) and a number of representative vectors for pure PCA (m_c=1) such that rendering performance matched that from CPCA with n'=8, m_c=256. For CPCA, we used iterative PCA encoding from Section 4.2. We applied superclustering (Section 4.3) to both VQ and CPCA to the extent permitted by hardware register limits (it is unnecessary for pure PCA since there is only one cluster). Example images, encoding error, and rendering rates appear in the figure for all three methods as well as the uncompressed original. Methods used before in computer graphics [Nishino *et al.* 1999][Wood *et al.* 2000] perform poorly: pure PCA is smooth but has high error; VQ reduces error but has obvious cluster artifacts. Our CPCA result (third column) is very faithful to the uncompressed image on the far right.



Order 10, static, n'=1 SE=15.293	Order 10, iter, n'=1 SE=8.83	Order 10, iter, n'=2 SE=2.23	Order 10, iter, n'=4 SE=0.432	Order 5 Uncompressed

Figure 6: Higher-order lighting for diffuse transfer (simple two-polygon scene). The left four columns show CPCA-encoded results for 10th order lighting (N=100) using various numbers of representatives (n') and m_c=64. The rightmost column shows uncompressed 5th order lighting (N=25) used in [Sloan *et al.* 2002]. Note how shadows are sharpened at higher order and how iterative PCA adapts cluster shapes to the transfer signal better than static PCA (leftmost two columns). CPCA with n'=4 provides an accurate approximation that can be rendered in real-time.

Figure 11 shows the effect on encoding accuracy of varying the per-cluster number of representative vectors (n'). The two rows show results on two models, one smooth (bird, bottom) and one bumpier (Buddha, top). Each column corresponds to a different n'. The signal encoded represents glossy transfer for an anisotropic

BRDF, including multiple bounce interreflections for the Buddha model, but only shadowing for the bird model. With about 8 cluster PCA vectors, we obtain very accurate results that can be rendered quickly. Rendering results using uncompressed transfer data (without CPCA encoding) is shown in the far right column. CPCA speeds up rendering by more than a factor of 10 compared to uncompressed rendering [Sloan *et al.* 2002] with little visual loss.

Model	original	Rec	sc	sc+rec
Buddha	1.93/26.8	1.72/29.6	1.61/29.7	1.48/33.2
Bird	1.25/39.3	1.23/39.3	1.14/45.2	1.138/45.3

Interestingly, though the Buddha model has higher error per transfer sample than the bird model, error is masked by its high-frequency spatial variation. The Buddha's n'=4 result looks quite accurate, whereas the bird's has cluster artifacts visible in the shadowed neck area. Error on the Buddha reveals itself in the neck/chin shadow and the pedestal shadow between the feet.

Figure 9 and Figure 8 show the quality of real-time rendering results achieved by our method. The transfer signal for Figure 9 represents the sum of a diffuse subsurface scattering term and a isotropic glossy term. The result is a realistically rendered Buddha that includes shadowing, translucency, and glossiness effects that respond to changes in lighting or view in real-time.

Figure 8 includes models from [Sloan *et al.* 2002] which could be rendered quickly only by fixing the light with respect to the object (allowing view change), or fixing the view (allowing light movement). We now render these models with visually identical quality in real-time without constraints. For comparison, uncompressed rendering using 25×25 matrices gives a frame rate of 2.9Hz for the head model, and 2.7Hz for the buddha model, a factor of 20× and 16× slower than rendering with CPCA-encoded 16×25 matrices. (For 16×25 matrices, the uncompressed rendering speeds are 5.2Hz and 4.7Hz.) This comparison is fair because 16×25 matrices with least squares optimal projection (Equation (1)) produce results almost indistinguishable from 25×25 matrices with the zero-hemisphere projection (see Figure 7). A Radeon 9800 runs 20% faster with CPCA while uncompressed rendering is 1% faster, showing that CPCA scales well with the GPU.

Table 1 compares encoding results, highlighting the preprocessing times and error values for static PCA (Section 4.1) vs. iterative PCA (Section 4.2). Iterative encoding is expensive, but it often reduces error significantly (see rows for bird model, for example). For the Buddha model, transfer signals tend to be more spatially incoherent, so error reduction from iterative encoding is less dramatic. Using more clusters (increasing m_c) could help matters, but we have done little experimentation with this parameter.

We also measured the effectiveness of cluster coherence optimization (Section 5). Using a 5% error threshold, which has little effect on visual quality, this table shows overdraw/frame rate (in Hz) using reclassification alone ("rec"), superclustering alone ("sc"), and both together ("sc+rec"). Results are for anisotropic glossy transfer ("gloss-anis" from Table 1). We achieve a 15-20% increase in rendering speed on these examples.

model	BRDF	figure	m_p	Transfer $Nr \times N_l$	m_c	n'	Static PCA		Iter PCA		Fps
							cpu	SE	cpu	SE	
Buddha	gloss-iso	11b	49990	16×25	256	6	3m30s	563	1h51m	451	42.8
Buddha	gloss-anis	10	49990	24×25	256	8	6m7s	10996	4h34m	8524	24.2
Buddha	subsurf+ gloss-iso	9	49990	25×25	256	8	6m21s	1992	4h32m	1439	27
bird	gloss-anis	8,10	48688	24×25	256	8	6m34s	898	3h43m	294	45
bird	diffuse	video	48688	1×100	256	8	43s	3.14	3m26s	0.668	227
head	gloss-iso	11a	50060	16×25	256	6	4m20s	78.8	2h12m	43.7	58.5
polys	diffuse	6	58624	1×100	32	4	14s	0.492	3m26s	0.432	294

Table 1: Encoding/performance results. SE means squared approximation error over all vertices. Fps is the rendering performance in Hz. All performance measurements were taken from a 2.2Gz Pentium IV with ATI Radeon 9700.

8. Conclusions and Future Work

We have shown that CPCA-encoded transfer provides real-time rendering of global transport effects for a variety of geometric models and material characteristics, including glossy/anisotropic BRDFs and translucency. Though they depend on prerecorded transfer data over specific models, these effects are new to real-time graphics. CPCA is an effective and very general technique for approximating high-dimensional signals (e.g., transfer matrices) over low-dimensional manifolds (e.g., 3D surfaces). It reduces error better than VQ or PCA for the same storage and yields data granularity in the approximation that better suits GPU implementation. Rather than grouping arbitrarily based on blocks in an image or polygons on a mesh, CPCA adapts cluster size and shape to the nature of the signal. Since the transfer signal is a linear operator on a light vector, representing a cluster containing many samples as a low-dimensional affine subspace not only reduces storage but converts a matrix/vector multiply per point into a weighted combination of a few pre-computed vectors. This is the key to our real-time performance.

In future work, we are interested in using CPCA compression to precompute transfer on deformable models, perhaps by constraining the number of degrees of freedom. We also believe CPCA can be used for surface signals other than radiance transfer of distant source lighting, including simpler signals like surface light fields and more complex ones like transfer for spatially varying illumination. CPCA could be enhanced by an automatic search over the number of clusters variable (m_c), at the cost of additional encoding time. Finally, we are interested in combining our transfer technique, which is specialized for low-frequency lighting, with others handling high-frequency lighting.

Acknowledgments

This research was supported by Microsoft, with additional support from ATI, NVidia, and the NSF ITR grant ACI-0113968. We thank David Thiel for editing the submission video.

References

CABRAL, B, MAX, N, AND SPRINGMEYER, R. 1987. Bidirectional Reflection Functions from Surface Bump Maps, SIGGRAPH 87, 273-281.

CHEN, W-C, BOUGUET, Y-V, CHU, MH, AND GRZESZCZUK, R. 2002. Light Field Mapping: Efficient Representation and Hardware Rendering of Surface Light Fields, SIGGRAPH 2002, 447-456.

GERSHO, A, AND GRAY, R. 1992. Vector Quantization and Signal Compression, Kluwer Academic, Boston, pp. 606-610.

GORTLER, SJ, GRZESZCZUK, R, SZELISKI, R, AND COHEN, M.F. 1996. The Lumigraph, SIGGRAPH 96, 43-54.

HAKURA, Z, LENGYEL, J, AND SNYDER, J. 2000. Parameterized Animation Compression. Eurographics Rendering Workshop, pp.101-112.

HAO, X, BABY, T, VARSHNEY, A. 2003. Interactive Subsurface Scattering for Translucent Meshes, to appear in Symposium on Interactive 3D Graphics.

HEIDRICH, W, SEIDEL H. 1999. Realistic, Hardware-Accelerated Shading and Lighting, SIGGRAPH 99, 171-178.

JENSEN, H, AND BUHLER, J. 2002. A Rapid Hierarchical Rendering Technique for Translucent Material, SIGGRAPH 2002, 576-581.

KAMBHATLA, N, AND LEEN, T. 1994 Fast Non-Linear Dimension Reduction, Advances in Neural Information Processing Systems 6.

KAMBHATLA, N, AND LEEN, T. 1997. Dimension Reduction by Local PCA, Neural Computation, 9, 1493.

KAUTZ, J, SLOAN, P, AND SNYDER J. 2002. Fast, Arbitrary BRDF Shading for Low-Frequency Lighting Using Spherical Harmonics, Eurographics Workshop on Rendering, 291-296.

KOENDERINK, J, VAN DOORN, A, AND STAVRIDI, M. 1996. Bidirectional Reflection Distribution Function Expressed in terms of surface scattering modes, ECCV.

LEHTINEN, J, AND KAUTZ, J. 2003. Matrix Radiance Transfer, to appear in Symposium on Interactive 3D Graphics.

LENSCH, H, KAUTZ, J, GOESELE, M, HEIDRICH, W, AND SEIDEL, H. 2001. Image-Based Reconstruction of Spatially Varying Materials, Proceedings of Eurographics Rendering Workshop, 104-115.

LENSCH, H, GOESCELE, M, BEKAERT, P, KAUTZ, J, MAGNOR, M, LANG, J, SEIDEL, H. 2002. Interactive Rendering of Translucent Objects, Pacific Graphics.

LEVOY, M, AND HANRAHAN, P. 1996. Light Field Rendering, SIGGRAPH 96, August 1996, 31-41

LINDE, Y, BUZO, A, AND GRAY, R. 1980. An algorithm for Vector Quantizer Design, IEEE Transactions on Communication COM-28, 84-95.

MATUSIK, W, PFISTER, H, NGAN, A, BEARDSLEY, P, ZIEGLER, R, AND MCMILLAN L. 2002. Image-Based 3D Photography using Opacity Hulls. SIGGRAPH 02, 427-437.

MEINICKE, P, AND RITTER, H. 2001. Resolution-Based Complexity Control for Gaussian Mixture Models, Neural Computation, 13(2), 453-475.

MILLER, G, RUBIN, S, AND PONCELEN, D. 1998. Lazy Decompression of Surface Light Fields for Pre-computed Global Illumination, In 9th Eurographics Rendering Workshop, June, pp. 281-292.

NISHINO, K, SATO, Y, AND IKEUCHI, K. 1999. Eigen-Texture Method: Appearance Compression based on 3D Model, Proceedings of 1999 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. Fort Collins, CO, June, pp. 618-24 Vol. 1.

RAMAMOORTHI, R, AND HANRAHAN, P. 2001. An Efficient Representation for Irradiance Environment Maps, SIGGRAPH 2001, 497-500.

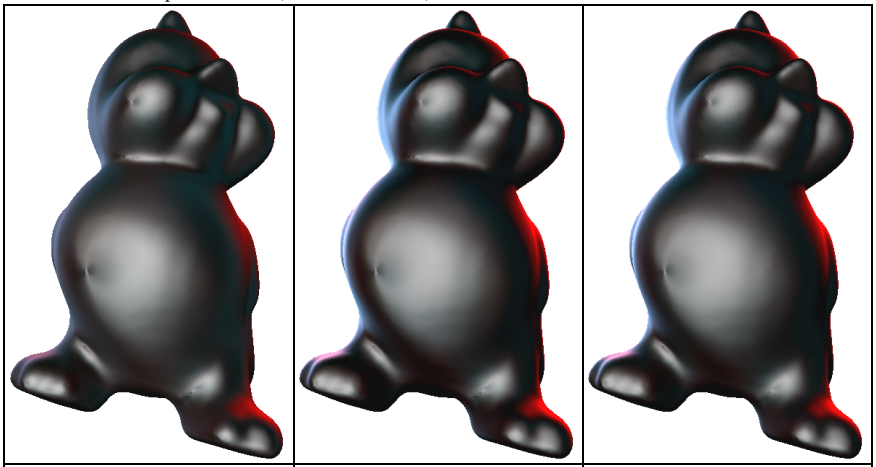
SILLION, F, ARVO, J, WESTIN, S, AND GREENBERG, D. 1991. A Global Illumination Solution for General Reflectance Distributions, SIGGRAPH 91, 187-196.

SLOAN, P., KAUTZ, J, AND SNYDER J. 2002. Precomputed Radiance Transfer for Real-Time Rendering in Dynamic, Low-Frequency Lighting Environments, SIGGRAPH 2002, 527-536.

TIPPING, M, AND BISHOP, C. 1999. Mixtures of Probabilistic Principal Component Analysers, Neural Computation, 11(2), 443-482.

WESTIN, S, ARVO, J, TORRANCE, K. 1992. Predicting Reflectance Functions from Complex Surfaces, SIGGRAPH 92, 255-264.

WOOD, D, AZUMA, D, ALDINGER, K, CURLESS, B, DUCHAMP, T, SALESIN, D, AND STUETZLE, W. 2000. Surface Light Fields for 3D Photography, SIGGRAPH 2000, 287-296.



(a) Zero-hemisphere, 16×25 | (b) Optimal Least-Squares, 16×25 | (c) Original signal, 25×25 (zero-hemisphere from [Sloan et al. 2002])

Figure 7: Projection comparison for glossy transfer matrices. Note the increased fidelity of the optimal least-squares projection (b) compared to zero-hemisphere (a) especially at the silhouettes (blue and red from colored light sources) where the Fresnel factor in the BRDF has high energy. Essentially, using optimal least-squares matches accuracy of a 25×25 matrix from [Sloan *et al.* 2002] via a 16×25 matrix (compare b and c).

9. Appendix: Hemispherical SH Projection

9.1 Least-Squares Optimal Projection

Let $f(s)$ be a function over the hemisphere $s=(x,y,z)$, $s \in \boldsymbol{H}$. We approximate f as a linear combination of SH basis functions $y_i(s)$ restricted to $\boldsymbol{H}$ where these basis functions are no longer orthogonal. So we seek

$$f(s) \approx \Sigma_i \, c_i \, y_i(s)$$

such that this approximation has minimum squared error over $\boldsymbol{H}$. We call this vector c the *least-squares optimal hemispherical projection* of f.

To derive the coefficients c_i of this projection, we minimize squared error

$$E = \int_{\boldsymbol{H}} (f(s) - \Sigma_i \, c_i \, y_i(s))^2 \, ds$$

This is an unconstrained minimization problem with the c_i forming the degrees of freedom. So we take $\partial E/\partial c_k$ and set it to 0:

$$\partial E/\partial c_k = \int_{\boldsymbol{H}} 2 \, (f(s) - \Sigma_i \, c_i \, y_i(s)) \, y_k(s) \, ds = 0$$

$$\Rightarrow \quad \Sigma_i \, c_i \int_{\boldsymbol{H}} y_i(s) \, y_k(s) \, ds = \int_H f(s) \, y_k(s) \, ds$$

This reduces to $Ac=b$ or $c=A^{-1} \, b$ where A is the symmetric matrix

$$A_{ik} = \int_{\boldsymbol{H}} y_i(s) \, y_k(s) \, ds$$

and b is the vector of integrals over the hemisphere of $f(s)$ multiplied by the SH basis functions

$$b_k = \int_{\boldsymbol{H}} f(s) \, y_k(s) \, ds$$

Alternatively, b can be thought of as the standard SH projection of a spherical extension of f which returns 0 when evaluated on the other half of the sphere, called the *zero-hemisphere hemispherical projection*. Note that A can be inverted once regardless of the function $f(s)$. Note also that A is the identity matrix when integrating over the entire sphere.

Readers familiar with biorthogonal bases used for wavelets will find this familiar; $y(s)$ is the primal basis and $A^{-1} \, y(s)$ forms its dual basis.

For 5th order SH projection (25 basis functions), the matrix A is nearly singular – its smallest singular value is 6.59×10^{-6} whereas its largest singular value is 1 (for comparison, the second smallest singular value is 3.10×10^{-4}). We can therefore discard one of the SH basis functions, since at least one is very well approximated as a linear combination of the others when restricted to a single hemisphere. A simple analysis shows that discarding the $l=1,m=0$ SH basis function (i.e., the SH basis function that is linear in z) has the smallest squared error, 1.48×10^{-5}, when approximated as a linear combination of the other basis functions.

9.2 Error Analysis of Various Projections

We first compare the difference between the zero-hemisphere and least-squares optimal projections. The integral, $\int_{\boldsymbol{H}} (\Sigma_i \; c_i \; y_i(s))^2 \, ds$, of the squared value of an approximated function specified by its least-squares optimal coefficient vector c is given by $c^T \, A \, c$. If, as before, b is the zero-hemisphere hemispherical projection of f, then $c = A^{-1} \, b$ is the optimal least-squares hemispherical projection of f. The squared difference between these two projections integrated over $\boldsymbol{H}$ is

$$E_1=(c-b)^T \, A \, (c-b)= \; c^T \, [(A - I)^T \, A \, (A - I)] \, c = c^T \, Q_1 \, c$$

where I is the identity matrix. E_1 attains a maximum value of 0.125 and an average value of 0.0402 over all signals formed by linear combinations of up to 5th order SH basis functions having unit squared integral over the sphere; i.e., over all unit-length 25D vectors c. Worst- and average-case errors are derived as the largest and average singular value of the symmetric matrix Q_1. These are large differences as a fraction of the original unit-length signal; using the RMS norm enlarges them still more via a square root. Optimal projection represents any element of this function space without error.

Another way of restricting the SH basis to the hemisphere ([Westin *et al.* 1992]) is to reflect f's value about z to form a function defined over the whole sphere, via

$$f_{odd}(x,y,z) = \begin{cases} f(x,y,z), & \text{if } z \geq 0 \\ -f(x,y,-z), & \text{otherwise} \end{cases}$$

We can then derive a hemispherical projection of f as the coefficient vector given by the standard SH projection of f_{odd}. We call this method the *odd reflection hemispherical projection*. It is easy to see that a spherical function given by its SH coefficient vector c and then restricted to the $z \geq 0$ hemisphere yields a projection coefficient of $2c_i$ for the *odd* SH basis functions, for which $y_i(x,y,z) = y_i(x,y,-z)$, and 0 for the *even* SH basis functions, for which $y_i(x,y,z) = y_i(x,y,z)$ (all SH basis functions are either odd or even in z).

We analyze reflection projection in terms of squared error in the same way as for zero-hemisphere projection. Since the projection should have a comparable number (i.e, at least 25) of nonzero projection coefficients, we use SH basis functions up to order 8, which includes 28 odd (and thus nonzero) basis functions and 36 even ones, for a total of 64. Using this projection method for the same 5th order function space of interest, represented by the coefficient vector c, yields error E_2 defined as

$$E_2 = c^T [\, (D^* A - I)^T A \;\; (D^* A - I) \,]\, c \;\; = c^T Q_2 c$$

where D^* is a 64×64 diagonal matrix which scales odd basis functions by 2 and even ones by 0, and A is the symmetric matrix defined previously but now for up to 8th order (64×64). Using an SVD of the upper left 25×25 block of the symmetric matrix Q_2, the worst case error over all unit-length 25D vectors c is given by its largest singular value and equals 0.145. The average squared error is given by the average singular value of the upper-left block of Q_2 and equals 0.044. In other words, odd reflection is worse than zero-hemisphere projection in both worst-case and average-case, even though it has more projection coefficients.

A similar analysis can be applied to even reflection, by projecting the even reflection extension of f defined as

$$f_{even}(x,y,z) = \begin{cases} f(x,y,z), & \text{if } z \geq 0 \\ f(x,y,-z), & \text{otherwise} \end{cases}$$

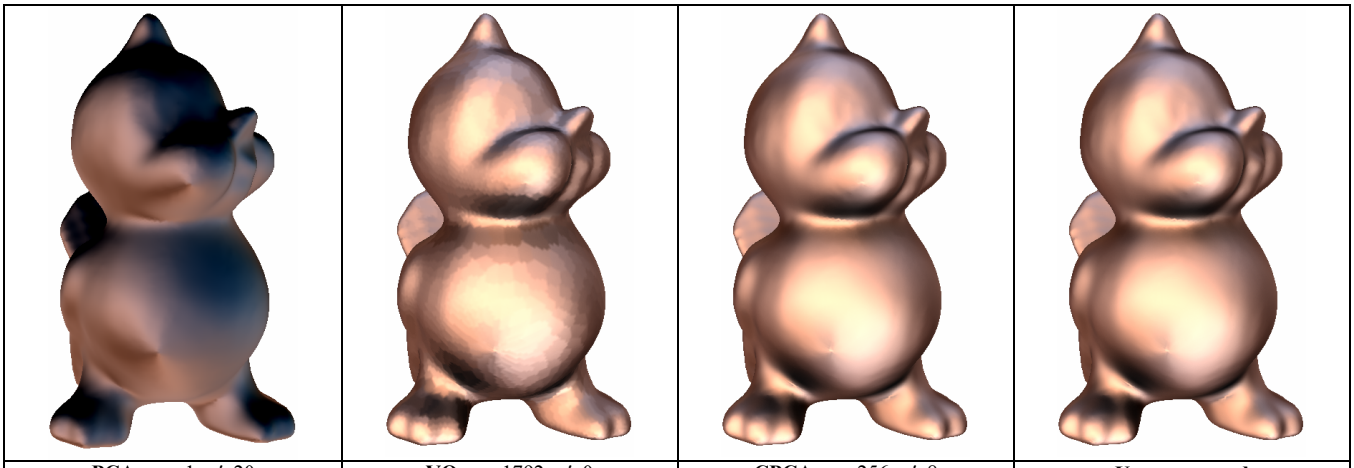
For 7th order SH basis functions, 28 are even and thus produce nonzero coefficients. An error measure for even reflection is identical to E_2 except that its diagonal matrix D^* scales by 2 the even basis functions and zeroes the odd. This projection provides worst-case error over all unit-length signals c of 0.022 and average-case error of 0.0036; still significant but far less than either the zero-hemisphere or odd reflection projections. Interestingly, even reflection using a smaller 5th order basis with only 15 relevant basis functions provides 0.193 worst-case and 0.030 average-case error – better average-case error than zero-hemisphere projection with many fewer coefficients.

(a) Head, 58.5Hz | (b) Buddha, 42.8Hz

Figure 8: Glossy phong models. We get a performance speedup of 16-20x over the method in [Sloan *et al.* 2002] without noticeable degradation, by encoding with CPCA and using least-squares optimal projection to reduce matrix rows from 25 to 16.

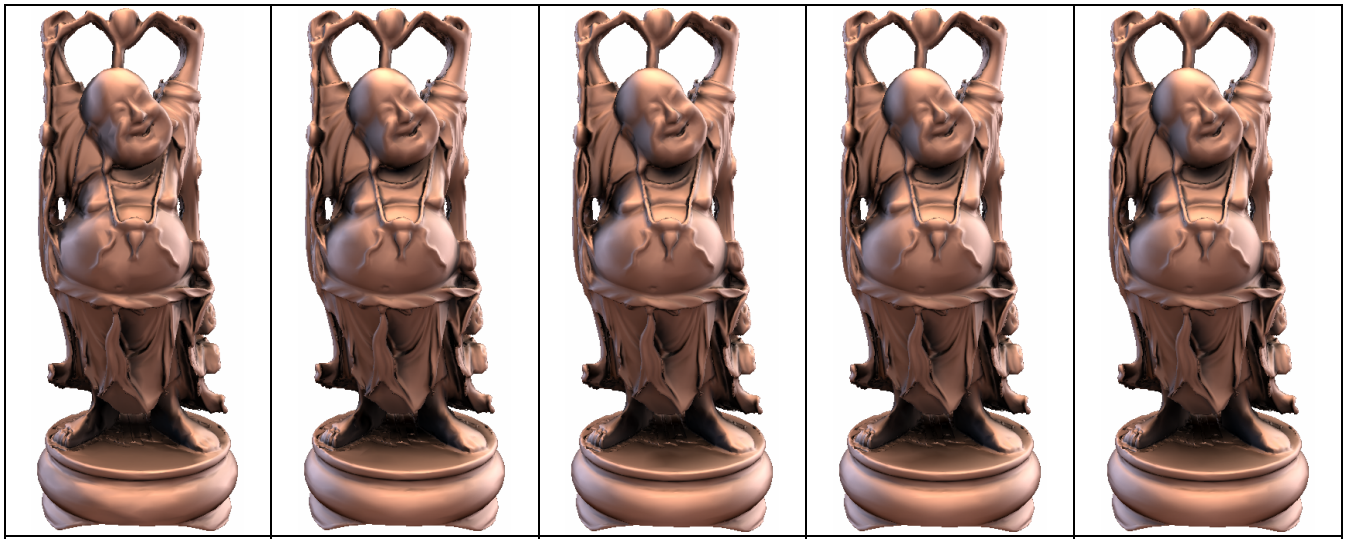
So even reflection is better than zero-hemisphere which in turn is better than odd reflection to approximate functions over the hemisphere. This can be understood because odd reflection produces a discontinuous spherical extension, while even reflection is continuous. Zeroing out the hemisphere is at least continuous for a portion of its basis functions – the odd ones. [Westin *et al.* 1992] also included scaling of the SH basis function by z, so that scaled odd reflection provides a continuous spherical extension. But such scaling yields high approximation error unless f roughly decreases as $z \rightarrow 0$ and $f(x,y,0)=0$. This is not generally true of our exiting radiance functions.

Figure 9: Translucent+glossy Buddha in two lighting environments. These images were rendered at 27Hz.



PCA, m_c =1, n'=20 45.7Hz, SE=101304	**VQ**, m_c=1792, n'=0 41.8Hz, SE=14799	**CPCA**, m_c=256, n'=8 45.5Hz, SE=294.4	**Uncompressed** 3.7Hz, SE=0

Figure 10: VQ vs. PCA vs. CPCA quality results for matched rendering performance. The transfer signal encoded was a 24×25 (600D) glossy transfer matrix for an anisotropic BRDF. CPCA achieves much better visual and quantitative accuracy than VQ and pure PCA. Rendering frame rates and error measurements are listed below each of the four columns. CPCA was encoded using the iterative method of Section 4.2.



40.4Hz, SE= 40353.5	36.4Hz, SE=21077.5	24.2Hz, SE=8524.1	18.7Hz, SE= 4413.01	3.3Hz, SE=0
58.9Hz, SE=9510.75	57.1Hz, SE=2353.09	45.5Hz, SE=294.421	31.9Hz, SE=66.7495	3.7Hz, SE=0
CPCA, n'=2	**CPCA, n'=4**	**CPCA, n'=8**	**CPCA, n'=12**	**uncompressed**

Figure 11: Varying the number of representatives per cluster (n'). Signal is glossy 24×25 transfer with anisotropic BRDF, m_c=256.