Precomputed Local Radiance Transfer for Real-Time Lighting Design

Anders Wang Kristensen
UCSD

Tomas Akenine-Möller
Lund University

Henrik Wann Jensen
UCSD

Abstract

This paper introduces a new method for real-time relighting of scenes illuminated by local light sources. We extend previous work on precomputed radiance transfer for distant lighting to local lighting by introducing the concept of unstructured light clouds. The unstructured light cloud enables a compact representation of local lights in the model and real-time rendering of complex models with full global illumination due to local light sources. We use simplification of lights, and clustered PCA to obtain a compressed representation. When storing only the indirect component of the illumination, we are able to get high quality with only 8–16 lighting coefficients per vertex. Our results demonstrate real-time rendering of scenes with moving lights, dynamic cameras, glossy materials and global illumination.

CR Categories: I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Shading

Keywords: Lighting design, global illumination, precomputed radiance transfer, real-time rendering

1 Introduction

For real-time lighting design, the goals are to be able to add, remove, and change the attributes, including position, of the lights in real time [Gershbein and Hanrahan 2000]. As the lights are moved, the illumination of the scene should be updated in real time with all global illumination effects such as indirect illumination, caustics, subsurface scattering, glossy BRDFs, etc. In addition, it should be possible to move the camera to investigate the lighting from different angles. Unfortunately, these are conflicting goals and all serve to illustrate the fundamental difficulty in computing global illumination for real-time applications, as is apparent in the previous work on this topic.

For lighting design, several deep frame buffer techniques have been proposed. These assume a fixed camera and store geometrical information and surface properties per pixel [Séquin and Smyrl 1989], which allows for rapid regeneration of the image when surface properties or light source properties are changed. Gershbein and Hanrahan exploit graphics hardware to speed up this process greatly [2000]. Brière and Poulin store visibility structures for a fixed camera in order to be able to quickly recompute, e.g., shadows and reflections [1996]. This also makes it possible to support near-interactive changes of smaller pieces of geometry. The fixed camera in the deep frame buffer algorithms does, however, limit the usefulness of such systems.

Figure 1: Our system supports real-time rendering of complex models such as the Sponza Atrium (114,000 triangles at 30–50 fps, including shadow map generation) with global illumination. The user can interactively move the camera and add and move the lights in the model.

Dorsey et al. have presented an impressive system for opera lighting design [1991; 1995]. A major strength is that it enables choreographing of time-dependent lighting. However, the positions of the lights are fixed — only the intensities of lights can change. They render new images as a linear combination of an already rendered set of basis images, as do several others [Nimeroff et al. 1994; Teo et al. 1997]. Similar to the deep frame buffer these techniques also assume a fixed camera.

Dobashi et al. [1995] precomputed the intensity at vertices in a scene from a fixed set of point lights each having a direction intensity distribution represented by spherical harmonics (SH). By assuming diffuse reflection their system enabled camera movement and the ability to alter the intensity and directional properties of the lights. Their system does not allow movement of the lights or the addition of new lights and it becomes costly for scenes with many lights as the illumination at each vertex depends on the basis coefficients for each light.

Wald et al. [2002] have demonstrated a highly optimized global illumination system running on a cluster of PCs. They are able to render scenes with dynamic lighting and diffuse indirect illumination and caustics.

Recently much research has been devoted to rendering static scenes lit by distant illumination including the pioneering work by Sloan et al. [2002; 2003], Liu et al. [2004], Ng et al. [2003; 2004], Ramamoorthi & Hanrahan [2002], and Wang et al. [2004]. These methods assume that the lighting environment is sampled at a single point in the center of the scene and the result is stored as an environment map. Common for all these methods is that they precompute light transport in the scene for many points (vertices) and heavily compress the resulting data. This makes it possible to interactively render scenes with time-varying illumination and in some cases with global illumination.

For non-distant illumination, it is not sufficient to sample the lighting environment only once. Instead, the lighting environment must be sampled at many locations in order to accurately represent incident radiance on the different parts of the model,

Annen et al. [2004] present one way of overcoming this

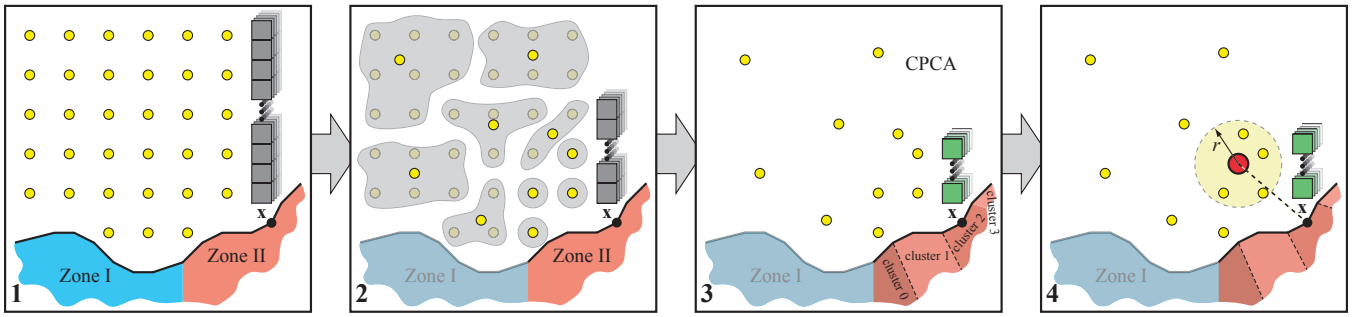

Figure 2: Overview of our real-time lighting design system. It consists of a pipeline of three preprocessing tools, which produce a compressed representation of the lighting that is used in the last step (right) for real-time interactive lighting design: 1) the outgoing radiance due to a set of input lights is computed at each vertex, $\mathbf{x}$. 2) the lights are clustered into sets of similar lights per zone; this results in an unstructured light cloud, 3) the lighting vectors at each vertex are compressed using clustered PCA, and 4) for real-time rendering, a new light (red circle) is inserted into the unstructured light cloud, and the illumination of the vertices is computed in real time.

problem. In addition to evaluating the lighting environment at the center of the scene, they also compute the gradients, which allow them to perform a first-order Taylor approximation. This enables the authors to handle what they call mid-range illumination, which assumes that the visibility in the neighborhood of a point, $\mathbf{x}$, remains static. As the authors point out, this is hardly ever true, but for low-frequency lighting environments, it can serve as a good approximation.

In this paper, we present a full global illumination relighting system with real-time performance that supports local light sources. We assume that the geometry of the scene is static, but light positions and intensities can be changed, lights can be added or removed, materials can be glossy, and the camera is fully dynamic. Thus our system is more flexible and more realistic than the previous ones.

Our algorithm is fundamentally different in that we represent source lighting in a light cloud encompassing the scene instead of using, e.g., environment maps. For each point in this cloud, we precompute a full global illumination solution for each vertex in the scene as if the scene was lit by a unit intensity light placed at this position. The precomputed data is compressed using clustered principal component analysis, and rendering is done directly from the compressed data. An example of a scene rendered using this method is shown in Figure 1.

2 Precomputed Radiance Transfer

Global illumination systems based on precomputed radiance transfer (PRT) [Sloan et al. 2002] solve a reduced version of the rendering equation [Kajiya 1986]:

$$L(\mathbf{x},\vec{\omega}_o) = \int_{\Omega_{2\pi}} L_i(\vec{\omega}_i)\, f_r(\mathbf{x},\vec{\omega}_i,\vec{\omega}_o)\, v(\mathbf{x},\vec{\omega}_i)\, (\vec{\omega}_i\cdot\mathbf{n})\, d\vec{\omega}_i, \quad (1)$$

where L is the reflected radiance in direction $\vec{\omega}_o$ from a point $\mathbf{x}$ with normal $\mathbf{n}$. L_i is the incident radiance from a distant environment, and therefore does not depend on $\mathbf{x}$, f_r is the BRDF, and v is the binary visibility function. For static scenes, a transport operator, $T(\vec{\omega}_i,\vec{\omega}_o)$, can be defined as:

$$T(\vec{\omega}_i,\vec{\omega}_o) = f_r(\vec{\omega}_i,\vec{\omega}_o)\, v(\mathbf{x},\vec{\omega}_i)\, (\vec{\omega}_i\cdot\mathbf{n}), \quad (2)$$

which then reduces Equation 1 to:

$$L(\mathbf{x},\vec{\omega}_o) = \int_{\Omega_{2\pi}} L_i(\vec{\omega}_i)\, T(\vec{\omega}_i,\vec{\omega}_o)\, d\vec{\omega}_i. \quad (3)$$

The power of PRT techniques comes from observing that $T(\vec{\omega}_i,\vec{\omega}_o)$ and $L_i(\vec{\omega}_i)$ can often be represented much more compactly in certain other bases than the standard explicit angular representation. This means that the signals can be represented using only a few terms, and thus, $T \approx \tilde{T} = \sum_k t^k \Psi_k$ and $L_i \approx \tilde{L}_i = \sum l_i^k \Psi_k$, where Ψ_k are the basis functions. Spherical harmonics [Sloan et al. 2002] and Haar wavelets [Ng et al. 2003] are currently the most commonly used bases for PRT. In the case of spherical harmonics, PRT supports efficient rotation [Kautz et al. 2002] of the lighting environment. In particular, for diffuse surfaces, Equation 3 reduces to:

$$L(\mathbf{x},\vec{\omega}_o) = \sum_k l_i^k t^k. \quad (4)$$

This equation is a simple dot product, which can be efficiently evaluated using graphics hardware for a sufficiently small number of terms. For glossy surfaces, the dot product is replaced with a matrix/vector multiplication. The visibility can also be factored out of Equation 3 and represented in the same basis, but then a triple product [Ng et al. 2004] is needed for evaluation.

3 Precomputed Local Radiance Transfer

To account for local lighting in precomputed radiance transfer, it is necessary to solve the full rendering equation [Kajiya 1986]:

$$L(\mathbf{x},\vec{\omega}_o) = \int_{\Omega_{2\pi}} L_i(\mathbf{x},\vec{\omega}_i)\, f_r(\mathbf{x},\vec{\omega}_i,\vec{\omega}_o)\, (\vec{\omega}_i\cdot\mathbf{n})\, d\vec{\omega}_i, \quad (5)$$

The primary difference between this equation and Equation 1 is that the incident radiance, L_i, now is a function of position as well. While this may seem like a small change, it adds significant complexity to the problem as the dimensionality of the incident radiance increases from 2D to 4D (assuming that the position $\mathbf{x}$ is on a 2D surface). A direct replacement of the environment map in precomputed radiance transfer by a light field is not practical as it would be too memory consuming and difficult to manipulate in real time for lighting design.

Our solution to this problem is to discretize the incident lighting into a set of localized lights (as shown in Figure 2). For each light, we precompute the exitant radiance [Wood et al. 2000; Chen et al. 2002] for the vertices in the scene by evaluating Equation 5. For a given vertex, $\mathbf{x}$, the radiance L^j due to light source j is computed as:

$$L^j(\mathbf{x},\vec{\omega}_o) = \int_{\Omega_{2\pi}} L_i^j(\mathbf{x},\vec{\omega}_i)\, f_r(\mathbf{x},\vec{\omega}_i,\vec{\omega}_o)\, (\vec{\omega}_i\cdot\mathbf{n}) d\vec{\omega}_i, \quad (6)$$

where L_i^j is the incident radiance (both direct and indirect) at $\mathbf{x}$ due to light j. For each vertex, we represent the exitant radiance using spherical harmonics. This makes it possible to handle glossy materials. Since spherical harmonics are defined on the sphere, whereas the exitant radiance is hemi-spherical, we use least-squares optimal projection [Sloan et al. 2003].

The spherical harmonics coefficients, X_{ij}, for a given vertex are computed as:

$$X_{ij} = \int_{\Omega_{4\pi}} \tilde{L}^j(\mathbf{x}, \vec{\omega})\, y_i(\vec{\omega})\, d\vec{\omega}. \quad (7)$$

With an n'th order spherical harmonics representation, each vertex stores $n_b = (n+1)^2$ coefficients per light (where $n_b = 1$ for diffuse surfaces). For n_l lights, this means that $X = \{X_{ij}\}$ is an $n_b \times n_l$ matrix. Even for simple models, this is a significant amount of data, and we therefore use a novel multi-stage compression technique to reduce the size of X by several orders of magnitude.

Initially, we cover the space of possible lighting configurations by using a uniform sampling of light sources in the space that will be used for real-time lighting design. Any sampling scheme can be used here to address specific needs of the design problem (e.g. denser sampling along walls etc.). As described in the previous paragraphs, we precompute the exitant radiance representation, X_{ij}, for each vertex. In order to reduce the size of X, we first divide the geometry into a set of discrete *zones*. For each zone, a clustering of the lights is performed such that similar lights are combined into one light (see Section 4.1). This technique is particularly effective at reducing the number of lights far away from the zone. The output of the light clustering step is an unstructured light cloud, which in our examples have up to roughly an order of magnitude fewer lights than the original sampling scheme.

After clustering the lights for each zone, we perform compression of the remaining spherical harmonics coefficients using clustered PCA (similar to Sloan et al. [2003]). Since CPCA is done within each zone, it is able to compress the coefficients efficiently (see Section 5).

For rendering, we allow the user to insert a light at an arbitrary position in the cloud, and approximate the effect of this light from the lights in the cloud. For each PCA cluster, we use this filtered representation of the user-defined light to compute the effect on the PCA basis vectors yielding a set of basis vectors particular to this light configuration. Finally, we combine these new basis vectors with weights specific to a given vertex in the cluster to obtain the final approximation of the exitant radiance, $\mathbf{a}$, in the SH basis. Evaluating the radiance reflected in the direction of the viewer can then be done using a dot product of this vector and a vector of SH basis function evaluated in the direction of $\vec{\omega}_o$:

$$L(\mathbf{x}, \vec{\omega}_o) \approx \mathbf{y}(\vec{\omega}_o)^{\mathrm{T}}\, \mathbf{a}. \quad (8)$$

4 Constructing the Light Cloud

The light cloud defines the region of interest in the model. If a lighting designer is interested in placing lights in a smaller part of the scene then the light cloud can be restricted to that region, which reduces the cost of precomputation.

Determining an optimal representation of the light cloud is extremely difficult. We initially considered using a hierarchical grid, but this structure was problematic in terms of how to smoothly interpolate between different levels of the grid and how to refine it adaptively. Instead, we decided to use a two-stage approach to construct the light cloud. Initially, we make a sufficiently dense uniform sampling of the region of interest (typically hundreds or thousands of lights). This light cloud is then simplified by clustering similar lights.

For the dense light cloud, we evaluate Equation 5 using photon mapping [Jensen 2001]. For each light, a photon map is constructed and the reflected radiance at each vertex is evaluated using the nearest photons. In some scenes, we also use a gathering pass (Monte Carlo ray tracing of the first bounce of indirect lighting at a vertex) to reduce the number of photons needed to get sufficient quality. Finally, we project the outgoing radiance at each vertex onto the spherical harmonic basis to get the coefficients, X_{ij}, described in Equation 7.

4.1 Clustering the Lights

To simplify the light cloud, we apply a bottom-up clustering algorithm that effectively combines several lights with a similar response at the vertices into one single light source. In the multivariate analysis literature, our approach is a *single linkage* clustering technique; cf. [Johnson and Wichern 2002]. To estimate whether two lights, indexed by j and k, are sufficiently similar and can be combined, we compute the following metric:

$$\Delta_{jk} = w_1 ||\mathbf{p}_j - \mathbf{p}_k|| + w_2 \max_c \left[\sqrt{\sum_{i=1}^{n} (X_{ij}^c - X_{ik}^c)^2} \right] + V_{jk}, \quad (9)$$

where w_1 and w_2 are user-specified weights, $\mathbf{p}_j$ and $\mathbf{p}_k$ are the positions of the two lights (this metric assumes that the light cloud contains point lights — directional lights would require an additional penalty based on the direction of the lighting), and X_{ij}^c is described in Equation 7, with c being an index to a particular vertex. V_{jk} is 0.0 if the two light are mutually visible, i.e., there is no geometry lying in between, and 1.0 otherwise. If $\Delta_{jk} < 1$ then the lights are considered "close." Note that for diffuse surfaces, the expression in brackets in Equation 9 simplifies to $||X_{0j}^c - X_{0k}^c||$. The weight w_1 determines how spatially distant lights can be at most in order to be merged, while the weight w_2 controls the maximum difference in the response over all vertices, and thus providing an upper bound of the radiance error induced by a merge. The V_{jk}-term sees to it that only lights that are not obscured can be merged. The goal of the metric is to combine lights that are close to each other and have a similar response at the vertices in the scene. Initially, we considered skipping the distance term, but that ended up oversimplifying those parts of the point cloud that had very similar response at the vertices. This, in turn, made the subsequent interpolation step (Section 6.1) very inaccurate.

One complication of the metric is the weighting of both distances and radiance values. These values are unrelated and w_1 and w_2 should be adjusted accordingly. w_1 should be roughly proportional to the reciprocal of the scene size with a larger w_1 penalizing distant lights more, and w_2 should be related to the reciprocal of the largest intensity in the scene with a larger w_2 penalizing differences in the radiance values more. To avoid using a w_3 value to penalize the visibility term (ensuring that only mutually visible lights are merged), we have normalized w_1 and w_2, such that the maximum error given by the metric should be one. Both w_1 and w_2 are scene dependent: in scenes with complex illumination details, they should be large to preserve the detail of the light cloud, while they can be small in simple scenes. Smaller values lead to a more aggressive reduction of the size of the light cloud.

Now, assume that we have two clusters, G and H, of lights, containing each m_g and m_h local lights. These can be merged into a super-cluster only if $\Delta_{jk} < 1$ for all possible pairs: $(j,k) \in G \times H$. Our clustering approach starts by creating a cluster for each of the original light sources. This is called the *cluster list*. A priority queue is then initialized so that it contains a sorted list of all possible cluster pairs in the cluster list, sorted by Δ_{jk}. The cluster pair with smallest Δ_{jk} is then removed from the queue, and the respective clusters are removed from the cluster list, and all Δ's containing either j or k are removed from the queue. The two clusters are then merged, and the queue updated with the Δ_{sk} for the merged cluster, s, and all clusters, k, in the cluster list. As a final step, the merged cluster is put in the cluster list. New clusters are merged until all pairs in the queue indicate that $\Delta_{jk} \geq 1$. At that point, no more clustering can be done, and the lights in each cluster are replaced by a single light, positioned at the average of all $\mathbf{p}_k$, and each vertex now stores, for each cluster, an average exitant surface radiance vector X_{ij}, computed as the harmonic mean over the lights in each cluster. Alternatively, for higher quality, a new global illumination solution can be computed for each of the new light source positions.

4.2 Zoning

To further optimize the light cloud, we present a new technique where the geometry of the scene is partitioned into different *zones*. For each zone, we perform the above clustering process of the lights separately. This is illustrated in Figure 3. The advantage of this is that lights far away from a zone automatically get clustered much more aggressively. In three dimensions, this makes an even bigger impact — a real example is shown in Figure 4. To split a scene into zones, a simple top-down partitioning algorithm is used, where the box of the scene is split recursively along the longest axis of the box until some stopping criteria is met.

Interestingly, the zoning made it possible to handle much larger scenes in the precomputation step. The reason is that only one zone at a time needs to reside in memory.

5 Compressing Surface Radiance

We reduce storage requirements and increase rendering performance (see Section 6) by compressing the per vertex matrices, X, with $n_p = n_b \times n_l$ elements per color channel before compression.

We use the clustered principal component analysis (CPCA) algorithm as presented by Sloan et al. [2003], who used it for a similar purpose. The algorithm is based on local PCA [Kambhatla and Leen 1997] and uses an iterative clustering strategy to decide how many basis vectors, n, to assign each cluster.

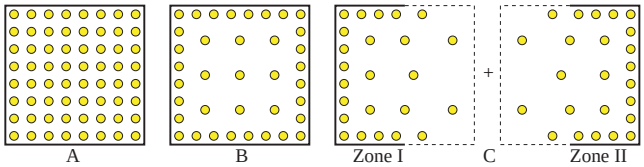

Figure 3: (A) shows an initial dense light cloud computed on a grid. Each yellow circle is a point light, and for each of those, the lighting has been computed at the vertices of the geometry (black box). (B) if all geometry is considered, then the simplification process creates 37 merged cluster lights from the 8×8 original lights. (C) A superior result is obtained when the geometry is split into several zones (in this case, only 2), and clustering is done separately for each zone. 24 clustered lights are obtained in this case.

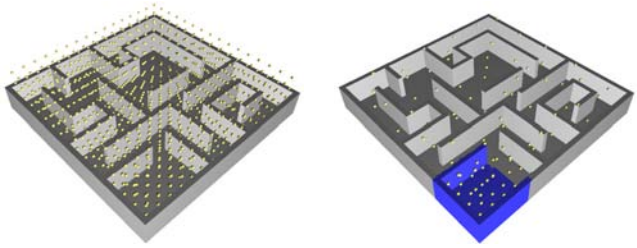

Figure 4: The left image shows all 1024 lights in the original dense light cloud. To the right, these lights have been simplified to only 101 lights for the geometric zone at the lower left (blue). All yellow spheres are light sources. Note that half of the original light sources are located above the maze.

Initial clustering of the vertices is done using k-means (Lloyd relaxation). In each subsequent iteration, we first compute the cluster mean, X^m, using the n_v currently assigned vertices. For each cluster we then construct a $n_v \times n_p$ matrix:

$$M = \begin{bmatrix} X_1^1 - X_1^m & \dots & X_{n_p}^1 - X_{n_p}^m \\ \vdots & \ddots & \vdots \\ X_1^{n_v} - X_1^m & \dots & X_{n_p}^{n_v} - X_{n_p}^m \end{bmatrix}, \qquad (10)$$

and perform a local PCA. We then proceed to assign a PCA vector to the cluster where adding this vector would give the largest per vertex decrease in approximation error. This cluster can be found from M's $(n+1)$'th largest eigenvalue divided by n_v .

After assigning PCA vectors, we redo the clustering of the vertices but this time based on the approximation error computed using the currently assigned number of PCA vectors, n, in each cluster. We typically repeat this step 20–30 times or until convergence (no vertices changed cluster in the previous step).

We continue adding PCA vectors until we reach a desired average number of basis vectors per vertex. These basis vectors, written for clarity here as $n_b \times n_l$ matrices, B_i, allow us to approximate each member of a given cluster as:

$$X \approx X^m + \sum_{i=1}^{n} B_i \, z_i, \qquad (11)$$

with $z_1 \dots z_n$ being particular to a given matrix X.

In order to avoid discontinuities across borders between different clusters, we linearly blend across triangles shared between multiple clusters [Sloan et al. 2003]. When rendering a cluster, the vertices in the cluster are assigned a weight of $\alpha_i = 1$, while all other vertices are assigned $\alpha_i = 0$. Blending is performed for triangles with both types of weights, while the other triangles belonging to the cluster are rendered as usual. We use the same method to blend at the borders between geometric zones.

Discussion Our method allows us to represent direct illumination, indirect illumination, or both, for a given scene. Direct illumination includes high frequency content, which compress poorly using CPCA. An example is shadow boundaries from point lights. In contrast, it is well known empirically that indirect illumination, in the absence of caustics,

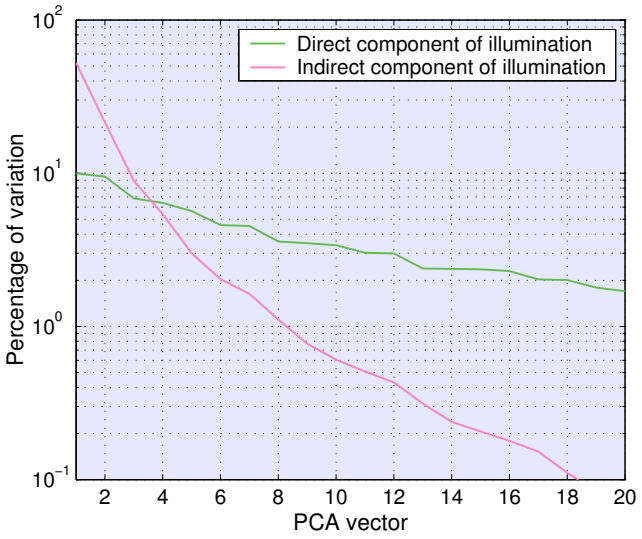

Figure 5: The graph shows how much of the variation in the data is accounted for by each PCA vector. The indirect component of the illumination tends to compress much better than the direct component, as indicated by the much faster falloff. Data is from the back wall of the Cornell box.

varies much more smoothly and thus compresses much better. For this reason, we have found it advantageous, for scenes where we base the illumination on point lights, to represent only the indirect illumination using our method, since this enables us to achieve very high compression ratios using CPCA. We render the direct illumination term using graphics hardware.

On the other hand, when both direct and indirect illumination are compressed using our techniques, other types of light sources are possible. For example, instead of using point lights, one can use diffuse spherical light sources. These provide soft shadows, which in general contain a much lesser amount of high frequency content compared to point lights. As a further alternative, soft shadows can also be rendered using graphics hardware [Assarsson and Akenine-Möller 2003], and thus only indirect illumination could again be compressed.

Figure 5 illustrates the difference in compression ratio that can be achieved when compressing the direct and indirect component separately.

6 Real-Time Relighting

Once the lighting information has been compressed at the vertices, the scene can be relit in real time. The goal is to insert a light source at an arbitrary position in the light cloud, and rapidly reconstruct the exitant radiance at all the vertices in a given direction, $\vec{\omega}_o$, using our compressed representation. This section describes the different steps of this process.

6.1 Adding New Lights

Given a user defined light source configuration consisting of a light source location, $\mathbf{l}$, inside the light cloud and the light's power, Φ_l, the effect of this light source can be approximated using nearby local lights in the light cloud. We organize the local lights in a kd-tree and locate the m nearest lights using a fast search algorithm [Bentley 1975]. For isotropic lights, the distance is the Euclidian distance, while lights

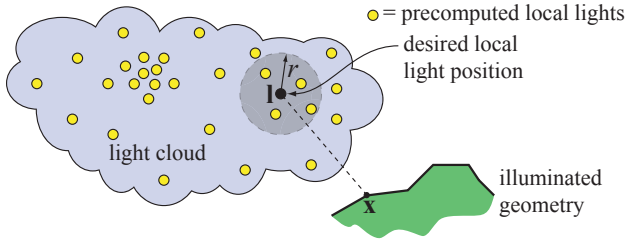

Figure 6: A vertex, $\mathbf{x}$, should be lit by an imaginary isotropic point light positioned at position $\mathbf{l}$. In this case, there are four precomputed local isotropic point lights inside a sphere with radius r. These are used to interpolate the exitant surface radiance at $\mathbf{x}$.

with a directional emission pattern require a 5D kd-tree. The distance from $\mathbf{l}$ to the farthest of these m lights is denoted r, and thus $\mathbf{l}$ and r can be seen as a sphere used for filtering. See Figure 6 for an illustration of this.

With an appropriate filter kernel, a vector $\mathbf{w}_t$ (used in Equation 12), with n_l elements, is created that weighs the closer light sources more than the distant ones. To get smooth interpolation and avoid undesirable "popping" effects when the user moves a light source and a previously used local light source suddenly is located outside the sphere, we set the weight for each precomputed local light to $w_i = (1 - d_i/r)^4$ [Silverman 1986], where d_i is the distance from the precomputed local light to $\mathbf{l}$. When all weights have been computed, normalization is used to ensure that the weights sum to one, and then all weights are multiplied by Φ_l, the power of the light at $\mathbf{l}$.

Locating the nearest light sources based solely on distance can lead to artifacts. Imagine a user defined light source position near a thin wall with light sources from the light cloud located on both sides of the wall. If some of the m nearest lights are on the far side of the wall, these lights may get assigned a non-zero weights, resulting in both sides of the wall being illuminated. To solve this problem, we only use lights that are directly visible from $\mathbf{l}$ (this can be tested efficiently using ray tracing with a few rays).

Although this strategy solves the problem of light "leaking" through walls, it also has the unfortunate side effect of creating potential discontinuities in the illumination as the user moves the light source. The reason is that local lights with non-zero weights inside the sphere can suddenly disappear due to occlusion, while others may appear abruptly with non-zero weights. To avoid this, we smoothly fade out lights over time that previously were assigned a non-zero weight, but that now have become obscured by geometry. Similarly we detect local lights that has recently become visible and smoothly fade them in.

Multiple user-defined light sources are handled by computing a weight vector, $\mathbf{w}_k$, for each light source. The total weight vector can then be computed by summing the weight vectors for all lights, $\mathbf{w}_t = \sum_k \mathbf{w}_k$.

6.2 Computing Exitant Radiance

Given the weights for each light in the cloud, we compute the exitant radiance directly from the compressed representation of X:

$$L(\mathbf{x}, \vec{\omega}_o) \approx \mathbf{y}(\vec{\omega}_o)^{\mathrm{T}} X \mathbf{w}_t \approx \mathbf{y}(\vec{\omega}_o)^{\mathrm{T}} \left(X^m + \sum_{i=1}^{n} B_i z_i \right) \mathbf{w}_t \quad (12)$$

To obtain efficient rendering suitable for graphics hardware, we rewrite Equation 12 as:

$$L(\mathbf{x}, \vec{\omega}_o) \approx \mathbf{y}(\vec{\omega}_o)^{\mathrm{T}} \left(X^m \mathbf{w}_t + \sum_{i=1}^{n} B_i \mathbf{w}_t z_i \right) = \mathbf{y}(\vec{\omega}_o)^{\mathrm{T}} \left(\mathbf{x}^m + \sum_{i=1}^{n} \mathbf{b}_i z_i \right),$$

where $\mathbf{x}^m = X^m \mathbf{w}_t$ and $\mathbf{b}_i = B_i \mathbf{w}_t$. The $\mathbf{x}^m$ and $\mathbf{b}_i$ are constant for each cluster, and only need to be evaluated once before the rendering of a cluster begins.

6.3 Fast Evaluation using Graphics Hardware

We use a multipass rendering algorithm to an offscreen floating-point buffer to obtain images with full global illumination using our methods. In the first pass, we initialize the depth buffer from the current viewpoint, and after this, we disable writing to the depth buffer. For the remaining passes we enable additive blending.

We reconstruct the vector of SH coefficients representing exitant radiance in a vertex program by evaluating:

$$\mathbf{a} = (\mathbf{x}^m + \sum_{i=1}^{n} \mathbf{b}_i z_i)\, \alpha_i. \tag{13}$$

Recall that this is the same $\mathbf{a}$ as being used in Equation 8. We multiply by α_i to ensure that we get correct blending at the borders between clusters and between zones.

The final evaluation of the exitant radiance is done in a fragment program. We precompute and store the SH basis functions in low-resolution cube maps. During rendering, we perform a lookup in the cube maps in the direction $\vec{\omega}_o$, and compute exitant radiance using a dot product with the interpolated vector $\mathbf{a}$ of SH coefficients. This effectively evaluates Equation 8. If direct illumination was not included in the previous pass, the direct illumination is rendered using graphics hardware with point lights, or possibly area/volumetric lights.

Finally, we render our floating-point frame buffer to a screen-sized quadrilateral, and at the same time perform on-the-fly gamma-correction.

7 Results

We have implemented the real-time lighting system on a PC with a 3.4GHz Pentium 4 and a GeForce 6600 graphics card. Our implementation supports isotropic point lights and diffuse sphere lights as the lighting basis. In the following test scenes, we render direct illumination using shadow mapping. Table 1 summarizes various statistics for the scenes discussed in the following.

The first test scene, shown in the top row of Figure 7, demonstrates global illumination in a Cornell box. This small scene contains only diffuse surfaces and renders at 190 frames per second. We used eight zones and 30 PCA clusters, which resulted in 12 coefficients per vertex. In the bottom row, we have created a hole in the wall of the Cornell box to create stronger indirect lighting and demonstrate how a small movement of the light can produce significantly different illumination in the scene. This is particularly noticeable as the light source moves from outside the box through the hole and into the box. For this model we used 10 zones and 18 PCA clusters, which also resulted in 12 coefficients per vertex.

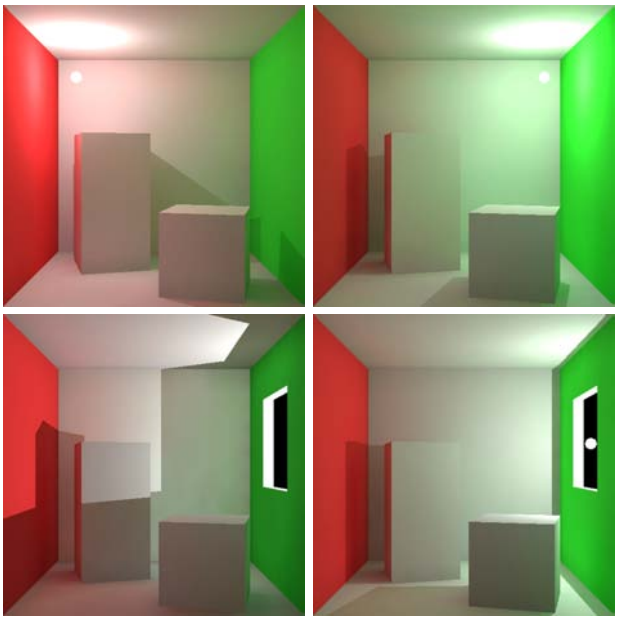

Figure 7: Top row: the Cornell box with different light positions, rendered with full global illumination at 190 fps, while the user moves the light source around. Bottom row: the Cornell box with a hole in the wall that creates strong indirect illumination and discontinuities in the lighting. The left image shows the lighting of the model when the light is outside the window, and the right image shows how the lighting changes as the light moves inside the model. Rendering done at 185 fps, while the light source is being moved.

Figure 8 shows the results of CPCA applied to the Cornell box scene. We are able to achieve good compression rates when compressing the indirect illumination, reducing the number of coefficients stored per vertex from 216 for the uncompressed light cloud to fewer than 16 without noticeable loss of quality.

Figure 9 shows a glossy bunny in an enclosure. The glossy materials reflect each other as well as the indirect illumination from a red and blue wall. For this scene we used a third order spherical harmonics approximation of the glossy Phong model (we used a coefficient of 12). This scene renders at 20 fps, and it has 23 clusters with 48 coefficients per

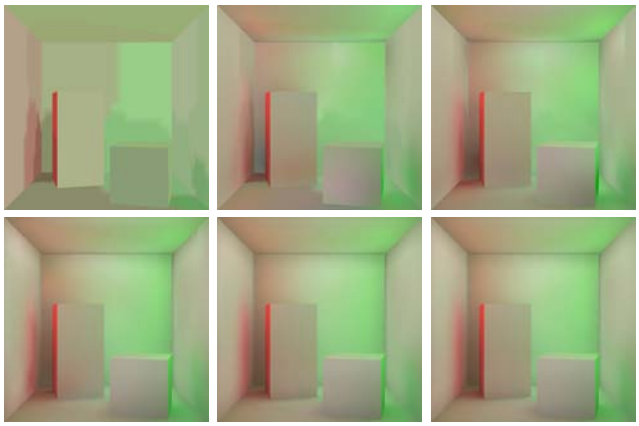

Figure 8: CPCA compression of the irradiance in the Cornell box. Images show (left to right, top to bottom) cluster mean only, reconstruction using two, four, eight, and sixteen PCA vectors. The last image shows the uncompressed result for reference.

Scene	Triangles	Zones	n_b	$w1/w2$	Org. lights	Red. lights	Clusters	Vectors	t_{precomp}	t_{comp}	fps
Cornell	4k	8	1	0.5/20	216	53	30	12	1.5h	1.0h	190
Window	5k	10	1	0.8/5	250	49	18	12	2.0h	1.5h	185
Bunny	15k	19	16	*	64	64	23	48	1.5h	2.0h	20
Maze	43k	16	1	0.5/300	1024	128	191	16	4.0h	1.5h	35
Sponza	114k	247	1	0.3/300	384	127	354	16	4.5h	2.0h	30

Table 1: Sizes, settings, and timings for the scenes in the results section. The global illumination calculations and CPCA compression for the first two test scenes were carried out on a single 3.4GHz Pentium 4, whereas the remaining scenes were computed on a cluster of 32 2.8GHz PCs. The glossy bunny was only compressed with CPCA.

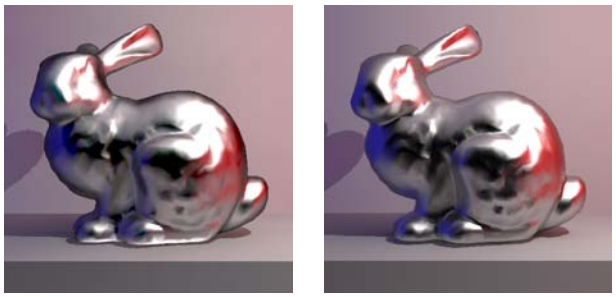

Figure 9: The left image shows a shiny bunny rendered using our method. The bunny demonstrates glossy reflections of both direct and indirect illumination. Exitant radiance is stored using a third order SH approximation (16 basis functions). The right image shows a ray traced reference with per pixel lighting.

vertex.

Figure 10 demonstrates real-time lighting of a complex model, the Sponza Atrium with 114,000 triangles. We used 247 zones, and approximately 384 lights to illuminate this scene. Clustering reduced the number of lights to an average of 127 lights per zone, while CPCA compressed the number of coefficients per vertex down to 16. This scene renders at 30 fps.

Our last test scene in Figure 11 demonstrates global illumination in a maze. It has complex discontinuities in the visibility and strong indirect illumination. We initially illuminated this scene using 1024 lights; clustering reduced the number of lights to roughly 130 per zone, while CPCA further reduced this to roughly 16 coefficients per vertex. This scene has 42450 triangles and it renders at 35 fps.

8 Conclusion and Future Work

We have presented a system for real-time lighting design based on precomputed local radiance transfer. Contrary to existing methods, our system supports dynamic local light sources, camera movement and glossy materials with full global illumination. Key to making our method practical is the concept of the unstructured light cloud. We use this sparse representation of local light sources in combination with clustered PCA in a novel two-stage compression technique to reduce storage and increase rendering speed. We have demonstrated that our system handles indirect illumination efficiently for models with more than 100,000 triangles. Thus, our algorithm offers a practical alternative to the traditional ambient term often used in real-time graphics.

For future work, spotlights with emission described by spherical harmonics could be incorporated into our framework. Furthermore, hardware-accelerated soft shadows could be rendered using graphics hardware, and the indirect illumination term using our system.

Figure 10: Lighting in the complex Sponza model. Images show full global illumination (top), direct illumination (bottom left), and indirect illumination (bottom right).

Acknowledgements

This research was supported by a Sloan Fellowship and the National Science Foundation under Grant No. 0305399.

Tomas was supported by the Hans Werthén foundation, Carl Tryggers foundation, Ernhold Lundströms foundation, and the Swedish Foundation for Strategic Research.

References

Annen, T., Kautz, J., Durand, F., and Seidel, H.-P.

Figure 11: The maze has complex discontinuities in the visibility and strong indirect lighting. The images show (from left to right) the maze illuminated from above, the light has been placed inside the maze, direct component of illumination, indirect component of illumination and full global illumination.

2004. Spherical Harmonic Gradients for Mid-Range Illumination. In *Eurographics Symposium on Rendering*, 331–336.

Assarsson, U., and Akenine-Möller, T. 2003. A Geometry-Based Soft Shadow Volume Algorithm using Graphics Hardware. *ACM Transactions on Graphics, 22*, 3, 511–520.

Bentley, J. L. 1975. Multidimensional Binary Search Trees Used for Associative Searching. *Communications of the ACM 18*, 9, 509–517.

Brière, N., and Poulin, P. 1996. Hierarchical View-Dependent Structures for Interactive Scene Manipulation. In *Proceedings of ACM SIGGRAPH 96*, ACM Press / ACM SIGGRAPH, New York, ACM, 83–90.

Chen, W.-C., Bouguet, J.-Y., Chu, M. H., and Grzeszczuk, R. 2002. Light Field Mapping: Efficient Representation and Hardware Rendering of Surface Light Fields. *ACM Transactions on Graphics 21*, 3, 447–456.

Dobashi, Y., Kaneda, K., Nakatani, H., and Yamashita, H. 1995. A Quick Rendering Method Using Basis Functions for Interactive Lighting Design. *Computer Graphics Forum 14*, 3, 229–240.

Dorsey, J. O., Sillion, F. X., and Greenberg, D. P. 1991. Design and Simulation of Opera Lighting and Projection Effects. In *Computer Graphics (Proceedings of ACM SIGGRAPH 91)*, ACM, 41–50.

Dorsey, J., Arvo, J., and Greenberg, D. 1995. Interactive Design of Complex Time-Dependent Lighting. *IEEE Computer Graphics & Applications 15*, 2, 26–36.

Gershbein, R., and Hanrahan, P. 2000. A Fast Relighting Engine for Interactive Cinematic Lighting Design. In *Proceedings of ACM SIGGRAPH 2000*, ACM Press / ACM SIGGRAPH, New York, ACM, 353–358.

Jensen, H. W. 2001. *Realistic Image Synthesis using Photon Mapping*. AK Peters Ltd.

Johnson, R. A., and Wichern, D. W. 2002. *Applied Multivariate Statistical Analysis*. Prentice Hall.

Kajiya, J. T. 1986. The Rendering Equation. In *Computer Graphics (Proceedings of ACM SIGGRAPH 86)*, ACM, 143–150.

Kambhatla, N., and Leen, T. K. 1997. Dimension Reduction by Local Principal Component Analysis. *Neural Comput. 9*, 7, 1493–1516.

Kautz, J., Sloan, P.-P., and Snyder, J. 2002. Fast, Arbitrary BRDF Shading for Low-Frequency Lighting using Spherical Harmonics. In *Proceedings of the 13th Eurographics workshop on Rendering*, 291–296.

Liu, X., Sloan, P.-P., Shum, H.-Y., and Snyder, J. 2004. All-Frequency Precomputed Radiance Transfer for Glossy Objects. In *Eurographics Symposium on Rendering*.

Ng, R., Ramamoorthi, R., and Hanrahan, P. 2003. All-Frequency Shadows using Non-Linear Wavelet Lighting Approximation. *ACM Transactions on Graphics 22*, 3, 376–381.

Ng, R., Ramamoorthi, R., and Hanrahan, P. 2004. Triple Product Wavelet Integrals for All-Frequency Relighting. *ACM Transactions on Graphics 23*, 3, 477–487.

Nimeroff, J. S., Simoncelli, E., and Dorsey, J. 1994. Efficient Re-rendering of Naturally Illuminated Environments. In *Fifth Eurographics Workshop on Rendering*, 359–373.

Ramamoorthi, R., and Hanrahan, P. 2002. Frequency Space Environment Map Rendering. *ACM Transactions on Graphics 21*, 3, 517–526.

Séquin, C. H., and Smyrl, E. K. 1989. Parameterized Ray Tracing. In *Computer Graphics (Proceedings of ACM SIGGRAPH 89)*, ACM, 307–314.

Silverman, B. 1986. *Density Estimation for Statistics and Data Analysis*. Chapman and Hall.

Sloan, P.-P., Kautz, J., and Snyder, J. 2002. Precomputed Radiance Transfer for Real-Time Rendering in Dynamic, Low-Frequency Lighting Environments. *ACM Transactions on Graphics 22*, 3, 527–536.

Sloan, P.-P., Hall, J., Hart, J., and Snyder, J. 2003. Clustered Principal Components for Precomputed Radiance Transfer. *ACM Transactions on Graphics 22*, 3, 382–391.

Teo, P. C., Simoncelli, E. P., and Heeger, D. J. 1997. Efficient Linear Re-rendering for Interactive Lighting Design. Tech. Rep. STAN-CS-TN-97-60.

Wald, I., Kollig, T., Benthin, C., Keller, A., and Slusallek, P. 2002. Interactive Global Illumination using Fast Ray Tracing. In *13th Eurographics Workshop on Rendering*, 15–24.

Wang, R., Tran, J., and Luebke, D. 2004. All-Frequency Relighting of Non-Diffuse Objects using Separable BRDF Approximation. In *Eurographics Symposium on Rendering*.

Wood, D. N., Azuma, D. I., Aldinger, K., Curless, B., Duchamp, T., Salesin, D. H., and Stuetzle, W. 2000. Surface Light Fields for 3D Photography. In *Proceedings of ACM SIGGRAPH 2000*, ACM Press / ACM SIGGRAPH, New York, ACM, 287–296.