LightSlice: Matrix Slice Sampling for the Many-Lights Problem

Jiawei Ou*  
Dartmouth College

Fabio Pellacini†  
Dartmouth College

Abstract

Recent work has shown that complex lighting effects can be well approximated by gathering the contribution of hundreds of thousands of virtual point lights (VPLs). This final gathering step is known as the many-lights problem. Due to the large number of VPLs, computing all the VPLs’ contribution is not feasible. This paper presents LightSlice, an algorithm that efficiently solves the many-lights problem for large environments with complex lighting. As in prior work, we derive our algorithm from a matrix formulation of the many-lights problem, where the contribution of each VPL corresponds to a column, and computing the final image amounts to computing the sum of all matrix columns. We make the observation that if we cluster similar surface samples together, the slice of the matrix corresponding to these surface samples has significantly lower rank than the original matrix. We exploit this observation by deriving a two-step algorithm where we first globally cluster all lights, to capture the global structure of the matrix, and then locally refine these clusters to determine the most important lights for each slice. We then reconstruct a final image from only these locally-important lights. Compared to prior work, our algorithm has the advantage of discovering and exploiting the global as well as local matrix structure, giving us a speedup of between three and six times compared to state-of-the-art algorithms.

CR Categories: I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Color, shading, shadowing, and texture

Keywords: many light, global illumination, matrix sampling

Links: ✈DL  ●PDF

1 Introduction

Realistic Rendering as Many-Lights. Fast computation of global illumination in large scenes with a complex lighting configuration is still a challenging problem in computer graphics. Many methods have been proposed to compute fast global illumination solutions, e.g. bidirectional path tracing and photon mapping. (see [Pharr and Humphreys 2010] for a recent review). In this paper, we focus on computing images using a variant of instant global illumination [Keller 1997], where direct and indirect illumination are approximated by converting the original light sources into a large number of virtual point lights (VPLs) distributed across the entire scene. In this model, computing a global illumination solution is equivalent to computing an image lit solely by a large number of point light sources, i.e. the many-lights problem. Prior work in offline, high-fidelity rendering [HaSan et al. 2007; Walter et al. 2005] has shown that for scenes with diffuse and low gloss materials, hundreds or thousands of VPLs effectively approximate complex direct and indirect illumination effects, while having the advantage of treating both equally within the same algorithm framework. VPLs have also been used in real-time applications, where they handle a smaller number of lights at the price of the accuracy of approximation [Ritschel et al. 2009]. VPLs have also found much use in feature film production [Christensen 2008]. In this paper, we focus on high-fidelity rendering in complex environments rather than interactive applications.

Matrix Interpretation of Many-Lights. It is useful to consider an alternative interpretation of the many-lights problem as a matrix sampling problem. Let us arrange all pixels of an image as a long column vector. We can then arrange all columns corresponding to each VPL as a large unknown matrix. Computing the final image amounts to computing the sum of each row in the matrix. Figure 1 shows an example of such a matrix. While many-lights algorithms
handle all lights equally, it is useful to note that the lights have two common behaviors. Ignoring shadows, some lights have strong contributions to all pixels; these VPLs typically correspond to direct illumination, e.g., the sun. We term these global lights. When unshadowed, these appear as bright matrix columns. When shadowed, these appear as columns with bright and black sections or an entirely black column. Other lights have more local behavior affecting only a few pixels; typically, these are VPLs derived by sampling indirect lighting, and thus have lower intensity and an r-squared falloff. We term these local lights. In the matrix, these lights appear mostly as black columns with a small, low intensity section.

For hundreds of thousands of lights, a brute force solution that computes all columns of the many-lights problem is prohibitively expensive. Many methods have been proposed to reduce the computation complexity of the many-lights problem to be sublinear in the number of VPLs. The two main observations that allow this is that the elements of the matrix have repeating patterns and that large areas of low contributions are present (see Fig. 1). All scalable many-lights algorithms exploit these two observations by clustering groups of similar VPLs and approximating their contribution using a single representative. In other words, all these methods subsample the matrix by approximating blocks of similar elements as constant values computed from only one element. The size and shape of these blocks changes for different algorithms. We compare our work with two main prior algorithms.

**Exploiting the Matrix Structure.** Lightcuts [Walter et al. 2005] hierarchically clusters the lights into a light tree using geometric proximity as the cluster metric. It then renders the final image by choosing a set of representative clusters differently for each pixel. Matrix Row-Column Sampling (mrcs in short) [Hašan et al. 2007] clusters entire matrix columns together and renders one representative column for the entire clusters. This is motivated by the observation that the transport matrix is close to low rank. As we will discuss later in details, for large environments and complex lighting, neither lightcuts nor mrcs optimally exploits the structure found in these matrices as shown in Fig. 1. The former works well for local lighting and mostly-visible global lights, but oversamples shadowed global lights (corresponding to bright columns with large black segments or entirely black). The latter works well for global lights, quickly determining the global visibility behavior, but is inefficient for local lighting (corresponding to low intensity columns that are mostly black) in that it samples them for all pixels. We are interested in deriving a matrix sampling algorithm that has the benefits of both approaches.

The main observation of our work is that, if we cluster similar pixels together, the slice of the matrix corresponding to these pixels has significantly lower rank than the original matrix. Intuitively this is true since for each slice, local lighting and shadowed regions can all be approximated together with a low intensity representative. This observation was already exploited in the domain of precomputed radiance transfer to either compute the transport matrix [Huang and Ramamoorthi 2010; Mahajan et al. 2007] or for compression [Slano et al. 2003]. However each of these approaches are significantly different from ours since they aim to approximate the whole matrix, while we only need to compute the sum of matrix columns.

**LightSlice.** In this paper we present LightSlice, an algorithm that efficiently solves the many-lights problems by sampling matrix slices. In LightSlice, we first determine matrix slices by clustering similar image pixels based on their geometric proximity. For each of these slices we render a representative and roughly cluster all columns based on all row values. This initial clustering effectively captures the global structure of the matrix. For each slice, we then refine such global clusters into per-slice clusters based on representative rows of the given slice and its neighboring slices. This effectively captures the local structure of the matrix, including its shadowing behavior. We render each slice by choosing representative columns and only rendering the column elements corresponding to the slice rows. LightSlice combines the advantages of both lightcuts and mrcs by effectively capturing the global structure of the matrix, including its shadowing, while adapting to the local changes for each slice.

We tested our algorithm on a variety of complex scenes with indoor and outdoor illumination. LightSlice is consistently faster than other algorithms, with between three and six times speedup. More importantly, each of these prior algorithms works well for some scenes, but becomes inefficient for others. This is due to the fact that each of them is optimal for some matrix structure but inefficient for others. LightSlice is instead consistently efficient in all our scenes since it can adapt to the typical matrix structures found in complex lighting scenarios.

### 2 Related Work

Realistic rendering is one of the most studied problems in Computer Graphics. A wide variety of algorithms have been presented to efficiently render scenes lit by complex direct and indirect illumination. Rather than attempting to review all these methods, we refer the reader to [Pharr and Humphreys 2010] for a review. In this section we quickly review recent work that uses VPL as the main rendering primitive.

**Lightcuts.** Lightcuts [Walter et al. 2005] hierarchically clusters the lights into a light tree using geometric proximity as the cluster metric. Each tree node corresponds to a light cluster where a single representative VPL is selected to approximate the cluster contribution. lightcuts renders the final image by choosing a set of representative clusters differently for each pixel and computing their contribution by raytracing. The choice of representative is done by hierarchically exploring the light tree, where at each node the algorithm descends into its children if the current representative is deemed to be a poor approximation of the cluster for the pixel being rendered. The estimation is made by evaluating a conservative error metric based on properties of light, pixel geometry and materials, but ignoring visibility. In a matrix interpretation, lightcuts can be thought of as sampling each row of the matrix independently, but with a precomputed ordering in light tree construction. Typically, the columns corresponding to the highest level clusters are rendered mostly in full, while sparse elements are selected independently for each row thereafter to efficiently adapt to local lighting, for which lightcuts works very well. For global lighting, however, lightcuts is not as efficient because it ignores shadowing when estimating light contributions. This means that if large shadows are present, most pixels will sample that amount of matrix elements with zero contribution. Multidimensional Lightcuts [Walter et al. 2006] extend lightcuts by introducing a gather tree to reduce the cost of antialiasing. However, because the error metric of Multidimensional Lightcuts is designed for the use of evaluating one single integral, it cannot be easily extended to evaluate multiple integrals (e.g. bound the error of a group of gather points from different pixels) simultaneously. Thus the overhead of walking the light tree cannot be amortized across pixels.

**Matrix Row-Column Sampling.** In Matrix Row-Column Sampling, [Hašan et al. 2007] makes the observation that the transport matrix is close to low rank, since the image corresponding to nearby lights is similar, so are their columns. To compute a final image, mrcs computes light clusters and renders one representative per cluster. The same representative is used for all pixels. To determine the optimal clustering, mrcs samples a small set of matrix elements and approximates them using a single representative. In other words, all these methods subsample the matrix by approximating blocks of similar elements as constant values computed from only one element. The size and shape of these blocks changes for different algorithms. We compare our work with two main prior algorithms.

**Exploiting the Matrix Structure.** Lightcuts [Walter et al. 2005] hierarchically clusters the lights into a light tree using geometric proximity as the cluster metric. It then renders the final image by choosing a set of representative clusters differently for each pixel. Matrix Row-Column Sampling (mrcs in short) [Hašan et al. 2007] clusters entire matrix columns together and renders one representative column for the entire clusters. This is motivated by the observation that the transport matrix is close to low rank. As we will discuss later in details, for large environments and complex lighting, neither lightcuts nor mrcs optimally exploits the structure found in these matrices as shown in Fig. 1. The former works well for local lighting and mostly-visible global lights, but oversamples shadowed global lights (corresponding to bright columns with large black segments or entirely black). The latter works well for global lights, quickly determining the global visibility behavior, but is inefficient for local lighting (corresponding to low intensity columns that are mostly black) in that it samples them for all pixels. We are interested in deriving a matrix sampling algorithm that has the benefits of both approaches.

The main observation of our work is that, if we cluster similar pixels together, the slice of the matrix corresponding to these pixels has significantly lower rank than the original matrix. Intuitively this is true since for each slice, local lighting and shadowed regions can all be approximated together with a low intensity representative. This observation was already exploited in the domain of precomputed radiance transfer to either compute the transport matrix [Huang and Ramamoorthi 2010; Mahajan et al. 2007] or for compression [Slano et al. 2003]. However each of these approaches are significantly different from ours since they aim to approximate the whole matrix, while we only need to compute the sum of matrix columns.

**LightSlice.** In this paper we present LightSlice, an algorithm that efficiently solves the many-lights problems by sampling matrix slices. In LightSlice, we first determine matrix slices by clustering similar image pixels based on their geometric proximity. For each of these slices we render a representative and roughly cluster all columns based on all row values. This initial clustering effectively captures the global structure of the matrix. For each slice, we then refine such global clusters into per-slice clusters based on representative rows of the given slice and its neighboring slices. This effectively captures the local structure of the matrix, including its shadowing behavior. We render each slice by choosing representative columns and only rendering the column elements corresponding to the slice rows. LightSlice combines the advantages of both lightcuts and mrcs by effectively capturing the global structure of the matrix, including its shadowing, while adapting to the local changes for each slice.

We tested our algorithm on a variety of complex scenes with indoor and outdoor illumination. LightSlice is consistently faster than other algorithms, with between three and six times speedup. More importantly, each of these prior algorithms works well for some scenes, but becomes inefficient for others. This is due to the fact that each of them is optimal for some matrix structure but inefficient for others. LightSlice is instead consistently efficient in all our scenes since it can adapt to the typical matrix structures found in complex lighting scenarios.

**Matrix Row-Column Sampling.** In Matrix Row-Column Sampling, [Hašan et al. 2007] makes the observation that the transport matrix is close to low rank, since the image corresponding to nearby lights is similar, so are their columns. To compute a final image, mrcs computes light clusters and renders one representative per cluster. The same representative is used for all pixels. To determine the optimal clustering, mrcs samples a small set of matrix elements and approximates them using a single representative. In other words, all these methods subsample the matrix by approximating blocks of similar elements as constant values computed from only one element. The size and shape of these blocks changes for different algorithms. We compare our work with two main prior algorithms.
rows in full to form a reduced matrix. The columns of this matrix are then clustered in such a way that similar columns that have low contribution are grouped together. \textit{mrcs} works very well for global lighting since it uses subsampled rows (including shadowing) to choose the matrix columns that have the highest contribution to the final image. The cost of carefully choosing columns is amortized over all pixels in the image, making the algorithm effective. At the same time, since \textit{mrcs} reconstructs the final image by rendering entire columns, it is not efficient for local lights whose corresponding columns have most matrix elements with zero contribution. Note that while the original implementation of \textit{mrcs} computed rows and columns on the GPU using shadow mapping, we found that in large scenes shadow maps exhibit too many biasing and sampling artifacts to produce high quality images. In this paper, we will compare our algorithm to an implementation of \textit{mrcs} executed on a raytracer that does not suffer from these artifacts. [Hašan et al. 2008] extends this methods to animation by using reprojection and tensor clustering.

**Interactive Many-Lights Rendering.** [Keller 1997] introduced instant radiosity, a method that uses a relatively small number of VPLs to simulate the global illumination in a scene. The main advantage of VPLs is that their contribution can be gathered efficiently by using shadow mapping on the GPU. Today, VPL rendering is still the basic building block of many interactive global illumination algorithms. Imperfect shadow maps [Ritschel et al. 2008] uses low quality shadow maps to approximate visibility when gathering VPL contributions. [Ritschel et al. 2009] uses small rasterization buffers warped by BRDF importance to accurately account for the glossy reflection in the final bounce. VPL approaches are also used in conjunction with precomputed radiance transport for interactive relighting. [Hašan et al. 2006] presents an interactive GPU-based system for cinematic relighting with multiple-bounce indirect illumination from a fixed view-point. [Cheslack-Postava et al. 2008] uses precomputed visibility cuts to perform interactive lighting and material design. Our method is in different scope from all these approaches since it is designed to handle significantly larger number of VPLs without precomputation.

**Improving VPL Rendering Quality.** A recent extension to VPL rendering methods uses virtual sphere lights [Hašan et al. 2009] to reduce the loss of energy due to clamping. [Křivánek et al. 2010] showed that VPL methods have trouble in accurately representing the look of highly-glossy materials. To handle these cases, [Davidić et al. 2010] combined row-column sampling with selective raycasting to handle glossy BRDFs. We refer to our introduction for a detail discussion of lightcuts and \textit{mrcs} as they compare to our method.

**Precomputed Radiance Transfer.** Precompute radiance transfer algorithms also utilize the fact that the light transfer matrix is locally low rank. The CPCA method reduces the high-dimensional transfer signal to low-dimensional by partitioning many samples into fewer clusters [Sloan et al. 2003]. [Mahajan et al. 2007] gives a theoretical analysis of the local low-rankness of the light transport matrix. [Huang and Ramamoorthi 2010] utilizes the locally low-rankness to sparsely sample the spatial domain. Although this analysis inspired our work, it is not directly applicable to our problem since PRT methods sample and reconstruct the entire matrix, while we only seek to compute the sum of matrix columns.

### 3 Algorithm

**Overview.** We formulate the many-lights problem as a matrix sampling problem, where matrix $A$ is the transport matrix of size $m \times n$, where $m$ is the number of surface samples and $n$ is the number of virtual point lights collected in the scene. Each element $A(i, j)$ of matrix $A$ is the contribution of virtual light $j$ to surface sample $i$, and the final rendering of surface sample $i$ is the sum of the contributions from all VPLs: $I(i) = \sum_j A(i, j)$. In our implementation, we generate VPLs for direct illumination by randomly sampling area lights and environment maps using stratified sampling, and VPLs for indirect illumination by particle tracing as in [Keller 1997]. For our test scenes, we use 150-300K VPLs taking roughly 3 to 5 seconds to generate.

With this high number of VPLs, the cost of computing matrix $A$ with a brute force algorithm is prohibitively high. \textit{LightSlice} is an algorithm that provides efficient and accurate approximation of the many-lights problem by exploring and exploiting the structure of the light transport matrix. Figure 2 shows the steps of our algorithm:

1. **Matrix Slicing:** We first partition the surface samples based on their geometric proximity. This is equivalent to slice the rows of the transport matrix in such a way that each slice contains the contribution of closeby surface samples, thus the lighting effects are likely similar within the slice. For each slice, we seek to optimally choose the important lights for each slice.

2. **Slice Sampling:** Next, we pick a representative point per slice and compute the corresponding row of the matrix $A$. These sampled rows form a reduced transport matrix $R$, which is a submatrix of $A$. As shown in [Hašan et al. 2007], $R$ contains enough information to capture the global structure of $A$ while being significantly smaller in size. Intuitively, one can interpret $R$ as a stratified sampling of $A$.

3. **Initial Light Clustering:** Since the structure of global lighting effects is captured well by $R$, we perform a rough initial clustering of the columns of $R$, i.e., the scene’s lights, just as in \textit{mrcs}. This provides a good initial estimate of light clusters, which can be further refined on a per-slice basis to better capture local effects. Overall, this initial clustering significantly reduces the cost of determining per-slice light clusters.

4. **Per-slice Cluster Refinement:** Starting from the initial clustering, our algorithm further refines the light clusters for each slice independently, by splitting global clusters into local ones in order to efficiently capture local lighting effects. In matrix form, this can be interpreted as splitting global clusters of high rank, while leaving the low rank ones unsplit. Since the rank of local lighting effects varies strongly per slice, this generates different light clusters for slice, each of which captures

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
<th>Dimensionality</th>
</tr>
</thead>
<tbody>
<tr>
<td>$m$</td>
<td>Number of pixels</td>
<td>scalar</td>
</tr>
<tr>
<td>$n$</td>
<td>Number of VPLs</td>
<td>scalar</td>
</tr>
<tr>
<td>$r$</td>
<td>Number of slices</td>
<td>scalar</td>
</tr>
<tr>
<td>$A$</td>
<td>Full lighting matrix</td>
<td>$m \times n$</td>
</tr>
<tr>
<td>$R$</td>
<td>Global reduced matrix</td>
<td>$r \times n$</td>
</tr>
<tr>
<td>$A_l$</td>
<td>Column of $A$</td>
<td>$m \times 1$</td>
</tr>
<tr>
<td>$R_l$</td>
<td>Column of $R$</td>
<td>$r \times 1$</td>
</tr>
<tr>
<td>$l_s$</td>
<td>Number of pixels in slice $S_i$</td>
<td>scalar</td>
</tr>
<tr>
<td>$S_i$</td>
<td>A slice of matrix $A$</td>
<td>$l_s \times n$</td>
</tr>
<tr>
<td>$S_i^l$</td>
<td>Column of $S_i$</td>
<td>$l_s \times 1$</td>
</tr>
<tr>
<td>$k$</td>
<td>Number of k-nearest-neighbour</td>
<td>scalar</td>
</tr>
<tr>
<td>$L_i^l$</td>
<td>Local reduced matrix for slice $S_i^l$</td>
<td>$k \times n$</td>
</tr>
<tr>
<td>$C_i^l$</td>
<td>Column (light) clusters of $A$ estimated from $R$</td>
<td>set of VPLs</td>
</tr>
<tr>
<td>$C_i^l$</td>
<td>Column (light) clusters of $S_i^l$ derived from $L_i^l$</td>
<td>set of VPLs</td>
</tr>
<tr>
<td>$c$</td>
<td>Number of $A$'s clusters</td>
<td>scalar</td>
</tr>
<tr>
<td>$c^l$</td>
<td>Number of $S_i$'s clusters</td>
<td>scalar</td>
</tr>
</tbody>
</table>

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

5. Per-Slice Reconstruction: For each slice, we reconstruct the final image by rendering the slice elements of one representative column for each light cluster.

Matrix Slicing. We cluster surface samples using a top-down kd-partition scheme as proposed in [Walter et al. 2006]. First, we map the surface samples onto a 6D space, where the 6D coordinates encompass both position and normal at each point. We then iteratively split the 6D bounding box along the longest axis until either the bounding box or the number of samples is small enough. We will also stop refining adaptively by checking if less than 10000 gather points are contained in a leaf. Each cluster $S^i$ corresponds to a slice of matrix $A$:

$$A = \begin{bmatrix} S^1 \\ S^2 \\ \vdots \\ S^n \end{bmatrix}$$

Slice Sampling. We sample each slice by randomly picking one representative point and compute the full corresponding row using raytracing. We assemble the sampled rows for all slices in a reduced matrix $R$ that represents a stratified sampling of the full matrix $A$.

Initial Light Clustering. As pointed out by previous work, since $A$ is close to low rank, the structure of $A$ that is shared by all slices, typically originating from global lights, is captured well by $R$. For example, bright lights visible within the entire scene (e.g., bright, long columns in Fig. 1) correspond to columns of $R$ with dense, large numbers. Therefore, we perform a rough clustering on the columns of $R$ as initial light clusters for all slices. This greatly reduces computation time needed to compute light clusters for each slice. In our implementation, we adapted the cluster metric proposed by [Hašan et al. 2007] to compute clustering. More specifically, we cluster columns of $R$ by minimizing the summed cost of its $c$ clusters $C^R_k$:

$$\sum_{k=1}^{c} cost(C^R_k) = \sum_{k=1}^{c} \sum_{p,q \in C^R_k} d_R(p,q)$$

$$d_R(p,q) = ||R_p|| \cdot ||R_q|| \cdot ||R_p - R_q||^2$$

where $d_R(p,q)$ is the distance between two column $p$ and $q$ in the reduced matrix $R$. $||x||$ denotes the norm of vector $x$ and $\bar{x}$ is the normalized vector $\bar{x} = x/||x||$. Intuitively, the objective of this cost function is to partition the columns into clusters such that the similar, low-intensity columns are grouped together. We implement the clustering using the multi-set sampling algorithm described in [Hašan et al. 2007]. In our implementation, the number of initial clusters is set to roughly 30% of the total cluster budget.

Per-Slice Cluster Refinement. The initial light clustering roughly captures the global structure of the matrix, but cannot adapt efficiently to the local matrix structure. We further refine this clustering for each slice to compute per-slice light clusters that adapt well to local lighting. For each slice, we assemble a local matrix $L'$, a submatrix of $R$, by combining the representative row for the slice $S'$ with the representative rows for spatially close slices that are likely to exhibit similar local structure. We find these spatial neighbors by performing a nearest neighbor search in the 6D-space kd-tree used for matrix slicing. We do this to ensure that spatially close slices have similar local clustering, thus avoiding image-space discontinuity artifacts commonly found when rendering image blocks separately. We then cluster the columns of the local matrix $L'$ to determine the light clusters for the slice $S'$.

We determine light clusters for each slice by iteratively splitting the highest cost cluster until a maximum number of clusters, corresponding to our reconstruction budget, is reached. We initialize the procedure by assigning each slice column to the same clusters found in the global clustering. In other words, for each slice $S'$, we initialize the clusters as $C^L_k = C^R_k$ for the rows in $L'$. At each iteration, we evaluate the clustering cost function in equation (2) with the elements of $L'$, i.e., $\sum_{k=1}^{c} \sum_{p,q \in C^L_k} d_L(p,q)$. We then split the cluster with the largest cost. We can do so efficiently by maintaining a priority queue. To split the cluster, we first randomly pick two columns with probability proportional to their norm $||L_p||$ and compute the corresponding line in r-dimensional space. We then project all other columns onto this line and find the best position to cut the line into two. After this procedure, we have clustered the columns of $L'$ in $c_l$ as clusters $C^L_k$.

Per-Slice Reconstruction. Similarly to [Hašan et al. 2007], we render each slice $S'$ by summing the contribution of each of its clusters $C^L_k$. We estimate the contribution of each cluster $C^L_k$ by rendering the elements of one representative column $j_k$ belonging to the slice $C^L_k$ using raytracing. We choose the representative column randomly with probability proportional to its global norm.
\[ ||R_j|| \] and estimate its total weight as \((\sum_{j \in C_k} ||R_j||)/||R||\). We consider \( ||R_j|| \) as a measure of the contribution a virtual point light to the scene and it is proportional to \( ||L_j|| \). We use \( ||R|| \), instead of \( ||L_j|| \), to compute the weighting, because it contains more row samples and can provide a more numerically-stable estimation.

**Anti-aliasing and Multiple Representatives.** In the case of anti-aliasing, we render \( s \) gather points for each pixel. To obtain better anti-aliasing of lighting effects (e.g., such as soft shadows), we modified our per-slice reconstruction. For each cluster, instead of selecting one representative, we randomly pick a set of \( s \) representatives as described above. Each gather point in the pixel is paired with a different representative in the set and this set of representatives is shared by all pixels in the slice. By having each gather point in a pixel connected to a different representative, we enable a better shadow ray distribution, while still maintaining per-pixel noise to a minimum as in \textit{mrcs}.

### 4 Results and Discussion

**Overview.** Table 2 shows statistics for the four scenes we tested, with geometric complexity ranging from 0.6 to 1.6 million triangles. We include timings for \textit{LightSlice}, \textit{lightcuts}, and \textit{mrcs}, where we show equal-time and equal-quality timings for the latter two methods. Our implementations of all algorithms are available as source code in supplemental material. For each scene we tested, Fig. 3 shows the equal-time renderings obtained with the three methods as well as a reference solution computed by rendering all VPLs. High resolution images and color-coded difference images are available in supplemental. We generate all \textit{LightSlice} results using the same parameters for the gather kd-tree construction and the number of columns. Note that since we use an adaptive scheme to select the slices during the kd-tree build, the number of slices is different for each scenes. The right most column of Fig. 3 shows a visualization of the gather point clusters for each scene, where each patch in the image corresponds to a slice of the transport matrix. We report timing results for parallel implementations of the various algorithms running on a machine with four Intel Xeon 7560 processors, each with 8 cores, running at 2.27GHz. \textit{LightSlice} converges within minutes. We report the average per-pixel relative error and the \( L^2 \) image error for all renderings.

For the \textit{lightcuts} algorithm, we chose to compare with the multi-dimensional \textit{lightcuts} variant [Walter et al. 2006], because this is more efficient in the presence of antialiasing. We augment the conservative error bound of \textit{lightcuts}, which we set to 0.01, with a maximum cut size to induce stopping not satisfies any scene. This bound reduces the number of possible subdivisions. For our \textit{mrcs} implementation, we chose to use raycasting rather than shadow mapping since our environments are large and contain fine geometric details. In these situations, shadow maps may not be sufficiently sampled as well as aliasing, as noted in the original paper.

**Scenes.** The \textit{Sanmiguel} model is an outdoor scene with complex geometry lit by an environment map where most of the illumination come from the sun. This is our most challenging scene. For most parts of the image, global lighting is dominant, but with complex shadows. In the archway region, local lighting is dominant. \textit{mrcs} quickly finds the dominant global lights but has trouble converging.

### Table 2: Rendering statistics comparing \textit{LightSlice}, multidimensional \textit{lightcuts} and \textit{mrcs} row column sampling. Note how our algorithm consistently gives us a significant speed up in equal quality comparisons or a smaller error in equal time comparisons.
Figure 3: Equal-time comparison of VPL rendering methods. From the top: Sanmiguel, Museum, Condo and Lobby. Note that while LightSlice reproduces the complex lighting effects in the reference image, both other algorithms show various artifacts. The last column contains a visualization of the gather point clusters corresponding to the matrix slices.

on the archway region dominated by local effects. Lightcuts works well on those regions, but invest significant resources in rendering environment map lights, some of which are mostly occluded, since it lacks a visibility bound. Our algorithm combines the advantages of both prior methods since it can quickly exclude occluded global VPLs while converging quickly on local VPLs. These same trends are visible on all scenes tested.

The Museum model is an indoor scene with relatively simple geometry lit by an outdoor sun shining through small windows. The archway and ceiling area are lit only by indirect illumination. This scene is challenging for all VPL methods since most of the image is lit by strong local lighting. Our algorithm performs better than others, which exhibit more banding in equal time comparisons.

The Condo model is a relatively simple environment mostly lit by direct illumination coming from the ceiling. We chose this scene because it highlights the effect of multiple bounces of indirect lighting, which is particularly visible in the region below the stairs. Even in this lighting scenario, our algorithm performs very well, while the other two are less efficient in converging in the area mentioned.

Finally, the Lobby model is an indoor scene with an open ceiling covered by a thin metal frame. The illumination is dominated by global lighting from an environment map with regions of local lighting in the bottom floor. We chose this scene because it exhibits sharp and soft shadow from thin as well as large geometry, covering the range of direct shadowing characteristics. We found that even in this case, our algorithm works very well compare to the other two.

Performance in Equal Quality Comparison. To compare the three algorithms in terms of performance, we perform an equal-quality comparison by finding the best settings for mrcs and lightcuts with an exhaustive search. We chose to match the average relative error that better captures perceptual issues and is related to the lightcuts stopping criteria. With these settings, we have a speed up of between roughly three to six times when compared to the other methods. LightSlice allocates most of its resources to cluster refinement and final gathering, which are executed together in our parallel implementation. This, combined with the speed up we obtain, demonstrated that the exploration phase of our algorithm is well worth executing since it allows us to better direct resources according to the matrix structure.

In the scenes we tested, we found that mrcs works best when we dedicate most of its resources to column sampling, as summarized in Tab. 2. This results in a significantly larger number of gather rays per pixel than our method for final image reconstruction. This slow
Figure 4: Average relative error vs. time plot for each algorithm rendering the Sanmiguel scene. The images are rendered using the same number of rows and slices as reported in Table 2 while varying the number of columns or maximum cut size. The result shows that LightSlice is able to reduce error quicker than the other two VPL methods.

Figure 5: Per-pixel relative error for the equal-time comparisons of Fig. 3. From the top: Sanmiguel, Museum, Condo and Lobby. The images show that given the same amount of time, LightSlice has lower error than other methods.

5 Limitations and Future Work

Parameters Selection. LightSlice implicitly avoids spatial discontinuities by using a high number of slices and including neighboring slices in the cluster refinement. Compared to mrcs which inherently has no discontinuities, LightSlice is more conservative when refining clusters and splits more aggressively. When lowering the number of slices, the coarse matrix slicing causes two types of artifacts, shown in Fig. 6 (left). For some regions (e.g., red box), the algorithm does not capture the local matrix structure and thus does not allocate sufficient resources to locally-important VPLs. In regions with strong dissimilarity between neighboring slices (e.g., yellow box), the discontinuities between gather point clusters become visible. Increasing the number of columns lowers the error thus alleviates these artifacts considerably, but also significantly increase the rendering time, as shown in Fig. 6 (right).

Glossy Surfaces. All VPL methods have difficulty in handling highly glossy materials, since the number of VPLs needed grows significantly for these algorithms to remain efficient. Moreover, glossy transports increase the local matrix rank, causing additional inefficiencies. In the case of LightSlice, a higher number of columns is needed to avoid banding artifacts coming from the dissimilarity between neighboring slices. A promising approach to support glossy surfaces is to combine selective raycasting with LightSlice as shown in [Davidović et al. 2010].

Animation. In general, we expect a larger number of VPLs to be
necessary for rendering flicker-free animations, lengthening rendering times. [Ha\v{s}an et al. 2008] proposed a tensor rendering approach that relies on reprojection to amortize final gathering cost. Applying this approach directly to our scenes would not work though because it would cause severe artifacts on fine geometry, preventing us from amortizing gathering cost. We leave it to future work to investigate an alternative approach to speed animations.

Matrix Sparsity. On the theoretical side, if the transport matrix is very sparse, LightSlice can not efficiently converge on the correct solution, because it will have trouble finding the sparse elements. mrcs will have a similar behavior but with slower render times. Lightcuts will eventually converge on these cases, but at the price of rendering most of the matrix in an attempt to find the sparse elements. Besides the case of highly glossy surfaces, we have not been able to find such a case in rendering various scenes, including the ones that have difficult shadowing, such shadows from tree leaves, small windows, or fine geometry. [Ha\v{s}an et al. 2007] drew similar conclusions. In other words, while we cannot theoretically prove that all transport matrices have low rank structure, or at least have locally low rank structure, we have yet to find a case for which this is not the case. This makes this limitation worthwhile to highlight from a theoretical standpoint, but unlikely to become important in practice.

6 Conclusion

In summary, this paper presents LightSlice, an algorithm that solves the many-lights problem by clustering and rendering per-slice columns in the transport matrix. Our algorithm can take advantage of the global and local behavior of VPL lighting, thus exploiting the transport matrix structure effectively. Compared to prior methods, our algorithm shows consistent speedups for a variety of lighting scenarios.

Acknowledgements

We would like to thank Chanjuan Wen, Xiaobo An, and Jonathan Denning for their help in preparing this paper. We also want to thank Guillermo M. Leal I\'laguno (Sonmigue\l) and Alvaro Luna Bautista (Museum) for the permission of using their model. This work was supported by NSF (CNS-070820, CCF-0746117), Intel, and the Sloan Foundation.

References


Davidovi\v{c}, T., Kriv\v{a}nek, J., Ha\v{s}an, M., Slusallek, P., and Bala, K. 2010. Combining global and local virtual lights for detailed glossy illumination. ACM Transactions on Graphics 29, 8 (Dec.), 143:1–143:8.

Ha\v{s}an, M., Pellacini, F., and Bala, K. 2006. Direct-to-indirect transfer for cinematic relighting. ACM Transactions on Graphics 25, 3 (July), 1089–1097.


