Eurographics Symposium on Rendering 2008
Steve Marschner and Michael Wimmer
(Guest Editors)

Volume 27 (2008), Number 4

Tensor Clustering for Rendering Many-Light Animations

Miloš Hašan[1], Edgar Velázquez-Armendáriz[1], Fabio Pellacini[2] and Kavita Bala[1]

[1]Cornell University [2]Dartmouth College

Images from animations rendered with our algorithm, with environment illumination and multiple-bounce indirect lighting converted into 65,536 lights. By sparsely sampling the light-surface interactions and amortizing over time, we can render each frame in a few seconds, using only 300-500 GPU shadow map evaluations per frame.

Abstract

Rendering animations of scenes with deformable objects, camera motion, and complex illumination, including indirect lighting and arbitrary shading, is a long-standing challenge. Prior work has shown that complex lighting can be accurately approximated by a large collection of point lights. In this formulation, rendering of animation sequences becomes the problem of efficiently shading many surface samples from many lights across several frames. This paper presents a tensor formulation of the animated many-light problem, where each element of the tensor expresses the contribution of one light to one pixel in one frame. We sparsely sample rows and columns of the tensor, and introduce a clustering algorithm to select a small number of representative lights to efficiently approximate the animation. Our algorithm achieves efficiency by reusing representatives across frames, while minimizing temporal flicker. We demonstrate our algorithm in a variety of scenes that include deformable objects, complex illumination and arbitrary shading and show that a surprisingly small number of representative lights is sufficient for high quality rendering. We believe out algorithm will find practical use in applications that require fast previews of complex animation.

Categories and Subject Descriptors (according to ACM CCS): I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism

1. Introduction

Rendering animations of deformable scenes with global illumination, high-frequency environment maps, and arbitrary shading is a long-standing challenge. Fast rendering algorithms exist for these problems, but usually resort to sparse sampling, or significantly restrict the problem being solved. On the other hand, PRT-based methods need a large amount of precomputation and memory, particularly for high frequency illumination and non-diffuse materials. Even well established off-line algorithms like irradiance caching and photon mapping can exhibit temporal artifacts when not specifically designed to be temporally stable. [HPB07] introduced a matrix-based formulation for rendering scenes with global illumination, many direct lights and environment maps, by sparsely sampling the rows and columns of the matrix to reconstruct images in a few seconds. However, using this approach across multiple frames leads to temporal flickering while not taking advantage of temporal coherence. Simple temporal filtering is not a viable solution because it

does not maintain quality; it blurs high-frequency lighting and features like indirect shadows.

In this paper we present a new, flexible approach to rendering animations of these scenes. We introduce a *tensor formulation* of the problem, defined over pixels, lights and time. Indirect illumination, environment map lighting, and direct lights are all converted to point lights in this framework. Moreover, any geometry and materials can be used, since the framework simply treats them as arbitrary values in the input tensor. Our algorithm clusters the columns of the tensor, where the information needed to choose a good clustering is obtained by sampling slices of the tensor. We also introduce *pixel mappings* that establish the relation of pixels (and their associated 3D surface points) for pairs of consecutive frames. In summary, our main contributions are:

- A flexible tensor formulation of the animation rendering problem that can incorporate global illumination, environment lighting, complex geometry and arbitrary shading,
- a time-aware clustering technique that takes advantage of temporal coherence and addresses temporal flicker,
- a pixel mapping algorithm that allows for reusing shading over frames.

We believe our approach can be practically useful in cinematic rendering applications, where an entire shot can be previewed with high quality.

2. Previous work

The body of research in rendering global and environment illumination is very large. We split the related approaches into four broad categories: those that convert the global illumination problem into many point lights, those that use a ray-tracing approach combined with sparse caching, those based on hierarchical finite elements, and finally those that are based on precomputation of light transport.

Many-light approaches. Instant radiosity [Kel97] and its variants approximate the global illumination problem by a number of virtual point lights (VPLs) that are usually generated by particle tracing and rendered on the GPU with visibility handled by shadow mapping. Most of these approaches are directed at interactive performance, with some sacrifices in quality and generality – the main reason is that at most a few dozen lights can be rendered at interactive rates, while a highly accurate solution usually needs thousands. Furthermore, the particles usually need to be retraced in every frame, which can lead to some temporal flicker (using quasi-Monte Carlo sampling can reduce this problem).

[DS05] and [DS06] present a fast one-bounce indirect illumination solution, and also a method to generate one-bounce indirect lights on the GPU; however, shadows from VPLs are ignored. [LSK*07] is a promising technique for one-bounce indirect illumination that manages a set of 256 VPLs as illumination changes, without retracing new particles. However, for interactive performance, it uses sparse interleaved sampling of the image, and re-renders at most 10 shadow maps per frame, which is only correct for a static scene. In contrast, our solution computes 300-500 shadow maps per frame, providing an accurate approximation to a reference solution with as many as 65,536 lights.

Lightcuts [WFA*05] and multi-dimensional lightcuts [WABG06] are high-quality many-light algorithms. The latter also handles motion blur and participating media, but does not amortize over multiple frames. These CPU-based methods use ray-tracing for visibility and take several minutes per frame, as opposed to several seconds in our case.

[HPB07] introduces an algorithm for single-frame rendering that sparsely samples rows and columns of the lighting matrix to reconstruct the image. This work is the closest to our research. But [HPB07] is a randomized algorithm that produces different error in every run. It does not explicitly handle temporal coherence, and using it directly to render animations leads to visible flicker. In this paper we focus on extending this method with two goals: first, decreasing the temporal flicker, and second, gaining better performance by amortizing shading over time.

Sparse sampling approaches. A well known technique in this area is the Render Cache and variants [WDP99, TPWG02, BWG03]. These methods generally have image artifacts when the scene changes; these artifacts gradually clear as new samples arrive. Temporally coherent extensions to (ir)radiance caching have been explored in [SKDM05] and more recently in [GBP07]. However, both of these approaches, despite using sparse sampling, take several minutes per frame on moderately complex scenes. Photon mapping [Jen01] has been extended to temporally coherent rendering, for example in [MTAS01, DBMS02, HDMS03]; however, these have some image artifacts from directly visualizing the photon map (without final gather).

Hierarchical radiosity. A technique for interactive scene editing with global illumination was presented in [DS97]; this work takes advantage of temporal coherence, but it might not scale to high-complexity geometry and materials. Recently, some promising approaches based on hierarchical radiosity have been presented that avoid the computation of visibility, notably [Bun05] and [DSDD07]. These methods provide very fast solutions; however, their drawback is that the antiradiance (i.e. light that has to be subtracted to correct for ignoring visibility) is a highly directional quantity, so a large number of directional samples is needed for each patch to maintain accuracy. This makes these methods hard to scale beyond a few thousand patches without compromising accuracy of indirect and environment shadows.

Precomputed transfer. PRT-style techniques, e.g. [SKS02, NRH04, HPB06], are based on extensive precomputation to render static or dynamic scenes under distant or indirect illumination. Some related methods [RWS*06] deal specifically with soft shadows. These techniques usually do not have temporal aliasing problems.

However, they usually have to give up some flexibility in order to be able to precompute the solution, e.g. dynamic deformations or indirect illumination. Our goal is to support animations with all of these effects without precomputation.

3. Overview

The **many-light problem** involves computing the contribution of n lights to m surface points. The value of each of these contributions can be expressed as the product of a material term, a light term and a visibility term, which is 1 when the sample is visible from the light and 0 otherwise. This problem is very important, not only for rendering with many direct light sources, but also because conversion to many lights can be used to handle environment and indirect illumination. This is achieved by approximating the environment map by many directional lights, and the indirect illumination by many "indirect lights" with cosine emission [Kel97].

However, it is a difficult problem for two reasons. First, the visibility term is costly to compute. Second, a naive algorithm that evaluates all interactions would take time $O(mn)$. Several approaches attempt to solve either of these two problems. [HPB07] attacks the first problem by using fast GPU shadow-mapping to evaluate the visibility, and the second one by sampling only a small subset of rows and columns from the $m \times n$ matrix of light-surface interactions.

Some randomized algorithms (including [HPB07]) demonstrate temporal flicker when used directly for animations, because they make different approximations in different frames. Apart from eliminating flicker, we would like to take advantage of the temporal coherence present in most animations to actually *decrease* the necessary number of samples, which is the goal of this paper. Simple ideas like applying a temporal filter do not work very well, in fact making error even more perceptible by smoothing it out over several frames. Therefore, we propose an algorithm that is time-aware by design.

Proposed algorithm. We formalize the problem as a large $m \times n \times f$ tensor $\mathbf{A}$ (3D array) of surface-light interactions over multiple frames. The columns of the tensor correspond to images rendered with a single light in a single frame, while the rows express the contribution of all lights to a single pixel in a single frame. As observed by [HPB07], the rows and columns can be computed very efficiently on the GPU, using shadow-mapping as the visibility technique.

The key idea of our algorithm (see Figure 2) is to cluster the columns of the tensor, pick a single representative in each cluster, and approximate all other columns in the cluster by reusing the information from the representative. To find a good column clustering, we use the information from the sampled rows. We introduce a clustering algorithm that repeatedly splits clusters in lights or time, producing results with very low temporal flicker using surprisingly low numbers of sampled rows and columns (300 or 500 per frame on average, instead of the 1000 or more required in [HPB07]).

To take advantage of temporal coherence, we need to be able to compute a representative column in some frame k_0 and use it in some other frame, k_1. There are two simple solutions to this problem, none of which work very well. The first one is to take a column rendered with a single light in frame k_0, and use this image *as is* in frame k_1, at most scaling the pixel intensities by a constant. Clearly, if the camera or some objects move between the frames, this will lead to ghosting artifacts. The other simple solution would be to compute illumination only at mesh vertices, and linearly interpolate it across polygons. This is suboptimal, since for high-quality results with arbitrary shaders we would like to compute illumination per pixel.

Therefore, we propose a pixel mapping technique that allows for an image rendered in one frame to be reprojected and used in another frame. For each pair of adjacent frames, we compute a forward and backward mapping between the view samples based on nearest neighbors. Note that this does not mean we compute sparse keyframes and extrapolate to in-between frames; the mappings are applied separately for each cluster that spans multiple frames. This lets us reuse shading between frames practically without noticeable reprojection artifacts.

In summary, a high-level overview of our algorithm is:

- Compute pixel mappings between frames (CPU),
- sample r rows of $\mathbf{A}$ per frame (GPU),
- partition the reduced columns into c clusters (CPU),
- compute a single representative in each cluster (GPU),
- use the representative to approximate the missing columns in the cluster, by appropriately scaling intensities and applying the pixel mappings (GPU).

4. Algorithm

We now formalize the problem we are solving and explain our algorithm in detail.

4.1. Tensor Notation

For the purposes of this paper, we define a tensor $\mathbf{T}$ to be a 3-dimensional array of real or RGB values of size $m \times n \times f$. An alternative way to view $\mathbf{T}$ is as a sequence of matrices $\mathbf{T}_1, ..., \mathbf{T}_f$. Throughout this paper we use Matlab-style notation for tensors and matrices. We will refer to a single element of $\mathbf{T}$ by $\mathbf{T}(i,j,k)$. We can specify subtensors by using sets of indices instead of single indices, e.g. $\mathbf{T}(I,j,k)$ or $\mathbf{T}(i,J,K)$. We use a colon to specify the set of all indices for a particular dimension, like $\mathbf{T}(:,:,k)$ or $\mathbf{T}(i,j,:)$. Our matrices and tensors are assumed to be in column-major format.

We define the *columns* of a tensor $\mathbf{T}$ to be the vectors of type $\mathbf{T}(:,j,k)$, and the *rows* of $\mathbf{T}$ to be vectors of type

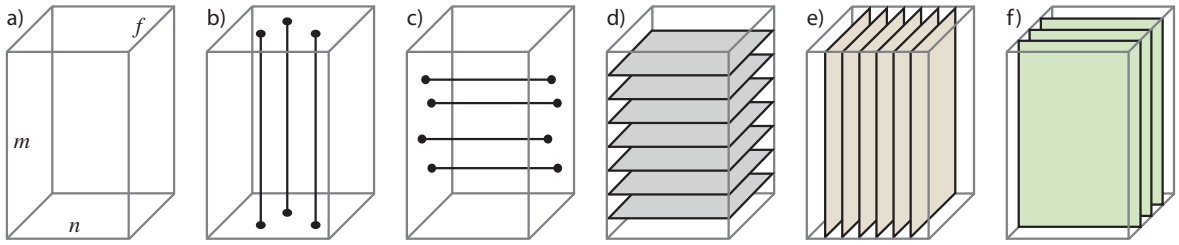

Figure 1: *A tensor is just a 3D array (a). Columns (b), rows (c), slices (d), slabs (e), frames (f).*

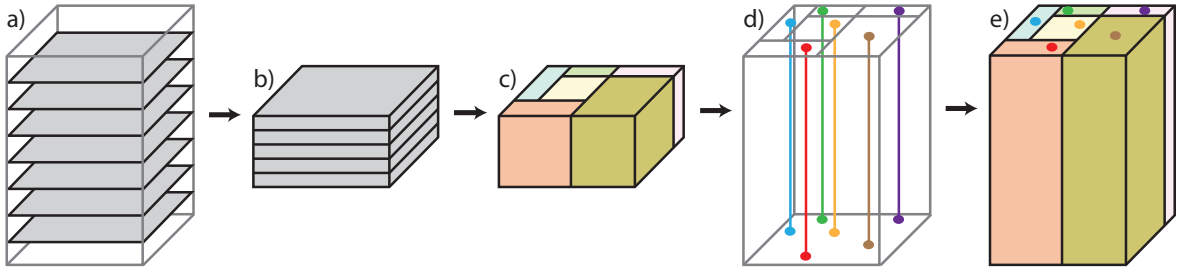

Figure 2: *Our algorithm first computes r slices (rf rows) of the tensor (a). These are assembled into a reduced tensor (b), whose columns are then clustered (c). The same clustering is then applied to the full tensor, where a representative column is rendered in each cluster (d). The missing columns in each cluster are filled in by scaling and mapping the representative (e).*

Symbol	Description	Type
m	Number of pixels	scalar
n	Number of lights	scalar
f	Number of frames	scalar
r	Number of slices (rows)	scalar
c	Number of clusters (columns)	scalar
$\mathbf{A}$	Full lighting tensor	$m \times n \times f$ RGBs
$\mathbf{R}$	Reduced tensor	$r \times n \times f$ scalars
$\mathbf{S}$	Sparse scaling tensor	$c \times n \times f$ RGBs
Rec	Reconstruction matrix	$c \times f$ RGBs
C_t	A cluster of light-frame pairs	set of pairs
$\mathbf{x}_p$	Normalized column of $\mathbf{R}$	$r \times 1$ scalars
w_p	Norm of a column of $\mathbf{R}$	scalar

Table 1: *Summary of the notation used in the paper.*

$\mathbf{T}(i,:,k)$. In other words, these are the rows and columns of the matrices $\mathbf{T}_k$. We also define *slices* of $\mathbf{T}$ to be the matrices of type $\mathbf{T}(i,:,:)$, and *slabs* to be of type $\mathbf{T}(:,j,:)$. We will use the Matlab function $\mathsf{permute}(\mathbf{T},d_1,d_2,d_3)$, which is a generalized transpose operator; it permutes the dimensions of the tensor into the order specified by d_1,d_2,d_3. We will also use the function $\mathsf{reshape}(\mathbf{T},\mathrm{dims})$, which changes the dimensions of the tensor while keeping the same data, and the function $\mathsf{sum}(\mathbf{T},d)$ which sums the tensor across dimension d, producing a matrix.

4.2. A Tensor Formulation of the Problem

We are looking for an efficient algorithm to shade m image pixels from n lights over f frames. We can view the problem as a 3-dimensional tensor $\mathbf{A}$ of size $m \times n \times f$, where the value of an element $\mathbf{A}(i,j,k)$ is the contribution of light j to pixel i in frame k.

To compute the animation, we would like to accumulate, for each frame, the contribution of all lights to each pixel. This can be expressed as the sum of the tensor $\mathbf{A}$ along the light dimension (the second dimension):

$$\mathbf{S}_\mathbf{A} = \mathsf{sum}(\mathbf{A},2)$$

A brute-force approach to compute $\mathbf{S}_\mathbf{A}$ would simply evaluate all the elements of the tensor, taking time $O(mnf)$. For typical high-quality rendering values of m and n in the thousands to millions, and hundreds of frames, this is unacceptably slow. We need an efficient way to reconstruct an approximate result by using only a small subset of the tensor elements.

As noted by [HPB07], the sampling pattern can make a significant difference: while a random pattern requires a ray-tracer, full rows and columns of a lighting matrix can be efficiently computed on the GPU using shadow mapping. This observation is still valid in the tensor setting.

4.3. Clustering

The high-level idea is to partition the columns $\mathbf{A}(:,j,k)$ into c clusters, compute a single representative column in each cluster, and approximate all other columns in the cluster by using information from the single sampled column. Note that the algorithm does not simply cluster lights or frames, but instead the *light-frame pairs* (LFPs) (j,k). (We will also refer to LFPs by a single index, going through all pairs.)

The quality of the result strongly depends on the chosen column clustering, which we optimize by using the information from the sampled slices. We define the *reduced tensor* $\mathbf{R}$ to be the tensor of the sampled slices stacked on top of each other. The size of $\mathbf{R}$ is $r \times n \times f$. In Matlab notation, if I is the set of indices of the selected rows, we can write:

$$\mathbf{R} = \mathbf{A}(I,:,:)$$

In essence, $\mathbf{R}$ is equivalent to the complete tensor $\mathbf{A}$, except for a smaller image size. The idea is to optimize the clustering on $\mathbf{R}$ since this is numerically feasible, and then use the resulting cluster assignment for $\mathbf{A}$. The assumption is that most of the significant structure of $\mathbf{A}$ is preserved in $\mathbf{R}$.

4.3.1. Clustering objective function

We call the columns of the reduced tensor $\mathbf{R}(:,j,k)$ the *reduced columns*. We use the clustering objective proposed by [HPB07] for single-frame rendering with many lights, where it was chosen to minimize the expected error of the algorithm. The key modification we introduce in this paper is that the objective function operates on light-frame pairs instead of just lights. Let $p = (j,k)$ and $q = (j',k')$ denote indices of LFPs. We define the cost of a clustering as the sum of the costs of all clusters C_t:

$$\sum_{t=1}^{c} cost(C_t) = \sum_{t=1}^{c} \sum_{p,q \in C_t} d(p,q)$$

Here we define the "distance" between two light-frame pairs p and q as

$$d(p,q) = w_p w_q \|\mathbf{x}_p - \mathbf{x}_q\|^2$$

where $w_p = \|\mathbf{R}(:,j,k)\|$ is the norm of the reduced column corresponding to the light-frame pair p,

$$\mathbf{x}_p = \mathbf{R}(:,j,k)/\|\mathbf{R}(:,j,k)\|$$

is the normalized reduced column, and analogously for w_q and $\mathbf{x}_q$. We can think of w_p as the *weight* of the light-frame pair p, since it approximates the energy that it contributes to the animation. We call $\mathbf{x}_p$ the *information vector* of light-frame pair p, since it abstracts away from the energy and expresses the "kind" of the light's contribution to the frame.

Intuitively, this clustering objective has very desirable properties: it tends to separate light-frame pairs with large weights or with different information vectors. An interesting property of the objective is that, in some sense, it treats time and lights equivalently, even though they model very different physical phenomena. This is because the fundamental approximation in our approach is the replacement of some column by a (possibly rescaled) copy of a representative column. This contributes some error (in the squared 2-norm sense) to the total error of the animation. If we ignore the (hard to quantify but usually negligible) error introduced by pixel mappings, then this contribution is the same, regardless of whether the representative is in the same frame as the column being approximated, or in a different frame.

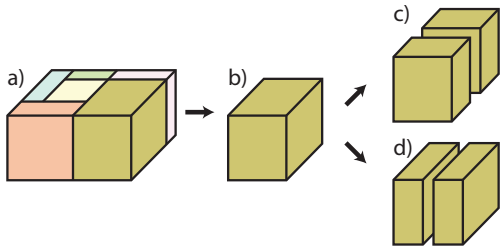

Figure 3: *The key operation of our clustering. Given a current clustering (a), we find the cluster with the highest cost (b). We consider splitting it in time (c) or lights (d), and pick the split that results in the lowest-cost clustering.*

4.3.2. Finding the Clustering

The above discussion leaves two issues unaddressed. First, the above clustering problem is NP-hard even if all weights are 1 [dlVKKR02], so we can only hope for an efficient heuristic to find a reasonably good clustering. Second, the objective function has no explicit constraint enforcing temporal smoothness; therefore even the optimal clustering might not be perceptually pleasing, even though its numerical error might be low. In fact, we also tried running the original clustering from [HPB07] on the columns of the matrix

$$\mathsf{reshape}(\mathbf{R}, r, nf)$$

which essentially treats the light-frame pairs as completely disconnected lights, without any knowledge about time. We found that the result, while converging to the correct answer, still flickered too much to be usable.

The key idea of our clustering algorithm that solves these issues is to consider only the subset of possible clusterings where all clusters are "rectangular", i.e. the set of LFPs in each cluster is always a Cartesian product of a set of lights and a set of frames. We can visualize the clustering as an $n \times f$ array that specifies for each light-frame pair the cluster it belongs to. Figure 3 shows example clusterings, visualizing cluster memberships as colors.

Top-down splitting approach. This is the key component of our technique. The idea is to find the cluster with the

highest cost, and split it in lights or in time; repeat this step until the desired number of clusters is reached. The question of whether to split in lights or time is resolved by trying both alternatives and choosing the one with the lower cost of the resulting clustering. (One could also consider weighting light splits differently from time splits, to force more subdivision in lights than in time or vice versa.)

Time split. When splitting in time, the situation is easier since there is already a natural ordering on the frames – their actual order in the animation. Therefore, we consider splitting the cluster at all positions and choose the one with the best objective function value.

Light split. There exists no natural ordering of lights, so the above idea does not apply to light splitting trivially. We instead find an ordering of lights by projecting them onto a line in rf-dimensional space as follows. Let C be the cluster we are splitting; let $C = J \times K$ where J is a set of light indices and K is a set of frame indices. Let

$$\mathbf{Y} = \mathbf{X}(:, J, K)$$

be the $r \times |J| \times |K|$ sub-tensor of $\mathbf{X}$ corresponding to C. The slabs $\mathbf{Y}(:, j, :)$ combine all information vectors of light j over the set of frames K. We reshape these slabs into vectors and project them onto a line. The orientation of the line is determined by picking two lights in the cluster with probability proportional to their total weights over frames in K, and taking the line they lie on (in rf-dimensional space). The ordering on the line will be used to find the split as in the time case. Essentially, we're slicing the cluster by a hyperplane in "information space". The advantage of using this approach is that it is invariant with respect to the initial ordering of the lights; moreover, with a high probability, two elements with large weights will be separated by the split.

Clustering initialization. Here we initialize the clustering to a small number of clusters (e.g. 100), by partitioning only in lights (i.e., each cluster will be a Cartesian product of all frames with a set of lights). The algorithm we use is the multi-set sampling algorithm described in [HPB07], applied to the matrix $\mathbf{R}'$ defined as:

$$\mathbf{R}' = \mathsf{reshape}(\mathsf{permute}(R, 1, 3, 2), rf, n)$$

Intuitively, we take the slabs $\mathbf{R}(:, j, :)$ for each light j, unfold them into vectors that form the columns of $\mathbf{R}'$, and then run the sampling algorithm on these vectors. As a further optimization, we use random projection to decrease the dimensionality of the clustering problem. This approximation is acceptable, since the goal of the initialization phase is just to provide a reasonable starting point for the splitting phase; it does not have to be very optimal itself.

After the initialization phase, the algorithm splits either on time or lights as described above. In fact, we could skip the initialization phase, and only run the splitting phase. This has two disadvantages: first, it is slower, since splitting large clusters takes longer; second, the algorithm might decide to split in time too early, which could result in a visible temporal cut (even though it might be the more optimal move in terms of objective function minimization).

4.4. Pixel Mappings

We compute, for each frame, a *pixel mapping* to its predecessor and successor frames. The forward mapping $\mathbf{M}(k \to k+1)$ specifies for each pixel in frame $k+1$ its *nearest neighbor* among pixels in frame k. The definition of the backward mapping $\mathbf{M}(k+1 \to k)$ is analogous. These mappings can be interpreted as sparse matrices of size $m \times m$. For example, if $\mathbf{x}$ is an $m \times 1$ vector storing an image rendered with a single light in frame k, then $\mathbf{M}(k \to k+1)\mathbf{x}$ is an approximate image with the same light in frame $k+1$. This simple improvement leads to very good results, and the artifacts due to the mappings are almost invisible in the final animations. More details of how we compute the mappings can be found in Section 5.

4.5. Final Reconstruction

Given a clustering of light-frame pairs, we choose a representative LFP $p = (j,k)$ in each cluster; we choose the one with the largest w_p. (In [HPB07] the representative was chosen randomly, with probability proportional to reduced column norm. This had the effect of making the algorithm unbiased in Monte Carlo sense. In the temporal setting, the algorithm can no longer be made unbiased, so we use a deterministic choice.) Then we render the corresponding column $\mathbf{A}(:, j, k)$ on the GPU and approximate all other columns in the cluster by scaled, mapped copies of the computed representative. Let the index of the column we are approximating be $q = (j', k')$; then the representative will be scaled by w_q / w_p, and mapped by the repeated application of the precomputed pixel mappings to get from frame k to k', i.e. by applying the product of sparse matrices $M(k'-1 \to k') \ldots M(k \to k+1)$ (here assuming $k < k'$).

Of course, we do not in reality reconstruct the full tensor $\mathbf{A}$, since we are only interested in its sum across lights. We bypass the creation of the full tensor by defining a *scaling tensor* $\mathbf{S}$ of size $c \times n \times f$, such that the value $\mathbf{S}(i, j, k)$ is the scaling of the representative of cluster i, when it is used to approximate the column for LFP (j,k). Clearly, $\mathbf{S}$ is very sparse, since every representative will only be used to approximate LFPs in its own cluster. Therefore, we can easily construct it in a compact representation. From the scaling tensor we get a *reconstruction matrix* $\mathbf{Rec} = \mathsf{sum}(\mathbf{S}, 2)$ of size $c \times f$, which specifies for each representative the scaling factor that it contributes to each frame.

5. Implementation Details

Creating the lights. In scenes lit by an environment map, we convert the map into directional lights by simple uniform

stratified sampling. We could certainly use a more advanced importance sampling approach, but the point is that our algorithm will automatically find the important lights and cluster them less aggresively than weak lights. For indirect illumination, we cover the surfaces of the scene uniformly with gather samples as in [HPB06]. These samples are firmly tied to their respective triangles, so they move as their underlying geometry moves and deforms. Again, some importance sampling technique could be substituted, but our algorithm essentially provides the light selection automatically. For the sun-sky model, we turn the sun (a small strong area light) into 16 directional lights, and sample the sky uniformly.

Shading the gather samples. To turn the gather samples into indirect lights, we need to determine the illumination on them. We shade the gather samples with direct illumination from point lights. To add multiple bounces of indirect illumination, we use a photon-mapping pre-pass. We shoot 10,000 photons into the scene without storing the first hit into the photon map. These photons do not directly become indirect lights; instead, we perform density estimation on the gather samples that are tied to the surfaces.

This approach also works for multiple bounces from the sky (used in the temple), and could be also used for area lights or other kinds of lights for which direct illumination computation is non-trivial; the only difference being that the first hit photons would also be stored in the photon map. Note that this photon mapping approach requires implementing a ray-tracer matching the GPU renderer. Photon mapping is a standard approach to add multiple bounces to final-gather algorithms [Jen01, HPB06].

Clustering and cost computation. Computing the cost of a cluster might seem to be quadratic in the number of its elements; however, it can be easily optimized by noting that

$$d(p,q) = w_p w_q \|\mathbf{x}_p - \mathbf{x}_q\|^2 = w_p(\mathbf{x}_p^T\mathbf{x}_p + \mathbf{x}_q^T\mathbf{x}_q - 2\mathbf{x}_p^T\mathbf{x}_q)w_q$$

so the matrix of distances between every pair of elements in the cluster is low-rank and can be manipulated in a compact form. The clustering problem is quite large-scale; in our case, the reduced tensor has $100 \times 65536 \times 40$ elements, which amounts to 1GB of data in 32-bit precision. Our implementation does all operations on the tensor in-place, to not waste memory. Note that the original RGB data is converted to scalars by taking 2-norms of the RGB triples; however, the per-channel norms of the columns need to be preserved, since they are needed for the reconstruction.

Pixel mapping computation. To compute the mappings, we render the "deep frame-buffers", i.e. world-space positions, normals, and diffuse reflectances for all pixels in each frame. We treat each sample as a 9-vector, and use a kd-tree for nearest neighbor queries. The mappings are stored as textures of indices on the GPU.

Row computation. We pick the rows by splitting the image into 10×10 bins and picking a pixel in the center of each bin. The surface samples for those pixels are in the deep frame-buffer. A row is computed on the GPU by rendering a cube shadow map at the surface sample position, and evaluating the shader for that sample and all lights.

Column computation and reconstruction. Column computation is a standard direct illumination evaluation for a single point light on the GPU, using a single shadow map for spotlights and directional lights, and a cube shadow map for indirect and omni-directional lights. Once a representative column is computed, the mappings are applied on the GPU, so that slow readbacks can be avoided until the whole sequence is rendered.

Anti-aliasing. Even though all our shading is computed with one sample per pixel, we apply an edge-respecting anti-aliasing post-process to our images that uses an idea similar to one presented in [HPB06]. A deep frame-buffer (DFB) of positions, normals and diffuse albedos (a total of 9 values per pixel) is rendered at 3x the standard resolution, and the shading is "upsampled" to this higher resolution. This is done by finding, for each 9-dimensional sample in the high-resolution DFB, the nearest neighbor among the 9-dimensional samples in the low-resolution DFB, and copying the image color from that pixel. The nearest neighbor search is done only in 3x3 neighborhoods of the low-res DFB, instead of full search using an acceleration structure. This idea is orthogonal to our main algorithm, and could be used with any approach where shading is sufficiently expensive to amortize the cost of the anti-aliasing pass.

6. Results and Discussion

Figure 4 shows images from five animation sequences, each of which is 10 seconds long at 20 frames per second (200 frames total). We refer the reader to the video for the final rendered animations. We also provide a video with a side-by-side comparison of our algorithm to the brute-force method (which renders all 64k lights exhaustively). For the more complex scenes, where rendering the brute-force solution is prohibitively expensive, we only show a few frames instead of the full 20 frames per second. Timings of the different components of our system for all rendered sequences are summarized in Table 2 and were measured on a desktop with a Core 2 Duo 2.6 GHz processor, 2GB of RAM and an Nvidia 8800 GTX GPU.

We apply the algorithm to five 40-frame chunks of the animation separately, and stitch together the results; this is to limit the memory needed to hold the reduced tensor. We have not found objectionable artifacts at the boundaries of the chunks, but we could also consider overlapping the chunks over several frames and linearly blending them.

One main strength of our algorithm is its generality with respect to types of lights, materials and geometry supported. Moreover, all scene elements can be dynamically deforming. The included rendered sequences show camera motion, rigid

	Iris	Still	Bunny	Temple	Horses
#triangles	51k	107k	869k	2.1m	8.2m
rows/frame	100	100	100	100	100
columns/frame	200	400	200	200	200
shadowmaps/frame	**300**	**500**	**300**	**300**	**300**
indirect lights	-	7.53	-	7.31	-
mappings	2.88	1.94	1.18	2.66	3.65
exploration	0.98	0.79	1.02	4.95	15.58
clustering	0.99	1.41	1.19	0.78	1.00
reconstruction	0.55	4.17	1.39	7.60	26.69
anti-alias	1.50	1.25	1.25	2.59	2.23
total	**6.90**	**17.09**	**6.03**	**26.40**	**49.15**
brute-force	126	301	395	2013	13221
speed-up	18x	18x	65x	76x	269x

Table 2: *All timings are per-frame averages in seconds. Columns/frame is the average number of representatives computed per frame. The stages of the algorithm are, respectively: indirect light creation and shading (only done for scenes with indirect illumination), pixel mapping computation, GPU row rendering (exploration), clustering, representative rendering on the GPU, and final anti-aliasing pass. The brute-force renders all 65,536 lights on the GPU using shadow mapping for visibility.*

and deformable geometry, arbitrary shaders, indirect illumination, HDR environment lighting, and a sun-sky model.

Iris. This is a relatively simple scene with 51k polygons; however, it shows that our algorithm can correctly deal with deforming geometry. It also clusters the more pronounced lights less aggressively than weak ones, resulting in nicely preserved high-frequency shadows. Textures are correctly handled by the pixel mapping algorithm. While similar results could be rendered by importance-sampling the map with a technique such as [ODJ04], our approach is much more general in that it works with an abstract tensor, not specific to environment map sampling.

Still life. This indoor scene shows multiple bounces of indirect illumination. We distribute 65,536 gather samples over all surfaces of the scene, determine their shading by a combination of direct lighting and photon density estimation, and treat them as 65,536 cosine-emission lights illuminating the scene. Even though our algorithm knows nothing about indirect illumination, i.e. it is just sampling elements of an abstract tensor, we obtain a high quality solution. Note that the highly glossy bunny close to indirect lights is a particularly hard case, and some flicker can still be seen. However, this case would also be tricky for any other approach except for pure Monte Carlo ray-tracing.

Furry bunny. The bunny scene contains high-complexity fur geometry (50,000 hairs, 869k polygons). The detailed geometry and shadows would be tricky to handle by a sparse sampling technique such as (ir)radiance caching. The fur uses the Kajiya-Kay hair shader, further demonstrating the ability of our approach to handle arbitrary materials and procedural shaders. Some flicker on the bunny can be seen; this is due to geometric aliasing on the thin hairs, not due to lighting.

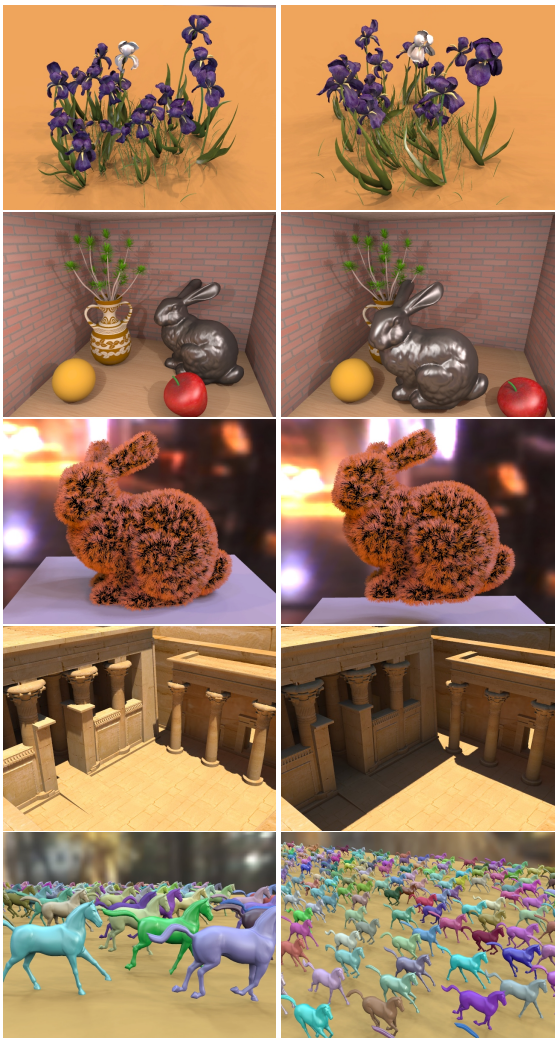

Figure 4: *A few frames from our result animations – Iris, Still life, Furry bunny, Temple, and Horses.*

Temple. This scene with over 2 million polygons shows the scalability of our technique to complex geometry. It also shows a combination of high and low frequency distant illumination (sun and sky), together with multiple bounces of indirect illumination. Note that while the scene is static, we still compute and apply the pixel mappings (even though they will be an identity mapping), since we do not treat this situation as a special case in our framework.

Horses. This scene contains 500 horses, for a total of over 8 million polygons. It demonstrates that our improvement

in comparison to brute-force rendering improves with more complex scenes, since the overhead of our technique (clustering and mappings) is independent of polygon count.

Discussion and Limitations. A valid question is why does this approach perform better on animations than [HPB07]. First, some coherence of the clustering across frames is achieved because of its "rectangular" nature. Second, since clusters can span several frames, the *effective* number of clusters that contribute to a frame is higher than the average number of representatives rendered per frame. For example, we use 200 average representatives per frame (400 in the still life scene); however, the effective number of clusters is about 230-400 in the iris scene, 1500-7000 in the still scene, and 7000-7300 in the temple. One can note that in scenes with less temporal change of geometry, the algorithm creates "longer" clusters in time, which translate to a larger number of effective clusters per frame.

Some of our animations still show some flicker; however, increasing the number of clusters should decrease this problem. A higher number of clusters slows down only the reconstruction phase and (very slightly) the clustering phase; performance remains identical for other phases.

The well-known limitations of shadow mapping and instant-radiosity type approaches are also present in our algorithm – shadow bias is required to reduce self-shadowing, indirect light clamping is needed to avoid the explosion of the $1/r^2$ term for indirect lights near surfaces, and not all light paths can be handled (e.g., caustics). We also currently have no automatic way of selecting parameters like the number of rows and columns to sample.

7. Conclusion

We have introduced a tensor formulation for rendering animations with complex geometry and lighting (including indirect illumination and environment maps), deformable objects and arbitrary materials and shaders. From this formulation, we have derived an efficient algorithm that takes advantage of temporal coherence to compute high-quality images with low temporal flicker in several seconds on the GPU.

Our contributions, other than the tensor formulation itself, are two-fold: the ability to amortize shading across animations, and a solution to the problem of temporal flicker. Notably, we only compute a few hundred (300-500) shadow maps per frame to achieve these goals, whereas the brute-force method would compute shadow maps for all lights (tens of thousands per frame).

We believe this approach can be very useful in applications like previewing in cinematic rendering, where an entire shot can be viewed with very high quality. In future work, we would like to explore combination of our sampling framework with motion blur, depth of field and volumetric effects in arbitrary animated scenes.

Acknowledgments

The first two authors and last author were supported by NSF CAREER 0644175 and 0403340, and grants from Intel Corporation, NVidia Corporation, and Microsoft Corporation. The second author was also supported by CONACYT-Mexico 228785. The third author was supported by NSF CNS-070820 and CCF-0746117. We thank Chris Twigg and Doug James for the iris dataset, Veronica Sundstedt and Patrick Ledda for the temple model, and Bruce Walter for code and support.

References

[Bun05] BUNNELL M.: Dynamic ambient occlusion and indirect lighting. *GPU Gems 2: Programming Techniques for High-Performance Graphics and General-Purpose Computation* (2005).

[BWG03] BALA K., WALTER B. J., GREENBERG D. P.: Combining edges and points for interactive high-quality rendering. *Proceedings of ACM SIGGRAPH* (2003), 631–640.

[DBMS02] DMITRIEV K., BRABEC S., MYSZKOWSKI K., SEIDEL H.-P.: Interactive global illumination using selective photon tracing. *Proceedings of the 13th Eurographics workshop on Rendering* (2002), 25–36.

[dlVKKR02] DE LA VEGA W. F., KARPINSKI M., KENYON C., RABANI Y.: Polynomial time approximation schemes for metric min-sum clustering. *Electronic Colloquium on Computational Complexity (ECCC)*, 025 (2002).

[DS97] DRETTAKIS G., SILLION F. X.: Interactive update of global illumination using a line-space hierarchy. *Proceedings of ACM SIGGRAPH* (1997), 57–64.

[DS05] DACHSBACHER C., STAMMINGER M.: Reflective shadow maps. *Proceedings of the Symposium on interactive 3D graphics and games* (2005), 203–231.

[DS06] DACHSBACHER C., STAMMINGER M.: Splatting of indirect illumination. *ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games* (2006).

[DSDD07] DACHSBACHER C., STAMMINGER M., DRETTAKIS G., DURAND F.: Implicit visibility and antiradiance for interactive global illumination. *Proceedings of ACM SIGGRAPH* (2007).

[GBP07] GAUTRON P., BOUATOUCH K., PATTANAIK S.: Temporal radiance caching. *IEEE Transactions on Visualization and Computer Graphics 13*, 5 (2007), 891–901.

[HDMS03] HAVRAN V., DAMEZ C., MYSZKOWSKI K., SEIDEL H.-P.: An efficient spatio-temporal architecture for animation rendering. *Preceedings of the 14th Eurographics Workshop on Rendering* (2003), 106–117.

[HPB06] HAŠAN M., PELLACINI F., BALA K.: Direct-to-indirect transfer for cinematic relighting. *Proceedings of ACM SIGGRAPH* (2006), 1089–1097.

[HPB07] HAŠAN M., PELLACINI F., BALA K.: Matrix row-column sampling for the many-light problem. *Proceedings of ACM SIGGRAPH* (2007).

[Jen01] JENSEN H. W.: *Realistic image synthesis using photon mapping*. A. K. Peters, Ltd., Natick, MA, USA, 2001.

[Kel97] KELLER A.: Instant radiosity. *Proceedings of ACM SIGGRAPH* (1997), 49–56.

[LSK*07] LAINE S., SARANSAARI H., KONTKANEN J., LEHTINEN J., AILA T.: Incremental instant radiosity for real-time indirect illumination. *Proceedings of Eurographics Symposium on Rendering* (2007).

[MTAS01] MYSZKOWSKI K., TAWARA T., AKAMINE H., SEIDEL H.-P.: Perception-guided global illumination solution for animation rendering. In *Proceedings of ACM SIGGRAPH* (2001), pp. 221–230.

[NRH04] NG R., RAMAMOORTHI R., HANRAHAN P.: Triple product wavelet integrals for all-frequency relighting. *Proceedings of ACM SIGGRAPH* (2004).

[ODJ04] OSTROMOUKHOV V., DONOHUE C., JODOIN P.-M.: Fast hierarchical importance sampling with blue noise properties. *Proceedings of ACM SIGGRAPH* (2004), 488–495.

[RWS*06] REN Z., WANG R., SNYDER J., ZHOU K., LIU X., SUN B., SLOAN P.-P., BAO H., PENG Q., GUO B.: Real-time soft shadows in dynamic scenes using spherical harmonic exponentiation. *Proceedings of ACM SIGGRAPH* (2006), 977–986.

[SKDM05] SMYK M., KINUWAKI S.-I., DURIKOVIC R., MYSZKOWSKI K.: Temporally coherent irradiance caching for high quality animation rendering. *Proceedings of EUROGRAPHICS* (2005).

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

[TPWG02] TOLE P., PELLACINI F., WALTER B., GREENBERG D. P.: Interactive global illumination in dynamic scenes. *Proceedings of ACM SIGGRAPH* (2002), 537–546.

[WABG06] WALTER B., ARBREE A., BALA K., GREENBERG D. P.: Multidimensional lightcuts. *Proceedings of ACM SIGGRAPH* (2006), 1081–1088.

[WDP99] WALTER B., DRETTAKIS G., PARKER S.: Interactive rendering using the render cache. *Eurographics Rendering Workshop* (1999).

[WFA*05] WALTER B., FERNANDEZ S., ARBREE A., BALA K., DONIKIAN M., GREENBERG D. P.: Lightcuts: a scalable approach to illumination. *Proceedings of ACM SIGGRAPH* (2005), 1098–1107.