A Matrix Sampling-and-Recovery Approach for Many-Lights Rendering

Yuchi Huo Rui Wang* Shihao Jin Xinguo Liu Hujun Bao*
State Key Lab of CAD&CG, Zhejiang University

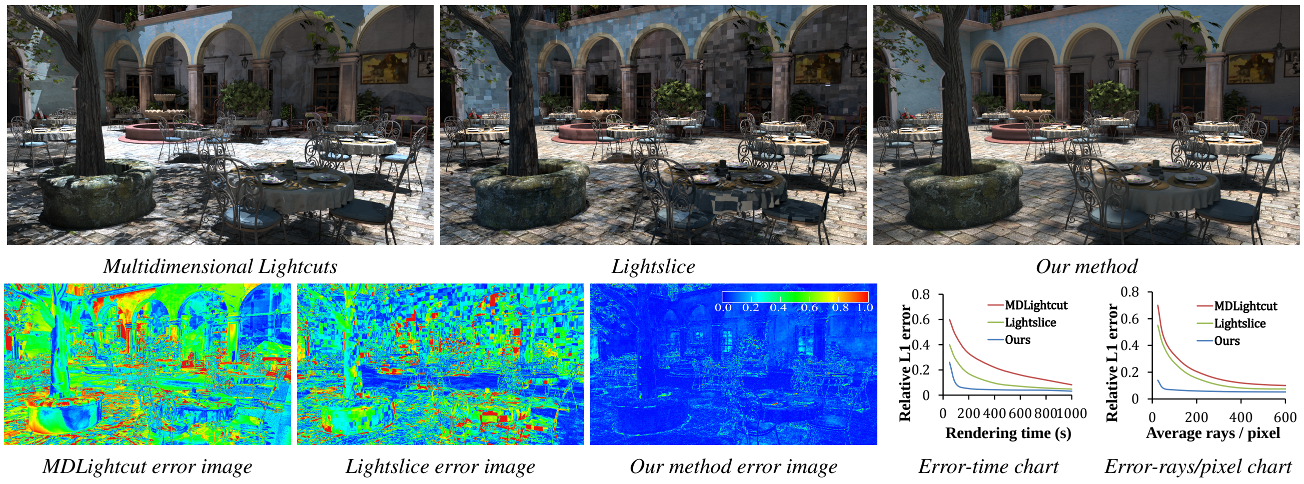


Figure 1: *We render a scene using multidimensional Lightcuts, Lightslice and our method. Three images are rendered using the same sampling rate, i.e. the same average rays per pixel. The relative L1 errors are shown in error images. The error-time chart and error-rays/pixels chart plot the performance of three methods. Results reveal that our matrix sampling-and-recovery approach heavily reduces the required visibility sample rays between lights and surface points. As can be seen, under the same sampling rate, our method captures more occlusions and illumination details, and produces high quality realistic image with much less artifacts comparing to previous methods.*

Abstract

Instead of computing on a large number of virtual point lights (VPLs), scalable many-lights rendering methods effectively simulate various illumination effects only using hundreds or thousands of representative VPLs. However, gathering illuminations from these representative VPLs, especially computing the visibility, is still a tedious and time-consuming task. In this paper, we propose a new matrix sampling-and-recovery scheme to efficiently gather illuminations by only sampling a small number of visibilities between representative VPLs and surface points. Our approach is based on the observation that the lighting matrix used in many-lights rendering is of low-rank, so that it is possible to sparsely sample a small number of entries, and then numerically complete the entire matrix. We propose a three-step algorithm to explore this observation. First, we design a new VPL clustering algorithm to slice the rows and group the columns of the full lighting matrix into a number of reduced matrices, which are sampled and recovered individually. Second, we propose a novel prediction method that predicts visibility of matrix entries from sparsely and randomly sampled entries. Finally, we adapt the matrix separation technique to recover the entire reduced matrix and compute final shadings. Experimental results show that our method heavily reduces the required visibility sampling in the final gathering and achieves 3-7 times speedup compared with the state-of-the-art methods on test scenes.

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

*Corresponding authors: Rui Wang, rwang@cad.zju.edu.cn, Hujun Bao, bao@cad.zju.edu.cn

Keywords: many lights, matrix sampling and recovery

1 Introduction

Realistic image rendering with global illumination involves a recursive rendering equation to integrate multiple bounces of light transports. To efficiently simulate these multi-bounce light transports, Keller [1997] introduced an intermediate representation, the virtual point light (VPL), yielding a fast realistic rendering algorithm. This work inspired a number of methods, which are called as VPL-based rendering. VPL-based rendering first generates a set of VPLs by tracing rays from the real light sources, and then renders the scene by accumulating the direct illuminations from these VPLs.

While the number of VPLs increases to thousands or millions, the efficiency of VPL-based rendering arises as a problem. To address this problem, one kind of methods, known as scalable many-lights rendering [Walter et al. 2005; Walter et al. 2006; Walter et al. 2012; Hašan et al. 2007; Ou and Pellacini 2011], have been proposed. They showed that a large number of VPLs can be clustered and approximated by a small number of representative lights, but still retains the perceptual quality of illumination effects in the result image. Thus, the challenge of rendering many lights converts to the computation and gathering of illuminations from representative lights. The matrix formation of VPL-based rendering [Hašan et al. 2007; Ou and Pellacini 2011] provides an alternative interpretation of many-lights rendering, and introduces matrix sampling techniques to achieve scalable rendering. However, even with aggressive approximations, these methods showed that hundreds or thousands of representative lights per matrix slice are indispensable to preserve local lighting details. Given these lights, the gathering of illumination contributions, especially the computation of visibility, is still tedious and time-consuming.

In this paper, we propose a new matrix sampling-and-recovery scheme to accelerate the gathering step. Our basic idea is to take the advantage of the low-rank property of the lighting matrix [Walter et al. 2006; Huang and Ramamoorthi 2010] that only accurately computes visibility of some entries and then numerically recovers the entire matrix. To achieve this, we use one kind of recently developed methods, named as matrix separation techniques [Candès et al. 2011], that a matrix can be separated from a dense but noisy and corrupted matrix with a few of sampled entries. These techniques have been successfully applied in many applications such as image processing, vision and machine learning.

However, not an arbitrary matrix can be successfully recovered by the matrix separation. The fundamental assumption to apply these techniques is the low-rank of the matrix. Besides that, other criteria, such as the distribution of values or the ratio and pattern of errors spreading in matrix, also impact on the success of matrix recovery. Therefore, to successfully implement our idea, we design preprocess steps to construct smaller matrices from the large full lighting matrix to satisfy the underlying assumptions of the matrix separation. In the preprocess, we first partition the full lighting matrix into a number of reduced lighting matrices, and then we propose a novel matrix prediction step to roughly recover these reduced matrices from some sampled entries. After the preprocess, given these predicted reduced matrices, we adapt one matrix separation method [Shen et al. 2014] to recover matrices and render the final image. Our approach is validated by testing on complex scenes and compared with two state-of-the-art methods.

The main contributions of this work include:

- A novel matrix sampling-and-recovery scheme for efficient many-lights rendering.
- A new light clustering algorithm using sparsely sampled visibility information.
- The designation and usage of a set of predictors to predict visibility of matrix entries from sparsely and randomly sampled entries.
- The introduction and adaption of the matrix separation technique in recovering lighting matrix, especially considering a visibility constraint for separating the lighting matrix and extra error detection and correction for the recovery.

The low-rank assumption of lighting matrix enlightens our new matrix sampling-and-recovery approach, but also brings one major limitation that high-rank parts of the lighting matrix might not be well captured. We believe the low-rank assumption is applicable in many applications, since light transports usually exhibit locality and coherence. Yet rendering highly glossy or specular light transport with VPL is still challenging problem and out of the scope of this paper.

2 Related Work and Background

VPL-based Rendering Keller [1997] introduced Instant Radiosity (IR) method, which is the first realistic rendering algorithm using virtual point lights (VPLs). It traces a number of VPLs from light sources, and then computes the direct illumination from VPLs as the indirect global illumination in the scene.

IR can be viewed as a specific bidirectional path tracing, where the VPLs are the vertices of the light sub-paths, and the camera sub-path has length one. By formulating light sub-path as VPL, IR algorithm shares the light sub-path among all pixels to amortize their cost and suppress the image noise. Such an idea inspired a number of followups known as VPL-based rendering. A comprehensive introduction of VPL-based rendering and many-lights rendering can be found in a state-of-the-art report [Dachsbacher et al. 2014]. Here, we briefly address some main work aiming to improve the VPL-based rendering at different aspects.

To improve the generation of VPLs, Segovia et al. [2006] traced paths from the camera and presented a bidirectional instant radiosity method. Then, Segovia et al. [2007] introduced Metropolis-Hastings sampling to better distribute VPLs. To handle glossy materials that have high frequency of reflections, Hasan et al. [2009] extended VPL to virtual spherical light (VSL) to avoid spiky artifacts. Davidovic et al. [2010] introduced local VPLs to compensate the loss of clamping and offer better approximation for sharp glossy reflections. To accelerate visibility tests, Ritschel et al. [2008] showed that imperfect shadow maps can effectively approximate occlusions between lights and pixels. Georgiev et al. [2012] cached probability of selecting VPL in the scene notably considering visibility information.

Scalable Many-lights Rendering One interesting extension of VPL-based rendering is to use a huge number of VPLs, in the order of thousands, millions and even more, for high quality realistic rendering. The solution to efficiently compute of illumination from such a large number VPLs is recently known as scalable many-lights rendering.

Walter et al. [2005] introduced the first scalable many-lights rendering approach, *lightcut*, to reduce the pixel-light computation cost from linear to sub-linear. The multidimensional lightcut method [Walter et al. 2006] expanded lightcut to multidimensional trees to handle high dimensional integrations while rendering volume scattering, depth of field or motion blur, etc. Recently, Walter et al. [2012] extended such a scalable framework to capture light transports in bidirectional paths, and Wang et al. [2013] extended it to handle extremely large scenes with out-of-core geometry and lights data.

Hašan et al. [2007] formulated the many-lights rendering as a lighting matrix and introduced a new matrix sampling method, the matrix row-column sampling (MRCS), to generate a global set of representative lights for the whole image. Such a global set of lights greatly improve the performance but fail at capturing local lighting, e.g. some locally important lights are not included in the global set due to global balancing. To adapt to these lighting details, Ou and Pellacini [2011] proposed a new method, named as lightslice. It partitions the image into slices, and then refines the global set of lights for each slice. Both of these two approaches, MRCS and lightslice, share the same assumption that the lighting matrix of the VPLs is low-rank. Therefore, a few rows and columns are able to well approximate the matrix. Besides the many-lights rendering, the low-rank property of lighting matrix has also been explored and analyzed in precomputation based rendering [Sloan et al. 2003; Huang and Ramamoorthi 2010]. In this paper, we further explore the low-rank property, and propose a new sample-and-recovery approach for efficient many-lights rendering.

Matrix recovery Matrix recovery from a small number of sampled entries has become an important problem in many areas such as image processing, graphics, vision and machine learning. The basic idea is to utilize the correlations of data in the matrix, i.e. the low-rank property. One type of recovery methods, the matrix completion, pursues a low-rank matrix $\mathbf{L}$ from a partial observed matrix $\mathbf{D}$ by minimizing the rank of the recovered matrix [Candès and Recht 2009]. However, the direct minimization of rank is a NP-hard problem [Candès and Recht 2009]. Thus, the rank of $\mathbf{L}$ is numerically approximated by the nuclear norm, $\|\mathbf{L}\|_*$, defined as the sum of its singular values.

Recently, another type of recovery methods, named as matrix separation, has been developed [Candès et al. 2011; Shen et al. 2014]. It separates the input matrix into two parts: $\mathbf{D} = \mathbf{L} + \mathbf{Z}$, where $\mathbf{L}$ is the latent low-rank part that we want to recover, and $\mathbf{Z}$ is the sparse part that contains errors. The matrix separation problem can be numerically solved using soft thresholding and Lagrange multipliers [Lin et al. 2009; Shen et al. 2014]. As far as we know, our work is the first method adapting matrix separation techniques in rendering.

While solving the matrix separation, we use matrix factorization to approximate the matrix we want to recover. There are several rendering problems involving matrix factorization, e.g. separable BRDF [Kautz and McCool 1999], environment mapping [McCool et al. 2001], precomputed radiance transfer [Sloan et al. 2003], etc. Compared with these methods using matrix factorization to compress the matrix data, our work targets on recovering matrix entries.

3 Overview

3.1 Mathematical Overview

Matrix Formulation of Many-lights Rendering. Many-lights rendering can be formulated as a matrix computing problem [Hašan et al. 2007; Ou and Pellacini 2011]. Let $\mathbf{A}$ be the lighting matrix, where each column represents a VPL, and each row corresponds to a surface point. One entry $\mathbf{A}_{ij}$ represents the illumination contribution from light j to surface point i. The final rendering of surface sample i is the sum of the contributions from all VPLs as:

$$I(i) = \sum_j \mathbf{A}_{ij} = \sum_j \mathbf{E}_{ij}\mathbf{V}_{ij} \tag{1}$$

$$\mathbf{E}_{ij} = \mathbf{M}_{ij}\mathbf{G}_{ij}I_j \tag{2}$$

where $\mathbf{V}_{ij}$ and $\mathbf{E}_{ij}$ are the visibility and the illumination contribution of light j at point i. $\mathbf{E}_{ij}$ is computed by the material term $\mathbf{M}_{ij}$, geometry term $\mathbf{G}_{ij}$ and the intensity I_j of light j.

Since the number of VPLs is very large, naively computing all the entries in matrix $\mathbf{A}$ is impractical. Similar to Ou and Pellacini [2011], we first partition the lighting matrix $\mathbf{A}$ into smaller slice matrices, and then construct the reduced lighting matrix for each slice. Since these reduced lighting matrices of slices are computed and processed independently, let us use $\mathbf{L}$ to denote one reduced lighting matrix, where columns of $\mathbf{L}$ are a set of representative lights, $\mathbb{C}$, clustered from VPLs. Therefore, the rendering of surface point i in the reduced matrix $\mathbf{L}$ is computed as:

$$I(i) = \sum_{k\in\mathbb{C}} \mathbf{L}_{ik} = \sum_{k\in\mathbb{C}} \mathbf{V}_{ik}\mathbf{E}_{ik}, \;\; \mathbf{E}_{ik} = \mathbf{M}_{ik}\mathbf{G}_{ik}I_k \tag{3}$$

where I_k is the representative light in cluster $\mathbb{C}$ and $\mathbf{E}_{ik}$ is the illumination contribution from the representative light to point i.

Since $\mathbf{E}_{ik}$ is easy to compute, the challenge to efficiently compute the reduced matrix $\mathbf{L}$ is the computation of the visibility term, where it incurs ray intersection tests in complex scenes. Even the reduced matrix $\mathbf{L}$ only has hundreds or thousands of columns, which is much smaller than the full lighting matrix $\mathbf{A}$, the cost to directly sample visibility of all entries is still very high.

We observe that without the visibility term, the illumination contribution $\mathbf{E}_{ik}$ is still a good clue to the real value, where we can treat it as corrupted or noise data in the matrix. Therefore, the computation of the reduced lighting matrix $\mathbf{L}$ can be formulated as a matrix separation problem that recovers a matrix from a corrupted matrix. In our case, the corrupted matrix is filled by a small number of accurate entries with sampled visibility and other corrupted entries with estimated visibility. We denote such a corrupted matrix as $\mathbf{D}$:

$$\mathbf{D}_{ik} = \begin{cases} \mathbf{V}_{ik}\mathbf{E}_{ik}, & \mathbf{V}_{ik} \text{ is actual visibility.} \\ \overline{\mathbf{V}}_{ik}\mathbf{E}_{ik}, & \overline{\mathbf{V}}_{ik} \text{ is estimated visibility.} \end{cases} \tag{4}$$

Matrix Recovery by Matrix Separation. Matrix separation has been recently developed [Candès et al. 2011; Shen et al. 2014]. Specifically in our scenario, the reduced lighting matrix $\mathbf{L}$ can be separated from the corrupted matrix $\mathbf{D}$ with a sparse error matrix $\mathbf{Z}$, $\mathbf{D} = \mathbf{L} + \mathbf{Z}$, by solving the following minimization:

$$\begin{aligned} \min_{\mathbf{L},\mathbf{Z}} \quad & \|\mathbf{L}\|_* + \lambda\|\mathbf{Z}\|_1 \\ s.t. \quad & P_\Omega(\mathbf{L}+\mathbf{Z}) = P_\Omega(\mathbf{D}) \end{aligned} \tag{5}$$

where λ is weighting coefficient, $\| \, \|_1$ denotes the ℓ_1 norm by regarding the matrix as a vector, Ω is the index subset of entries with accurate visibility, $\Omega \subseteq \{(i,j) : 1 \le i \le m, 1 \le j \le n\}$, P_Ω is the projection of the matrix onto the subspace of matrix whose entries are restricted to Ω. Such a minimization indicates that we can only sample a subset of entries of a matrix, and use these sampled entries with other unsampled or estimated entries to recover $\mathbf{L}$. Technically, to exactly and efficiently perform the recovery, Candes et al. [2011] showed that $\mathbf{L}$ should have following assumptions:

1. $\mathbf{L}$ is low-rank;
2. The singular vectors of $\mathbf{L}$ are reasonably spread out;
3. $\mathbf{Z}$ is reasonably sparse with random sparsity patterns;

The first assumption requires the rank of $\mathbf{L}$ is low, which is usually true for the reduced lighting matrix. The second assumption requires that no row or column of $\mathbf{L}$ dominates any basis. Given a matrix, if there are many columns have similar features, we can say that they share the same group of basis or the singular vectors of the matrix are spread out in the matrix. The measurement of how well singular vectors are spread out in the matrix is usually named as matrix coherence [Candès et al. 2011]. The third assumption says that the density of $\mathbf{Z}$ should not be dense in any regions, otherwise the separating problem become more difficult because it's unclear whether these dense regions should be separated to $\mathbf{L}$ or to $\mathbf{Z}$.

The matrix separation problem can be numerically solved using soft thresholding and Lagrange multipliers [Lin et al. 2009]. Recently, Shen et al.[2014] proposed an augmented Lagrangian alternating direction method based on low-rank factorization. It first detects the rank of $\mathbf{L}$, and then factorizes $\mathbf{L}$ into two matrices, $\mathbf{L} = \mathbf{XY}$. In this way, a new minimization derived from Eq. 5 is proposed as:

$$\begin{aligned} \min_{\mathbf{X},\mathbf{Y},\mathbf{Z}} \quad & \|P_\Omega(\mathbf{Z})\|_1 \\ s.t. \quad & P_\Omega(\mathbf{XY}+\mathbf{Z}) = P_\Omega(\mathbf{D}) \end{aligned} \tag{6}$$

Through numerical analysis, they showed that besides aforementioned three assumptions, one new assumption to successfully separate $\mathbf{L}$ using their approach is:

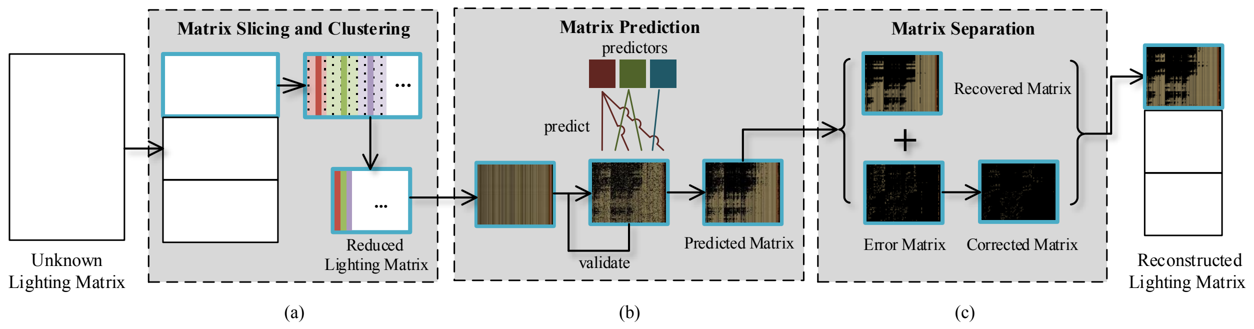


Figure 2: *Algorithm overview.*

4. The low-rank matrix, **L**, is not dominated by the sparse matrix, **Z**, in magnitude,

This assumption suggests that the error values in the error matrix should not be very large comparing to the true values we want to recover.

Compared with the original nuclear norm minimization Eq. 5, the new minimization Eq. 6 has some drawbacks, e.g. the non-convexity in the minimization may prevent the optimization from getting a global solution, and the requirement of an initial rank estimation. But, the optimization of new minimization is faster and more efficient. We find it works very well in our application.

3.2 Algorithm Overview

Our basic idea is to sample a small number of entries of the reduced matrix and apply the matrix separation technique to recover the entire matrix. However, it is non-trivial to directly apply matrix separation on the lighting matrix or on reduced lighting matrices produced by previous work [Walter et al. 2006; Ou and Pellacini 2011]. This is because that the reduced matrix may not be in a good shape for matrix separation, i.e. violating aforementioned assumptions, thereby resulting in poor matrix recovery (Fig. 3(a)(b)). To better implement our idea, we design a three-step algorithm. The algorithm overview is shown in Fig. 2. These steps are basically designed to process the lighting matrix in a way that it could be sampled and recovered by matrix separation. Two preprocess steps are proposed to process the lighting matrix into reduced matrices before we actually separate them. These two steps are based on the observation that these four assumptions can be classified into the requirements on the recovered matrix **L** (assumption 1, 2 and 4), and on the sparse matrix **Z** (assumption 3 and 4). Therefore, we separately process lighting matrix to satisfy the assumptions on **L** and **Z** respectively.

More precisely, in the first step (Fig. 2(a)), we partition the full lighting matrix into a small number of smaller matrices, the reduced lighting matrices. The rows of full lighting matrix are firstly grouped into slices, and then, in each slice, lights are clustered into columns, as representative VPLs. A new VPL clustering algorithm is designed to bound the magnitude of each column. The detail of this step is discussed in Sec. 4. The basic insight of this step is to construct matrices that approximately satisfy assumption 1, 2 and 4.

To satisfy the assumption on the sparse matrix **Z** (assumption 3), we design a novel lighting matrix prediction scheme in the second step (Fig. 2(b)). That is, for each reduced matrix, we sparsely sample entries, and then train three predictors to predict the visibility of unsampled entries (Sec. 5). The prediction procedure is taken iteratively with a sample-and-validate strategy. If the predicted visibility does not pass the validation, we iterate back with new samples to further train predictors. Otherwise we use the trained best predictor to fill up unsampled entries by multiplying the shading values with 0 or 1 predicated visibility values. Since the lighting matrix is low-rank and values in many regions of the matrix are usually smooth, the matrix prediction step well predicts unsampled entries, and results in sparse errors. The basic insight is that these predictors can be regarded as smooth and low-frequency approximations to real visibilities. After the prediction, the smooth and low-frequency part of visibility values in matrix are accurately predicted, and the errors are probably sparse.

In the third step Fig. 2(c), we separate the predicted reduced matrix into a low-rank matrix and an error matrix (Sec. 6). In some cases, the reduced matrix **L** is not always low-rank. To handle high-rank parts in the matrix, we detect them in the error matrix and directly sample the values instead of using recovered ones. These high-rank parts are combined with the recovered low-rank matrix to produce the final image.

4 Reduced Lighting Matrix Construction

The basic motivation of this step is to generate reduced lighting matrix with spread singular vectors (assumption 2) and make sure the reduced matrix not be dominated by the sparse error matrix (assumption 4). To achieve it, first, we slice the full lighting matrix by grouping rows, and then propose a new light clustering method to group lights into columns, thereby forming reduced lighting matrices.

4.1 Matrix Slicing

We adapt the method [Ou and Pellacini 2011] to group rows of the full lighting matrix. Specifically, surface points are represented in a $6D$ space consisting of position and normal: (x, y, z, n_x, n_y, n_z). The normal components n_x, n_y, n_z are weighed by the scene size so as to normalize the contributions of position and normal. Then, we build a KD-tree by iteratively splitting the nodes along their longest axis. The splitting stops when the associated surface points is less than a prescribed number, which is empirically set as 1024 in our implementation. After clustering the points, the lighting matrix is partitioned into slices accordingly, each of which corresponds to a cluster.

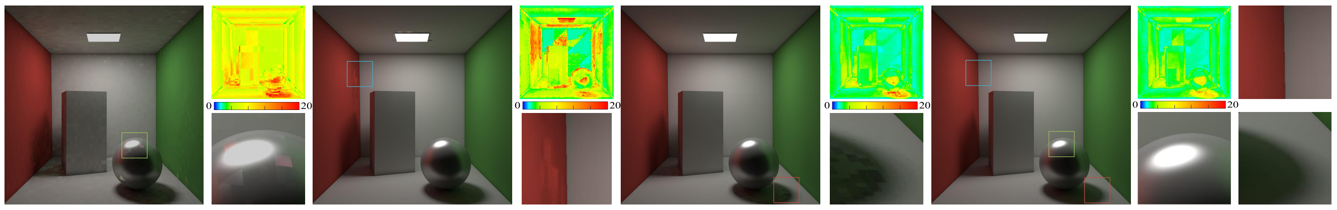

(a) (b) (c) (d)

Figure 3: *A Cornell Box scene is used to test different light clustering approaches. The upper-right image in each sub-figure shows the matrix coherence of reduced lighting matrices. The bottom-right shows a zoomed detail image. (a) light clustering proposed in [Ou and Pellacini 2011], (b) light clustering proposed in [Walter et al. 2006], (c) our light clustering without using sparse visibility samples. (d) our light clustering using sparse visibility samples.*

4.2 Per Slice Light Clustering

Overall, the goal of our per slice light clustering is to produce a reduced lighting matrix with reduced columns, decreased matrix coherence and bounded magnitudes of matrix entries. However, the cost to quantitatively compute the matrix coherence is high, where QR or SVD decompositions are usually required [Mohri and Talwalkar 2011]. Instead, we set a specific hard-threshold on the magnitude of illumination from clustered VPL to avoid the domination of some columns, thereby reduce the matrix coherence. To implement our sparse sampling-and-recovery idea, we also randomly sample visibility between clustered light and some surface points to better estimate the contribution from the light to all points in the slice. Our per slice light clustering algorithm takes two steps: first, initialize a coarse lightcut, then split the cut nodes adaptively by estimating the maximum magnitudes.

4.2.1 Initial Clustering

We adapt the multidimensional lightcut method [Walter et al. 2006] to fast compute the initial cut nodes. All surface points in one slice is regarded as a gathering tree. Using this gathering tree, starting from the root node of VPL tree, we keep splitting the light node with the highest error bound, until the number of cut nodes reaches a prescribed number. No visibility test is performed during the splitting procedure. The prescribed number of light clusters is set to 200 in our implementation.

4.2.2 Light Node Splitting

Once having the initial cut nodes, we iteratively split them using a magnitude threshold. Instead of accurately computing the contributions from light to all surface points in the slice, we only estimate a maximum contribution, and split node when its maximum contribution is greater than a *splitting threshold*. After all nodes on the cut are iteratively split, the final light nodes are used as columns to construct the reduced lighting matrix. The actual sampling ratio is given by a prescribed parameter ρ, which is set as 1% in our implementation, and works well in practice. Next, we give more details.

Maximum Contribution Estimation. A naive way to estimate the maximum contribution of light to surface points in slice is to use Eq. 3. However, only considering one light node in the tree is likely insufficient to give reliable result. We improve the reliability by borrowing geometry terms from nearby cut nodes, and using a small number of actually sampled visibility. Specifically, the maximum contribution of node j is estimated as follows:

$$\Gamma_j = \max_{k\in\mathcal{N}(j),i} \{\mathbf{M}_{ik}\mathbf{G}_{ik}\} \times \max_{k\in\mathcal{N}(j),(i,k)\in\Omega} \{\mathbf{V}_{ik}\} \times I_j \tag{7}$$

where $\mathcal{N}(j)$ is a set of nearby light nodes in the cut, i is surface points in the slice, Ω is the set of entry indices at which the matrix is sampled. Light nodes in $\mathcal{N}(j)$ are collected by a range search in the light tree, where the nearby light nodes are collected by a bound of light intensity. Please refer to the supplementary document for more details of this range search.

Splitting Threshold We decide the splitting threshold, Γ, according to the sum of all maximum contributions of columns as follow:

$$\Gamma = \alpha \sum_{j\in\mathbb{C}} \Gamma_j \tag{8}$$

where $\mathbb{C}$ is the set of current cut nodes, α is a prescribed scalar parameter. α is empirically found such that reasonable number of columns split. $\alpha = 0.02$ gives satisfactory rendering results in our experiments. More analysis on the threshold α can be found in the supplementary document.

If the maximum contribution of one light node is larger than the splitting threshold, Γ, we split this light node, and add its children into the cut. After estimating the maximum contributions of two children, we use them to update the splitting threshold and trigger another splitting when necessary.

In Alg. 1, we show the pseudo-code of the entire process of node splitting. We maintain a list, $currentCut$, which is initialized by light cut nodes. Then we split the node whose maximum contribution is larger than a given threshold, and replace it with its two children in $currentCut$. After processing this node, we check nearby nodes and split them if necessary.

In Fig. 3, we show a comparison of results using different light clustering approaches. Fig. 3(a) shows a result of the standard clustering used in [Ou and Pellacini 2011]. Fig. 3(b) shows a result using the lighting clustering proposed in [Walter et al. 2006], which is also used to create our initial light cut. Fig. 3(c) shows a result using our light clustering algorithm but without the sparse visibility sampling step. Fig. 3(d) is generated by our light clustering method. Besides the final results, we also give the accurately computed matrix coherence, where the lower coherence is the better for matrix separation. As can been seen, the standard clustering produces highest matrix coherence values per slice, which fails to capture shadow boundaries, yielding many visual artifacts. Moreover, the consideration of some visibility samples help us to better refine the cut nodes.

5 Matrix Prediction

Once having a reduced lighting matrix, before we recover it, we need entry values of the matrix are dense but their errors are sparse, i.e. $\mathbf{L}$ is dense but $\mathbf{Z}$ is sparse (assumption 4). According to the definition of our corrupted matrix $\mathbf{D}$, the error is introduced by the visibility. Observing that visibility from VPLs to surface points

Algorithm 1 Light Node Splitting

```
1:  function LighNodeSplitting($initialCut$)
2:      $currentCut \leftarrow initialCut$
3:      for each $node$ in $currentCut$ do
4:          Split($node$)
5:      end for
6:  end function
7:
8:  function Split($currentNode$)
9:      // Search nearby cut nodes in $currentCut$
10:     $neighbor \leftarrow$ GetNearby($currentNode$)
11:     $maxContribution \leftarrow$ CalMaximum($neighbor$)
12:     if $maxContribution > splittingThreshold$ then
13:         $left, right \leftarrow$ GetChildren($node$)
14:         replace $currentNode$ by $left, right$ in $currentCut$
15:         update $splittingThreshold$
16:         Split($left$)
17:         Split($right$)
18:         // nearby cut nodes might split given new $left, right$
19:         for each $node$ in $currentNode$ do
20:             if $left, right \in$ GetNearby($node$) then
21:                 Split($node$)
22:             end if
23:         end for
24:     end if
25: end function
```

has locality, we explore this locality by training some visibility predictors to predict unsampled entries from a small set of sampled entries. We name it matrix prediction.

5.1 Predictors

We design three types of predictors for visibility: kNN predictor, Linear predictor, and Naïve Bayes predictor. Each predictor is used to predict the visibility from one VPL to all surface points in the reduced lighting matrix, i.e. entries of one column, which bear the locality property of the visibility function.

kNN Predictor predicts the visibility using k nearest neighbors' visibility. It works on each column. To predict $\mathbf{V}_{ij}$ at column j, it first finds k nearest neighboring surface points that have been sampled in column j. Then it randomly picks a surface point from these sampled neighbor surface points, and uses its visibility as the prediction for $\mathbf{V}_{ij}$. We use 3 nearest neighbors.

To predict with the kNN predictor is very fast, but to find the nearest neighbors is not. Instead of using a spatial acceleration data structure, in this work, we adopt a spatial curve based kNN searching method [Liao et al. 2001]. It is an approximate kNN method based on a number (dimension +1) of shifted Hilbert Spatial Curve. Though it is an approximate method, it is very fast.

More precisely, we project all the surface points onto a 2D tangent plane defined by the average normal of the slice. Then we construct 3 Hilbert curves to sort the surface points. To search for k nearest neighbors, we travel along the Hilbert curves and examine k predecessors and k successor points.

Linear Predictor uses a plane to divide the space of shading points into two half spaces. One is shadowed (visibility is 0) and the other is not (visibility is 1). Therefore, the linear predictor is suitable for columns where there is a straight shadow boundary.

Let's consider a column j. The separating plane can be represented by a 4D vector $W = (a \;\; b \;\; c \;\; d)^T$. Let $\mathbf{X}$ be a matrix whose row vectors are the homogenous coordinate of the sampled surface

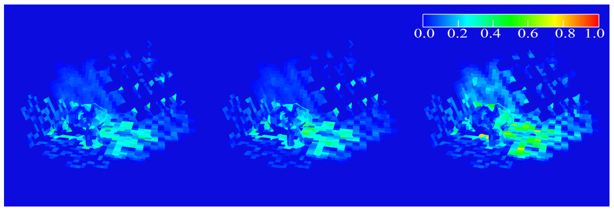


Figure 4: *Prediction error ratio of (Left) kNN predictor , (Middle) Linear predictor, (Right) Naive Bayes predictor. A view of the scene can be found in Fig. 5 (Left).*

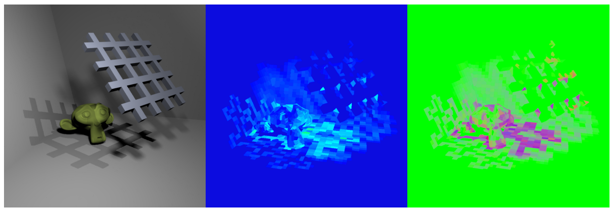


Figure 5: *Optimized predictor. (Left) rendered image, (Middle) prediction error ratio and (Right) the optimal selection of predictors per slice, where the percentages of columns predicted by three different predictors are rendered in RGB channels, red for kNN predictor, green for Bayes predictor, and blue for linear predictor.*

points, Y be column vector related to the sampled visibility values. It is 1 for visible, and -1 for invisible. Then we have equations: $\mathbf{X}W = Y$, for the separating plane W. Solve the above equation in least squares, the separating plane can be computed as follow:

$$W = (\mathbf{X}^T\mathbf{X})^{-1}Y, \tag{9}$$

Now, for any surface point q, we can predict its visibility V_q by the following possibilities:

$$\begin{aligned} P(V_q = 0) = \max\{0,\ 0.5 - f(q)\} \\ P(V_q = 1) = \max\{0,\ 0.5 + f(q)\} \end{aligned} \tag{10}$$

where $f(q) = \bar{q}^T W$, and $\bar{q}$ is the homogeneous coordinates of q. It is easy to validate that $P(V_q = 0) + P(V_q = 1) = 1$.

Naive Bayes Predictor. Both kNN predictor and linear predictor work inside a single column, i.e. the samples in a column are used to predict the unknown entries in the same column. But Bayes predictor predicts values from nearby columns to take advantage of the spatial coherence among lights. We use a naive Bayes predictor. More precisely, we define the feature vector as the (i, j) index of the entry, the concept function is the visibility. For each column j, the feature vector space consist of the entry indices in 5 nearby columns around j, marked as N_j. Specifically, the ratio of predicting (i, j) is $P(v|i, j) = \frac{P(i|v)P(N_j|v)P(v)}{P(i)P(N_j)}, v = 0, 1.$

Fig. 4 shows a false-color drawing of the prediction error ratio (averaged per slice for better visualization). Prediction error ratio is the percentage of wrong prediction of all prediction samples. It is obvious that there may not exist a best predictor for all slices. Bayes predictor is very efficient and reliable in smooth region. Linear predictor works well in strait shadow boundary and kNN can fit arbitrary occlusion. Randomness is introduced to predict visibility, thus produces some sparse and indeterminant errors.

Algorithm 2 Matrix Prediction and Validation

```
function PredictMatrix($reducedMatrix$)
    put all columns in $reducedMatrix$ into $activeColumns$
    for $i = 1 \rightarrow maxIteration$ do
        for each $column$ in $activeColumns$ do
            if PredictColumn($column$) then
                remove $column$ from $activeColumns$
            end if
        end for
    end for
end function

function PredictColumn($column$)
    $newSamples \leftarrow$ SparselySample($column$)
    evaluate the accurate values of $newSamples$
    for each $predictor_i$ do
        use $predictor_i$ to predict the values of $newSamples$
        $errorRatio_i \leftarrow$ error ratio of predicted values
    end for
    if $\min(errorRatio_i) < predictionErrorRatio$ then
        use $predictor_i$ with minimum $errorRatio_i$ to predict
        all unknown entries in $column$
        return true
    else
        add $newSamples$ to $column$
        improve all $predictor_i$ with $newSamples$
        return false
    end if
end function
```

5.2 Prediction and Validation

Given these three predictors, we use them to predict visibility of matrix entries column by column from a small number of accurately sampled visibilites. However, the prediction may not be correct for all entries, especially when the number of samples is small. To improve the prediction, we take another validation pass. The validation is started by randomly selecting another set of entries to accurately sample visibilities. Then, these newly sampled entries are used to test on predictors. The predictor that best predicts these samples, i.e. producing the least error ratio, is selected as the best predictor. If this least error ratio is less than a predefined prediction error ratio, e_p, we regard the prediction of this column success, and will not take any further sampling and prediction on this column. Otherwise, these new samples are used to further refine predictors. Such prediction, sampling and validation on columns are iteratively executed until the prediction on matrix is successful or a fixed number of iterations is reached. In our method, we set the maximum iteration number as 20. Once the prediction of all columns are successful, we use these column-wise best predictors to predict columns of the final predicted matrix, $\mathbf{D}$. The pseudocode of prediction and validation algorithm can be found in Alg. 2. Since the Bayes predictor shares samples from other columns, the order of predicting different columns can effect the performance. Therefore, we split the prediction process into $maxIteration$ loop. In each iteration, we only process columns in the active list. For each column, we sparsely sample, predict and validate it using ρ percentage of new samples.

In Fig. 5, we visualize the choice of different predictors in the simple scene. Fig. 5(b) shows the prediction error ratio optimized by three predictors. Fig. 5(c) visualize the optimal selection of predictors. The percentages of columns predicted by three different predictors are rendered in RGB channels. As can be seen, the linear predictor is the best for area with straight shadow boundary.

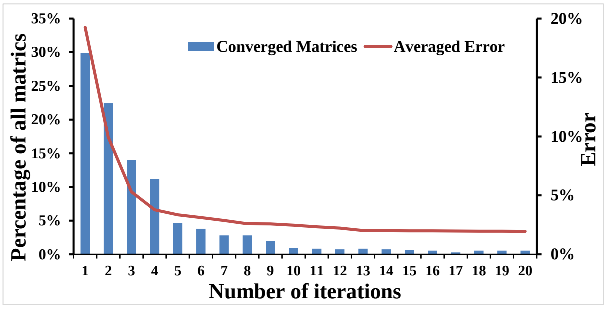


Figure 6: *Convergence with iterations. (Left vertical axis) We show a histogram of converged matrices at different iterations. Only after 4 iterations,* 95% *reduced matrices are successfully separated. After 10 iterations, most of separation of matrices are converged. (Right vertical axis) The averaged separation error is shown as a red curve.*

KNN predictor is suitable for complex geometry and shadow environment, such as the curve of the monkey and the hybrid shadow area. Naive Bayes predictor is the optimal selection for low-rank region because it can borrow information from nearby columns and its computation cost is very low.

6 Matrix Separation

In this section, we present the process to finally recover the reduced lighting matrix from the corrupted matrix $\mathbf{D}$ by separating it into two parts: $\mathbf{D} = \mathbf{L} + \mathbf{Z}$, where $\mathbf{L}$ is the low-rank approximation of the true reduced lighting matrix, and $\mathbf{Z}$ is the error part. We use the augmented Lagrangian alternating direction separation method [Shen et al. 2014] to perform the separation.

6.1 Separation Algorithm

The separation algorithm requires an approximate estimation of matrix rank. We use a partially SVD factorization [Halko et al. 2011] to estimate the rank. Once having the rank k, we can approximate the latent reduced lighting matrix as the product of two rank-k matrices: $\mathbf{L} = \mathbf{XY}$, where $\mathbf{X} \in R^{m\times k}$ and $\mathbf{Y} \in R^{k\times n}$, and reorder the separation equation as $\mathbf{D} = \mathbf{XY} - \mathbf{Z}$. Additionally, we observe that ideally, the error value at entry (i, j) between real value and the corrupted value is computed as $\mathbf{Z}_{ij} = (\mathbf{V}_{ij} - \overline{\mathbf{V}}_{ij})\mathbf{E}_{ij}$. Since the visibility is 0 or 1, the ideal error value should be 0 or $\mathbf{E}_{ij}$ accordingly. Thus, such an ideal error distribution provides us an extra constraint to better solve the latent reduced lighting matrix.

We optimize the minimization of Eq. 6 to recover $\mathbf{L}$ as:

$$\min_{\mathbf{X},\mathbf{Y},\mathbf{Z}} \quad \|P_\Omega(\mathbf{Z})\|_1$$

$$s.t. \quad P_\Omega(\mathbf{XY} + \mathbf{Z}) = P_\Omega(\mathbf{D}) \tag{11}$$

$$\mathbf{H} = 2\mathbf{Q} \circ \mathbf{Z} - \mathbf{1}, \;\; \|\mathbf{H}\|_F^2 = c. \tag{12}$$

where Eq. 12 is the newly introduced constraint. The matrix $\mathbf{Q}$ is defined as $\mathbf{Q}_{ij} = 1/\mathbf{E}_{ij}$, $\circ$ denotes element-wise matrix multiplication, and matrix $\mathbf{1}$ is the matrix with all 1 entries. By individually normalizing the illumination contribution, the error matrix $\mathbf{Z}$ is scale to normalized error matrix $\mathbf{H}$. The ideal normalized error matrix $\mathbf{H}$ should have -1 or 1 values. Instead of constraining element-wise values of $\mathbf{H}$ to -1 or 1, we relax on the entire matrix, i.e. constrain the Frobenius norm of matrix $\mathbf{H}$ to the number of entries, c. As can been seen, for an ideal $\mathbf{H}$, i.e. all entries are -1 or 1, it satisfies the equation $\|\mathbf{H}\|_F^2 = c$. To better understand this constraint, it can be regarded as a relaxation of $\mathbf{H}$ from the vertices of a

		sanmiguel	carnival	room	sponza
scene	triangles	9.2M	6.2M	735K	290K
	resolution	1920×1080, 4SPP	1920×1080, 4SPP	1600×1200, 4SPP	1600×1200, 4SPP
	VPLs	1M	1M	1M	1M
	surface points	8.5M	4.1M	7.7M	7.7M
Our method	avg. rays/pixel	46	96	41	28
	avg. sampling rate	4.4%	5.2%	5.8%	4%
	maximum columns	2000	4000	2000	2000
	matrix generation time(s)	70	92	45	38
	matrix separation time(s)	15	33	10	7
	total rendering time (s)	85	125	55	45
	avg. relative error	6.40%	4.50%	4.20%	4.20%
Multidim. lightcut (equal quality)	avg. rays/pixel	1250	981	831	635
	max cut size	2000	4000	2000	2000
	total rendering time (s)	1583	941	795	821
	avg. relative error	6.30%	4.20%	4.70%	4.30%
Multidim. lightcut (equal rays/pixel)	avg. rays/pixel	46	94	40	41
	max cut size	60	200	50	80
	total rendering time (s)	75	121	58	50
	avg. relative error	47.89%	19.00%	47%	33%
Lightslice initial clustering	slice number	3147	1492	2742	2655
	clustering time (s)	71	39	61	57
Lightslice gathering (equal quality)	avg. rays/pixel	1145	1953	1088	1164
	columns	300	500	300	300
	rendering time (s)	542	401	420	372
	avg. relative error	6.20%	9.70%	5.60%	4.30%
Lightslice gathering (equal rays/pixel)	avg. rays/pixel	47	98	43	31
	columns	50	100	45	35
	rendering time (s)	35	40	24	21
	avg. relative error	40.36%	16%	39%	42%

Table 1: *Summary of the testing scenes and the rendering results, compared with the multidimensional lightcut and lightslice methods. Timings are measured in seconds.*

c dimensional super-cube into the cube's externally tangent super-sphere [Boyd and Vandenberghe 2004].

Iterative Solver We adapt the alternating direction augmented Lagrangian method [Shen et al. 2014] to solve the minimization. More details can be found in the supplementary document. The iteration stops when the $||\mathbf{XY}+\mathbf{Z}-\mathbf{D}||_F$ converges, or a maximum iteration number is reached. We set the maximum iteration number as 20 in our implementation. Fig. 6 shows the convergence of slices with iterations. From the histogram of converged matrices at different iterations, it can be seen that only after 4 iterations, 95% reduced matrices are successfully separated.

6.2 Error Detection and Correction

Using the matrix factorization, the recovered matrix $\mathbf{L}$ is restricted to rank-k. To compensate the high-rank part missed by the matrix separation, we employ a error detection and correction step afterward. Since the errors are all in the error matrix $\mathbf{Z}$ and should be sparse, if the high-rank part has significant impact to the final images, it is unlikely to be sparse (likely to be dense). In another word, we can examine the matrix $\mathbf{Z}$, and find the high-rank part by identifying the dense area in $\mathbf{Z}$.

Therefore, we first convert $\mathbf{Z}$ to a binary image (nonzero entries are set to 1) and perform a Gaussian filtering with a 7×7 kernel, to obtain a density map. Then we identify the dense area with prescribed density threshold θ_d. For each entry index (i, j) of the dense area, the corresponding entry $\mathbf{L}_{ij}$ is accurately evaluated by directly computing it. In our practice, only very small part, less than 1% of entries, are resampled.

7 Results

We implement our algorithms in Microsoft Visual Studio 10, and use Intel MKL as the linear algebra library. All experiments tested in the paper is conducted on a PC with 2 Intel Xeon E5620 CPU and 32GB memory.

7.1 Parameter setting

In our method, several parameters are used at different stages. The main parameters used in these experiments are shown below:

The prescribed parameters in our implementation	
Sampling ratio	$\rho = 1\%$
Column splitting parameter	$\alpha = 0.02$
Prediction error ratio	$e_p = 10\%$
Density threshold for high rank	$\theta_d = 0.2$

ρ is the parameter used to control the sample ratio of each column at each prediction iteration. Since we use prediction and validation iterations to adaptively distribute samples, ρ is not decisive to the overall sample ratio and image quality. α is a parameter used to control the maximum entry values. In Fig. 9(d), we compare different settings and discuss them in Sec. 8.3. More discussion on this parameter can be found in the supplementary document. The e_p is the error threshold used in matrix prediction. The larger it is, the less samples are generated to sample the visibility. In Fig. 9(a)-(c), we show results using different number of visibility samples. Corresponding discussions also can be found in Sec. 8.3. θ_d determines the strength of error detection and correction. While θ_d is lower, more entries will be regarded error entries and resampled. The above table lists a group of suggested values work well for all our complex test scenes.

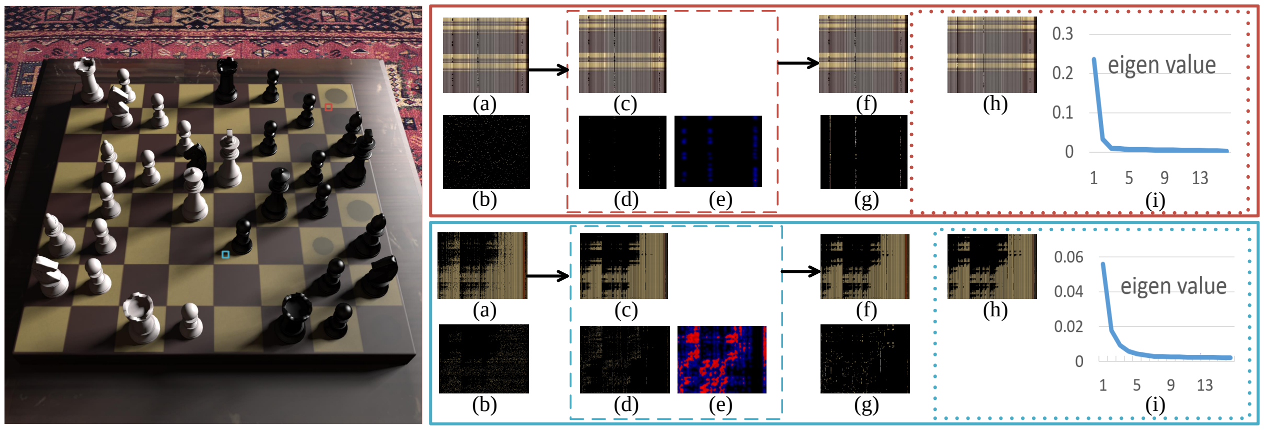


Figure 7: *An example of sampled and recovered matrices. Left: one final rendered image; Top/Bottom right: the light matrices for the slices framed in red/blue in the left image and a curve drawing the Eigen values, where (a) corrupted matrix* $\mathbf{D}$*, (b) sparse sampled entries, (c) separated latent matrix* $\mathbf{L}$*, (d) error matrix* $\mathbf{Z}$*, (e) density map computed from error matrix, (f) final recovered matrix* $\mathbf{L}$*, (g) errors comparing to the reference matrix, (h) the reference matrix by accurately computed all entries, (i) eigenvalues of reference matrix.*

7.2 Test Scenes

We compare our method with the multidimensional lightcut method [Walter et al. 2006] and the lightslice method [Ou and Pellacini 2011] on 4 test scenes. Statistics of testing scenes are shown in Tab. 1. The implementation of multidimensional lightcut and the lightslice methods are obtained online through the source code provided by the authors of lightslice method. All methods are accelerated by 16 threads. We separately add up the time used in computing initial clustering and the final lights gathering of lightslice, because the initial clustering is relatively independent to the final gathering. But, the final rendering time of lightslice is summed by the initial clustering time and the gathering time. In test scenes, we use omni, directional and diffuse oriented VPL [Walter et al. 2005]. For different scenes, we first set the direct light sources and then use these direct light sources to generate multi-bounce VPLs. The carnival scene contains several hundreds of direct point light sources, serval direct area light source, and one environmental light. The room scene has a large direct area light source from outside, and with many tiny objects and glossy surfaces. The sponza scene exhibits concave area lit by inter-reflections from the sky dome. For these test scenes, the maximum memory consumptions of VPLs and VPL tree are 48 MB and 128MB, respectively. The separation of one reduced light matrix requires approximately 20MB. Since reduced matrices are processed independently, the total memory used by matrix prediction and separation is determined by the number of matrices being processed in parallel.

All images are rendered with 2×2 super samples. In our method, super sampling only affects matrix slicing and visibility sampling. After matrix slicing, if super samples are clustered to different slices (mostly due to silhouette or bumpy detail), they are treated as individual surface point in each slice. At sampling visibility, e.g. sample entry $\mathbf{D}_{ij}$, we randomly pick one super sample in pixel i to compute the actual visibility.

For each scene, we fully sample all VPLs to generate the reference image. The differences between the reference image and the images generated by different methods are measured by the relative $L1$ absolute error, i.e. $e = \frac{||L_o - L||_1}{L_o}$, where L_o is the intensity in the reference image and L is that generated by different methods.

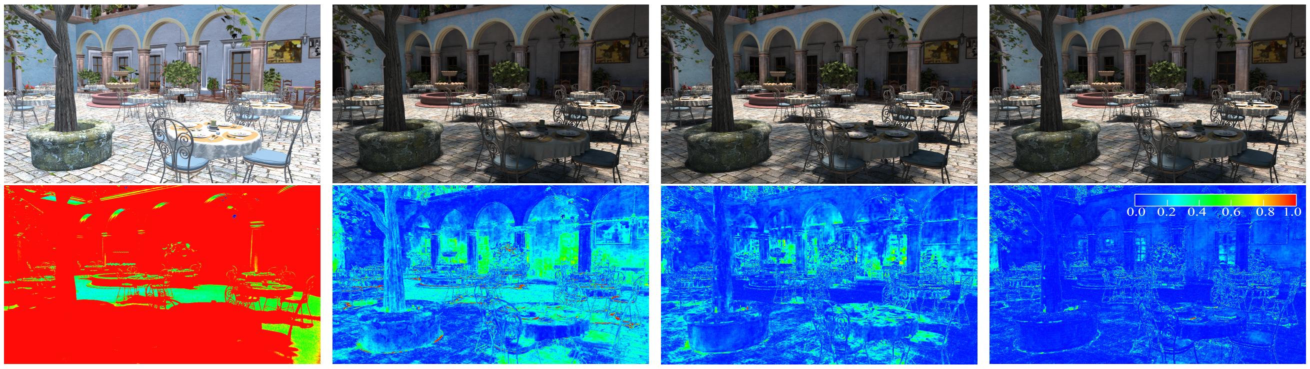
The comparison includes equal samples and equal quality. Fig. 10 shows the visual comparison of rendering results generated by different methods using equal samples. Note that in multidimensional lightcut and the lightslice, the sample number is the cut size, but in our method, the sample number is the number of accurately computed entries of visibility. As can be seen, with equal samples, these previous methods produce results with many artifacts and the relative error is very high. In contrast, our method produces high quality results with much less errors. We generate equal quality images by setting parameters to generate images with similar average relative error. Visual comparison can be found in the supplementary document. Compared with the multidimensional lightcut, our method is more than 7-20 times faster in equal quality. Compared with the lightslice method, our method also has 3 to 7 times speedup in equal quality.

8 Discussion and Limitation

8.1 Sampled and Recovered Matrices

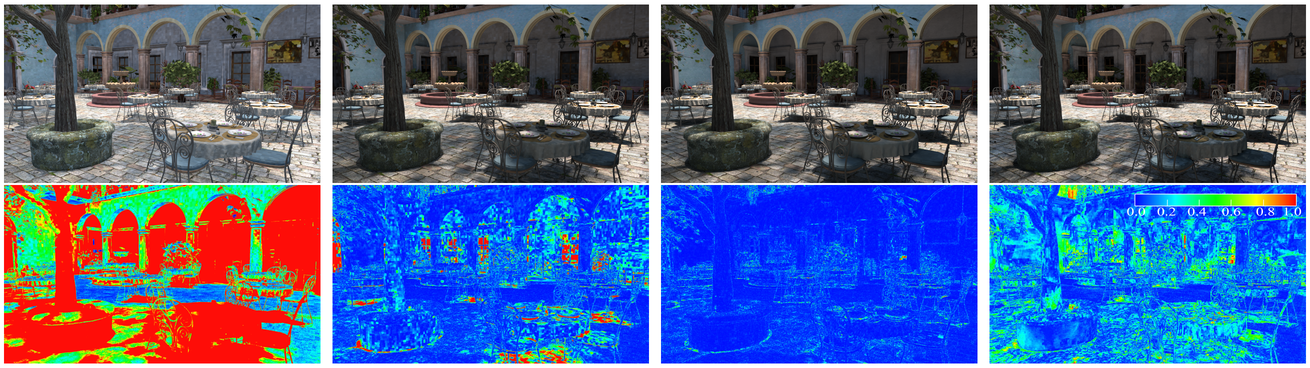
In Fig. 7, we show an example of sampled and recovered matrices. Intermediate matrices of two slices are visualized in red and blue boxes. These two slices are with different illumination and visibility. The one marked in red is without much occlusions, thereby has low-rank; but the one in blue crosses the shadow region with more complex visibility, which has relatively higher rank. In dashed boxes, we give the accurately sample true lighting matrix as reference, and its distribution of eigenvalues. As can be seen, the prediction matrix gives a relatively good approximation of true matrix. After the matrix separation, even for the matrix with complex visibility, errors in the error matrix is sparsely distributed. Error density map is computed from the error matrix, where detected dense error regions (entries in red) are probably high-rank, and are actually sampled. For the slice with simple occlusions, approximately no entry needs further samples, but for the slice with complex occlusion, about 10% entries are re-sampled. In our test senses, the average re-sampling rate over all slices is approximately $1\% - 2\%$. Comparing the difference between the reference and the final recovered matrix, it can be seen that our sample-and-recovery method well captures the main illumination of these two slices.

Currently, the separation step is limited to low-rank lighting matrix. Even though there is a high-rank error correction technique to detect the dense high-rank part from the error matrix $\mathbf{Z}$, some sparse high-rank parts (if any) will be missed. This limitation may bring problems for rendering highly specular or highly glossy objects.



(a) *Without prediction* (b) *Without separation* (c) *Without error correction* (d) *Prediction + Separation*

Figure 8: *Results generated by partial steps of our algorithm for comparison.*



(a) *Average 10 rays/pixel* (b) *Average 25 rays/pixel* (c) *Average 46 rays/pixel* (a) 50% *columns, 46 rays/pixel*

Figure 9: *Results generated by different numbers of samples.*

8.2 Steps in Matrix Prediction and Separation

In Fig. 8, we compare intermediate results generated by partial steps of our entire matrix recover algorithm. Fig. 8(a) shows a result directly recovered from a matrix only with some sparse visibility samples but without any predictions. As can be seen, many occlusions are missed and incorrectly recovered. This demonstrates that our matrix prediction well explores the locality of visibility and effectively approximates the distribution of the visibility. Fig. 8(b) shows a result directly generated after the matrix prediction. Without the matrix separation step, the failures of predictions directly reflect on the final image, e.g. the blocky artifacts at shadow boundaries and the reds regions in the error image. Fig. 8(c) shows a result after matrix separation but without applying the error detection and correction. It can be seen that the separation successfully separates some errors produced by prediction and reduce the overall errors. However, there are still some errors from high-rank parts remaining in the error matrix. Finally, by employing all steps in matrix prediction and separation (Fig. 8(d)), these high-rank parts in error matrix is resampled to further improve the final result. More results of this comparison on other test scenes can be found in the supplementary document. Results show that these steps are all necessary for producing high quality results.

8.3 Sampling Rate

The quality of our matrix recovery heavily depends on how many samples are used to actually compute the visibility. In three different steps of our algorithm, each generates a portion of samples to actually compute visibility. In constructing reduced lighting matrix, for each light, approximately 1% entries of each column are sampled. In matrix prediction, several iterations of prediction and validation are carried out until the prediction error ratio is less than the prediction error ratio, e_p. In each iteration, ρ (1%) entries of each column are sampled. In the final step, only a very small part, less than 1% of entries, are actually resampled in our practice. In Fig. 9, we compare several results generated by different sampling by tweaking the splitting threshold α in constructing lighting matrix, and the prediction error ratio, e_p, in matrix prediction. As can been seen in Fig. 9(a) with less samples, our method fails to capture shadows as well as produces blocky artifacts between slices. While increasing the sample rate, more and more shadows are better preserved, Fig. 9(b). Only some heavily occluded regions can not be finely recovered. Once samples are generated enough to capture the occlusions, in Fig. 9(c), our matrix recovery produces less errors. In Fig. 9(d), we show a result using a large α, only 50% columns comparing that in Fig. 9(c) are split to construct the lighting matrix. Currently, the number of columns or the samples used in prediction depend on some predefined parameters. Although these predefined values work well in our test scenes, they may not be applicable in some scenes. This is one limitation of our method.

9 Conclusion and Future Work

We have presented a matrix sampling-and-recovery scheme to efficiently compute the lighting matrix in scalable many-lights rendering. It outperforms the previous methods in terms of efficiency. The proposed method can be further improved by designing better algorithm for finding the similar VPLs, better predictors for the lighting matrix, and robust separation solver for handling larger matrix.

One interesting future direction is to extend our solution to some high-rank lighting matrix, e.g. scenes with highly specular or glossy materials. Another future work is to extend our sparse sampling-and-recovery idea to deal with other sub-paths, e.g. by sampling and recovering the contributions of camera rays. Additionally, one future work is to investigate its behavior in dealing with temporal coherent image sequences. Finally, it is very interesting to explore the connection between our method and the Monte Carlo rendering techniques.

Acknowledgement

We would like to thank all reviewers for their thoughtful comments. This work was partially supported by NSFC (No. 61472350), 863 program of China (No. 2012AA011902), 973 Program of China (No. 2015CB352503) and the Fundamental Research Funds for the Central Universities (No. 2015XZZX005-05).

References

BOYD, S., AND VANDENBERGHE, L. 2004. *Convex optimization*. Cambridge university press.

CANDÈS, E., AND RECHT, B. 2009. Exact matrix completion via convex optimization. *Found. Comput. Math. 9*, 6 (Dec.), 717–772.

CANDÈS, E. J., LI, X., MA, Y., AND WRIGHT, J. 2011. Robust principal component analysis? *Journal of the ACM (JACM) 58*, 3, 11.

DACHSBACHER, C., KŘIVÁNEK, J., HAŠAN, M., ARBREE, A., WALTER, B., AND NOVÁK, J. 2014. Scalable realistic rendering with many-light methods. In *Computer Graphics Forum*, vol. 33, Wiley Online Library, 88–104.

DAVIDOVIČ, T., KŘIVÁNEK, J., HAŠAN, M., SLUSALLEK, P., AND BALA, K. 2010. Combining global and local virtual lights for detailed glossy illumination. *ACM Trans. Graph. 29* (December), 143:1–143:8.

GEORGIEV, I., KŘIVÁNEK, J., POPOV, S., AND SLUSALLEK, P. 2012. Importance caching for complex illumination. In *Computer Graphics Forum*, vol. 31, Wiley Online Library, 701–710.

HALKO, N., MARTINSSON, P.-G., AND TROPP, J. A. 2011. Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions. *SIAM review 53*, 2, 217–288.

HAŠAN, M., PELLACINI, F., AND BALA, K. 2007. Matrix row-column sampling for the many-light problem. *ACM Trans. Graph. 26*, 3, 26:1–10.

HAŠAN, M., KŘIVÁNEK, J., WALTER, B., AND BALA, K. 2009. Virtual spherical lights for many-light rendering of glossy scenes. *ACM Trans. Graph. 28* (December), 143:1–143:6.

HUANG, F.-C., AND RAMAMOORTHI, R. 2010. Sparsely precomputing the light transport matrix for real-time rendering. *21st Eurographics Symposium on Rendering 29*, 4 (June).

KAUTZ, J., AND MCCOOL, M. D. 1999. Interactive rendering with arbitrary brdfs using separable approximations. In *Rendering Techniques*, 247–260.

KELLER, A. 1997. Instant radiosity. In *Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, SIGGRAPH '97, 49–56.

LIAO, S., LOPEZ, M. A., AND LEUTENEGGER, S. T. 2001. High dimensional similarity search with space filling curves. In *Data Engineering, 2001. Proceedings. 17th International Conference on*, IEEE, 615–622.

LIN, Z., CHEN, M., AND MA, Y. 2009. The augmented lagrange multiplier method for exact recovery of corrupted low-rank matrices. *UIUC Technical Report UILU-ENG-09-2215*.

MCCOOL, M. D., ANG, J., AND AHMAD, A. 2001. Homomorphic factorization of brdfs for high-performance rendering. In *SIGGRAPH*, 171–178.

MOHRI, M., AND TALWALKAR, A. 2011. Can matrix coherence be efficiently and accurately estimated? In *International Conference on Artificial Intelligence and Statistics*, 534–542.

OU, J., AND PELLACINI, F. 2011. Lightslice: matrix slice sampling for the many-lights problem. *ACM Trans. Graph. 30*, 6 (Dec.), 179:1–179:8.

RITSCHEL, T., GROSCH, T., KIM, M. H., SEIDEL, H.-P., DACHSBACHER, C., AND KAUTZ, J. 2008. Imperfect shadow maps for efficient computation of indirect illumination. *ACM Trans. Graph. 27*, 5, 129:1–8.

SEGOVIA, B., IEHL, J. C., MITANCHEY, R., AND PÉROCHE, B. 2006. Bidirectional instant radiosity. In *Proceedings of the 17th Eurographics Conference on Rendering Techniques*, Eurographics Association, Aire-la-Ville, Switzerland, Switzerland, EGSR '06, 389–397.

SEGOVIA, B., IEHL, J., AND PEROCHE, B. 2007. Metropolis instant radiosity. *Computer Graphics Forum 26*, 3, 425–434.

SHEN, Y., WEN, Z., AND ZHANG, Y. 2014. Augmented lagrangian alternating direction method for matrix separation based on low-rank factorization. *Optimization Methods and Software 29*, 2, 239–263.

SLOAN, P.-P., HALL, J., HART, J., AND SNYDER, J. 2003. Clustered principal components for precomputed radiance transfer. *ACM Trans. Graph. 22*, 3 (July), 382–391.

WALTER, B., FERNANDEZ, S., ARBREE, A., BALA, K., DONIKIAN, M., AND GREENBERG, D. P. 2005. Lightcuts: a scalable approach to illumination. *ACM Trans. Graph. 24*, 3, 1098–1107.

WALTER, B., ARBREE, A., BALA, K., AND GREENBERG, D. P. 2006. Multidimensional lightcuts. In *ACM Transactions on Graphics (TOG)*, vol. 25, ACM, 1081–1088.

WALTER, B., KHUNGURN, P., AND BALA, K. 2012. Bidirectional lightcuts. *ACM Trans. Graph. 31*, 4 (July), 59:1–59:11.

WANG, R., HUO, Y., YUAN, Y., ZHOU, K., HUA, W., AND BAO, H. 2013. Gpu-based out-of-core many-lights rendering. *ACM Trans. Graph. 32*, 6 (Nov.), 210:1–210:10.

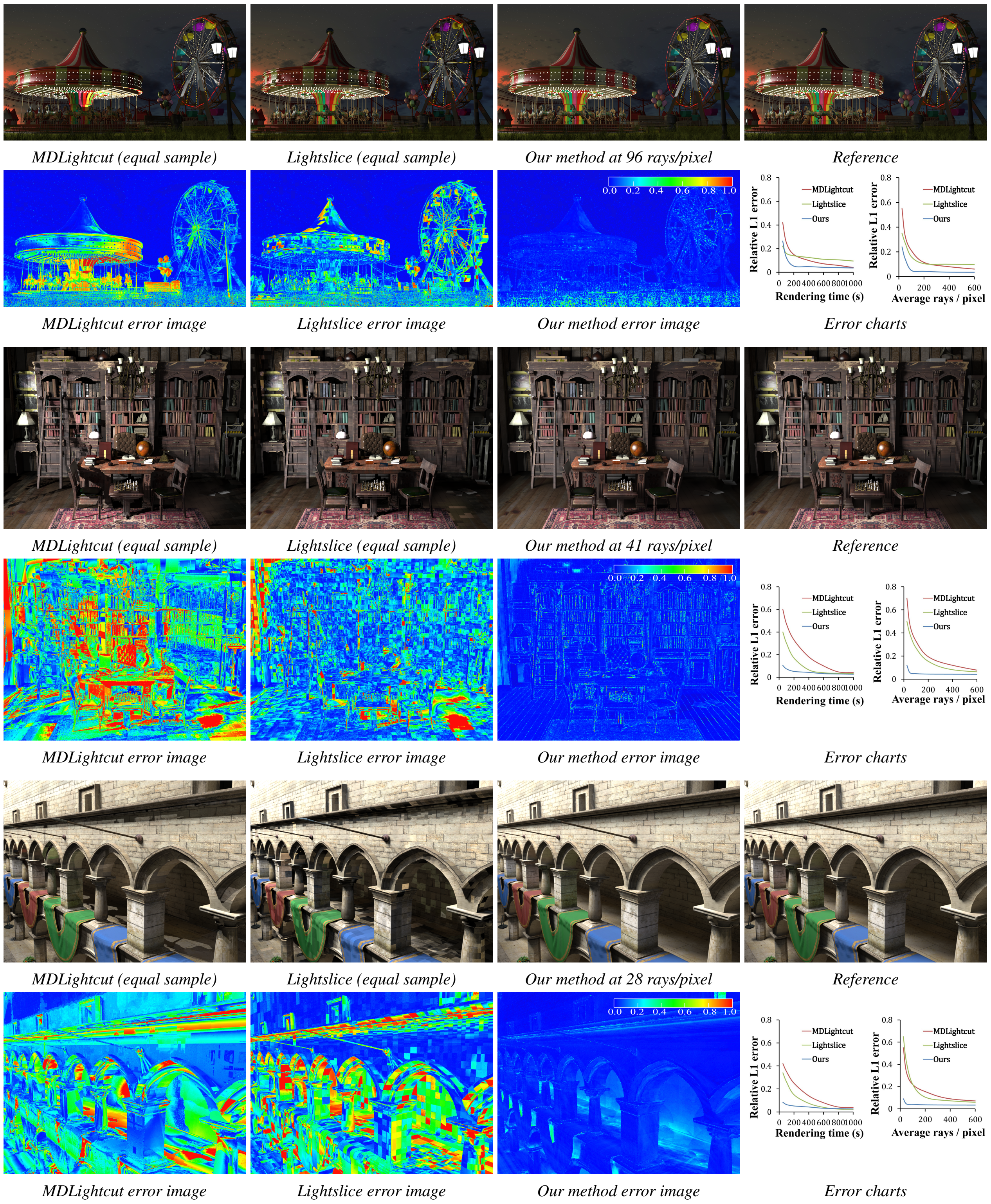


Figure 10: *Comparison of different methods with equal samples.*