Adaptive Matrix Column Sampling and Completion for Rendering Participating Media

Yuchi Huo  Rui Wang∗  Tianlei Hu  Wei Hua  Hujun Bao∗
State Key Lab of CAD&CG, Zhejiang University

Abstract

Several scalable many-light rendering methods have been proposed recently for the efficient computation of global illumination. However, gathering contributions of virtual lights in participating media remains an inefficient and time-consuming task. In this paper, we present a novel sparse sampling and reconstruction method to accelerate the gathering step of the many-light rendering for participating media. Our technique explores the observation that the scattered lightings are usually locally coherent and of low rank even in heterogeneous media. In particular, we first introduce a matrix formulation with light segments as columns and eye ray segments as rows, and formulate the gathering step into a matrix sampling and reconstruction problem. We then propose an adaptive matrix column sampling and completion algorithm to efficiently reconstruct the matrix by only sampling a small number of elements. Experimental results show that our approach greatly improves the performance, and obtains up to one order of magnitude speedup compared with other state-of-the-art methods of many-light rendering for participating media.

Keywords: participating media, many-light rendering, matrix completion, adaptive rendering

1 Introduction

Participating media, such as clouds, fogs, liquid, and transparent solid, significantly contribute to a wide variety of vivid visual effects. Different kinds of rendering methods have been proposed and developed based on the radiative transfer equation (RTE) [Chandrasekhar 1960] to simulate such effects. However, one of the greatest challenges to solve this equation is the tremendous amount of possible light transport and scattering among particles, which inspires a series of solutions [Lafortune and Willems 1996; Georgiev et al. 2013; Jarosz et al. 2011b; Novák et al. 2012a; Frederickx et al. 2012b; Novák et al. 2012a; Novák et al. 2012b; Frederickx et al. 2015].

Many-light rendering is one class of methods having attracted increasing attention and achieved significant progress. These methods simplify the rendering into two steps, namely, virtual light shooting and virtual light gathering, as well as utilize the direct illuminations of the virtual lights to approximate multiple bounces of light reflection and scattering [Keller 1997; Raab et al. 2008; Engelhardt et al. 2012; Novák et al. 2012b]. Different types of virtual lights have been proposed and utilized to simulate different scattered lightings, such as points, rays, and beams. However, even when state-of-the-art representations, such as virtual ray lights (VRLs) or virtual beam lights (VBLs), and the most recent scalable approach [Frederickx et al. 2015] are used, gathering the contributions from a large number of virtual lights remains a tedious and time-consuming task.

In this paper, we introduce a new sparse sampling and reconstruction scheme for rendering participating media. Our method is based on the observation that the scattered lightings are usually locally coherent and of low rank even in heterogeneous participating media. By formulating the gathering step in participating media as a matrix sampling and reconstruction problem, we exploit the local coherence and low rank properties of the matrix, and apply the latest low-rank matrix completion technique [Krishnamurthy and Singh 2014] to accelerate the gathering from a small number of samples. Specifically, we first partition the VRLs and eye rays into segments and pair them to construct the lighting matrix. The matrix columns correspond to the segments of VRLs, and the rows correspond to the segments of eye rays. A novel adaptive sampling and reconstruction approach is then proposed to recover the matrix by sparsely sampling and completing columns. The results show that our method significantly improves the performance of rendering participating media, and obtains up to one order of magnitude speedup compared with state-of-the-art methods of many-light rendering for participating media.

The main contributions of this work are as follows:

• a new many-light rendering formation based on the segments of VRLs and eye rays for rendering participating media;
• a new graph-partitioning algorithm to group and refine VRL and eye ray segments;
• a new adaptive matrix column sampling and completion method to recover the lighting matrix from a small number of elements; and
• an efficient rendering algorithm that heavily accelerates state-of-the-art methods and is capable of rendering a large amount of VRLs.

2 Related Work and Background

Many-light Rendering for Participating Media Numerous methods have been proposed to study the rendering of participating media. We recommend a survey [Pegoraro 2009] for a comprehensive introduction. In this paper, we focus only on relevant works, i.e., the many-light rendering approaches for participating media.

Many-light rendering originates from instant radiosity (IR) [Keller 1997], a special case of bidirectional path tracing, in which photons are first traced from light sources and stored as virtual point lights (VPLs), and indirect illuminations are gathered and approximated by these direct illuminations of VPLs. The reuse of light sub-paths of the IR method exhibits better performance. However, as the number of VPLs increases to thousands, millions, or more, efficiently gathering their contributions becomes a challenge, and motivates a series of scalable algorithms that are known as many-light rendering. Dachsbacher et al. [2014] provided a comprehensive survey on this class of methods.

Lightcuts [Walter et al. 2005] is a pioneer scalable solution that uses tree cuts from light tree to gather contributions from a large number of VPLs. Bidirectional Lightcuts [Walter et al. 2012] uses multiple bounce shading points to handle a high range of glossy material. Georgiev et al. [2012] sampled VPLs with sophisticated PDF to capture complex illumination. Matrix row and column sampling [Hašan et al. 2007] formulates the gathering of many lights as a problem of filling a lighting matrix. LightSlice [Ou and Pellacini 2011] clusters the rows of the lighting matrix into local slices and selects different representative VPLs to better capture subtle local lightings. Recently, a new matrix sampling-and-recovery method [Huo et al. 2015] has been proposed to recover the visibility of sub-lighting matrices with sparsely sampled matrix elements. Our work is inspired by this method, but solves a more challenging problem, i.e., rendering participating media.

Multidimensional Lightcuts [Walter et al. 2006] extends the idea of Lightcuts to high dimensional light trees; thus, this method can gather the scattered illuminations with a massive amount of VPLs in the media. However, the point-based representation suffers from either serious singularity artifacts or energy loss because of clamping. Raab et al. [2008] and Engelhardt et al. [2012] combined the traditional point-based IR with local path tracing to compensate the lost clamping energy. Nevertheless, the full compensation reverts to brute-force path tracing. Lately, new methods [Jarosz et al. 2011a; Jarosz et al. 2011b; Krivánek et al. 2014; Novák et al. 2012b; Novák et al. 2012a] extend the representation of scattering illumination from points to rays or beams, which significantly diminish the singularity artifacts. To represent virtual light, our method uses the VRL [Novák et al. 2012b] rather than VBL [Novák et al. 2012a] because the progressive scheme of the latter is more difficult to integrate in our matrix formation and sampling, although VBLs can better suppress the singularity artifacts than VRLs can.

Ray or beam-based approaches are more efficient in gathering virtual lights than point-based approaches. However, they still require tremendous sampling and integrations to render complex scenes with a large number, e.g., millions, of virtual lights. Fredericks et al. [2015] adapted LightSlice [Ou and Pellacini 2011] to accelerate the gathering in such a case. Different from their work, our method proposes a new matrix formation and introduces a new matrix sampling and reconstruction technique that completes the matrix only from a small number of matrix elements.

Adaptive Sampling The strategy of adaptive sampling has been widely used in rendering, especially in reducing the noise of Monte Carlo path tracing. Following the classical work [Mitchell 1987] in this field, adaptive sampling has achieved significant advances in recent years. Comprehensive reviews were conducted by Sen et al. [2015] and Zwicker et al. [2015]. In many-light rendering, numerous methods employ the idea of adaptivity. Lightcuts can be regarded as a generalized adaptive sampling algorithm [Walter et al. 2005; Walter et al. 2006; Walter et al. 2012; Fredericks et al. 2015], where it builds tree hierarchies of light sub-paths and adaptively connects them with eye sub-paths while considering a lower-error bound metric. In this paper, our method also possesses adaptivity by sampling matrix columns and completing the sampled columns adaptively.

Matrix Completion Matrix completion has recently become a popular topic in the research community. Theoretically, matrix completion aims to recover a low-rank matrix \( L \) from a set of sparsely sampled matrix elements by minimizing the matrix rank [Candès and Recht 2009]. The matrix rank of \( L \) is numerically approximated by its nuclear norm, defined as the sum of its singular values. Instead of completing the entire matrix at once [Candès and Recht 2009], Krishnamurthy et al. [2014] proposed an adaptive matrix completion method that recovers the matrix columns one by one by projecting each column into a low-rank sub-space of the matrix. We adopt this method in our approach because it column-wisely completes the matrix, so that we only need to sample and complete a small part of columns and then estimate the integration of all columns.

Matrix sampling and recovery techniques have also been used in recent rendering works. Huang et al. [2010] sparsely sampled and recovered light transport for precomputed rendering. Matrix row and sampling [Hašan et al. 2007] and its following work, LightSlice [Ou and Pellacini 2011], samples a subset of columns and rows of the lighting matrix to fill up the matrix. Wang et al. [2009] proposed a kernel Nyström method to reconstruct the lighting transport matrix from a few images. Peers et al. [2009] adopted compressive sensing techniques to reconstruct light transport for relighting. Ren et al. [2013] proposed to precompute radiance regression functions for real-time global illumination. Huo et al. [2015] sparsely sampled and recovered the visibilities of the lighting matrix. These methods accelerate the rendering of certain scenes but are unable to handle participating media.

3 Overview

In this section, we first present our matrix formation of rendering participating media and then provide an overview of our adaptive matrix column sampling and completion algorithm.

3.1 Rendering Participating Media with Virtual Lights

In our work, we follow the mathematical formation presented by Dachsbacher et al. [2014]. Theoretically, to render the participating media, the outgoing radiance at shading point \( x \) toward direction \( -\omega \) can be computed by gathering illuminations from VPLs as:

\[
L_o(x, -\omega) = \sum_j f_j(x) G(x, y_j) V(x, y_j),
\]  

(1)
where \( y_j \) and \( I_j \) are the position and intensity of \( j \)-th VPL, respectively; \( f(x), G(x, y) \) and \( V(x, y) \) are the generalized scattering distribution function (BRDF of the surface point or product of the phase function and scattering coefficient of media), generalized geometry term, and generalized visibility (product of normal visibility and transmittance), respectively. The detailed definitions of these functions can be found in the work of Dachsbacher et al. [2014].

To diminish the singularity produced by VPLs, Novák et al. [2012b] proposed to use eye rays and VRLs to replace shading points and VPLs, as well as integrated contributions between two rays as:

\[
L_m \approx \int_0^{x_m'} \int_0^{t_m'} \tau(a, x_m) I_j \tau(b, y_m) f(x_m) f(y_m) G(x_m, y_m) V(x_m, y_m) dt ds, 
\]

where \( x_m = a + t \omega \) defines the eye ray starting from \( a \) to direction \( \omega \); \( y_m = b + s \psi \) defines the light ray starting from \( b \) to direction \( \psi \); \( \tau(\cdot) \) is the transmittance between two points; \( f(x_m) \) and \( f(y_m) \) are the generalized scattering distribution functions starting from \( x_m \) toward \( y_m \) and vice versa; and \( I_j \) is the intensity of \( j \)-th VRL.

### 3.2 Matrix Formation

Inspired by previous many-light rendering approaches [Hašan et al. 2007; Ou and Pellacini 2011; Hao et al. 2015; Frederickx et al. 2015], we formulate the rendering of participating media as the sampling and integrating of a lighting matrix. In particular, we split the eye rays and VRLs in the media into segments, and use the light transport between light and eye ray segments as the matrix element. By introducing these segments, we can formulate the entire rendering of the scene with participating media into four types of matrices, with two types of virtual lights as matrix columns, i.e., VRL segments and VPLs, as well as two types of receivers as matrix rows, i.e., shading points and eye ray segments. Examples of these matrices are illustrated in Figure 2. The matrix element in \( i \)-th row and \( j \)-th column is the contribution from \( j \)-th virtual light to \( i \)-th receiver. If the virtual light is a VRL segment and the receiver is an eye ray segment, then the matrix element of the segment-to-segment contribution is computed as:

\[
E_{ij}^{ss} = \int_{s_j}^{s_i} \int_{t_j}^{t_i} \tau(a_i, x_m) I_j \tau(b_j, y_m) \int f(x_m) f(y_m) G(x_m, y_m) V(x_m, y_m) dt ds, 
\]

where \( x_m = a_i + t \omega \) and \( y_m = b_j + y \psi \) are the segments of VRL and eye ray, respectively; \( a_i \) is the origin of eye ray segment \( i \); and \( b_j \) is the origin of VRL segment \( j \).

Similarly, the other three types of matrix elements can be computed as:

\[
E_{ij}^{ps} = \int_{t_j}^{t_i} \tau(a_i, x_m) I_j f(x_m) G(x_m, y) V(x_m, y) dt, 
\]

\[
E_{ij}^{pp} = \int_{s_j}^{s_i} I_j \tau(b_j, y_m) f(x) G(x, y_m) V(x, y_m) ds, 
\]

\[
E_{ij}^{pp} = I_j f(x) G(x, y_j) V(x, y_j), 
\]

where \( E_{ij}^{ps}, E_{ij}^{pp} \) and \( E_{ij}^{pp} \) are the point-to-segment, segment-to-point and point-to-point contributions from \( j \)-th virtual light to \( i \)-th receiver, respectively; \( y_j \) and \( x_i \) are the VPL and surface shading point, respectively.

### 3.3 Algorithm Overview

To calculate the final illumination of every row (receiver) of a sub-lighting matrix, we need to integrate all columns as follows:

\[
s = \sum_j I_j, 
\]

where \( I_j \) denotes the \( j \)-th column of the sub-lighting matrix and the rows of \( s \) are the illuminations of the receivers. For a large scene with thousands or millions of virtual lights, accurately sampling and integrating every element of the lighting matrix are time-consuming and impractical. Therefore, we present a new adaptive matrix column sampling and completion algorithm to efficiently compute the final illuminations.

Based on the lighting matrix formations, three types of lighting matrices are composed of segments in rows, columns, or both. We use the matrix of eye ray and VRL segments as an example and show an overview of our algorithm in Figure 3. For other types of lighting matrices, these steps are similar but use points instead of segments. Our algorithm starts with the eye rays and VRLs generated by previous approaches [Novák et al. 2012b]. The algorithm then takes two steps, namely, matrix construction (Section 4) and adaptive column sampling and completion (Section 5). The basic idea is to first use the locality and coherence of the scattered lighting in media by slicing the entire lighting matrix on rows and building a hierarchy of columns, and then adaptively sample and complete columns in sliced sub-matrices. We take different strategies to explore the coherence of rows and columns. Compared with the number of eye rays, the number of virtual lights may vary dramatically in different scenes. Therefore, our algorithm slices the lighting matrix at first but postpones the selection of columns at the sampling step rather than at the matrix construction step to be scalable to these lights.

### Matrix Construction

Eye rays and VRLs are first divided into segments, which are clustered into groups and refined locally. We propose a similarity metric on segments and a graph-based partitioning algorithm to perform the segment clustering. Then, these clustered eye ray segments are used to slice the entire light matrix into sub-matrices, and the clustered VRL segments are used to construct a light tree. These sub-matrices with hierarchical columns are processed individually in the following step.
Adaptive Column Sampling and Completion. We adaptively sample only a few columns instead of all columns, and these columns are used to estimate the entire column integration. Furthermore, we subsequently employ an adaptive matrix completion technique that recovers these sampled columns by estimating only a few elements of rows. In summary, our scheme samples a subset of the matrix columns and rows, thereby remarkably reducing the sampling cost.

4 Matrix Construction

In this section, we describe the algorithm to construct the lighting matrix. For simplicity, we focus on the construction of the segment-to-column lighting matrix. Other types of lighting matrices can be constructed similarly if their rows or columns are ray segments.

Theoretically, the lower the rank of the matrix is, the fewer samples are required to complete it [Candès and Recht 2009; Krishnamurthy and Singh 2014]. Therefore, the goal of matrix construction is to obtain a lighting matrix with data coherence and low rank. Inspired by LightSlice [Ou and Pellacini 2011], we group the eye ray segments into clusters and slice the entire lighting matrix into submatrices. Furthermore, inspired by Lightcuts [Walter et al. 2005], we hierarchically group VRL segments into a light tree. These two operations on eye ray and VRL segments require a similarity metric to measure the difference in segments as well as a clustering algorithm to group similar ones. In the following, we first introduce the generation of an initial set of segments, then present a graph-based partitioning algorithm to group these segments, and finally present the step to refine the segments.

4.1 Generating Initial Segments

Once the rays are obtained, we first generate a set of initial ray segments by distributing \( h \) points along each ray to partition it into \( h + 1 \) segments, where \( h = 16 \) is set as default. We use Woodcock tracking to generate samples proportional to the transmittance to spread more samples in dense medium regions, which have more visual contributions to the image. A 2D VRL case is shown in Figure 4(a). However, even considering the importance of medium properties to generate points, these randomly distributed points may still fail to capture the local coherence of lighting transport among rays. Therefore, these segments are only regarded as an initial set of segments and will be refined later.

4.2 Graph-based Partitioning Algorithm

Given the initial set of segments, we propose a graph-based partitioning algorithm to group them into similar clusters and refine these ray segments using cluster boundaries. We first introduce the graph representation, then describe the partition algorithm, and finally present the step to refine the segments.

4.2.1 Graph Representation

We organize the segments into a graph, where the graph node is the phase function at two points of \( x_i \) and \( x_j \), which are the angles in the world space that produce maximum transmittance variations, especially in heterogeneous media, of two segments as:

\[
S(o_i, o_j) = \gamma(\delta_i - \delta_j) + \omega_i (\omega_j - \omega_i - \| x_i - x_j \|) + \omega_j (\omega_i - \omega_j - \| x_i - x_j \|),
\]

where \( \omega_i \) and \( \omega_j \) are the similarities of eye ray and VRL segments, respectively; \( \delta_i \) is the spatial distance term; \( \gamma \) is the directional scattering difference between two segments; \( \omega_i \) is 1/8 is a weight to normalize the spatial distance. The directional term \( \delta \) depicts the directional scattering difference between two segments as:

\[
\delta(x_i, x_j) = \max(p_{\omega_i}(\theta_{\omega_i}^{\omega_{\omega_i}}) - p_{\omega_j}(\theta_{\omega_j}^{\omega_{\omega_j}}), p_{\omega_j}(\theta_{\omega_j}^{\omega_{\omega_j}}) - p_{\omega_i}(\theta_{\omega_i}^{\omega_{\omega_i}})),
\]

where \( \kappa = 0.5 \) is a weight to normalize the term; \( p_{\omega_i} \) and \( p_{\omega_j} \) are the phase functions at two points of \( x_i \) and \( x_j \), respectively; \( \theta_{\omega_i}^{\omega_{\omega_i}} \) and \( \theta_{\omega_j}^{\omega_{\omega_j}} \) are the angles in the world space that produce maximum phase function values at \( x_i \) and \( x_j \), respectively; and \( \max(\cdot) \) returns the larger difference between these two phase functions. We use the maximum value difference of these two phase functions to depict the difference between them, because the computation is much cheaper. \( \tau \) distinguishes the energy attenuations caused by transmittance variations, especially in heterogeneous media, of two segments as:

\[
\tau(x_i, x_j) = 1 - \| \tau(x_i, x_j)\|,
\]

of which the values can be directly computed from the cached transmittance along VRLs. The pixel filtering term is a Gaussian filter specifically defined for eye rays as:

\[
P = 1 - \frac{1}{2\pi\sigma} e^{-\frac{x^2+y^2}{2\sigma^2}},
\]
where $\bar{x}$ and $\bar{y}$ are the horizontal and vertical image distances between $o_i$ and $o_j$, respectively; and $\sigma$ is a parameter set to 4. We define this term in the image space to differentiate eye ray segments with close start points from the camera.

Ideally, for each pair of segments, an edge with similarity exists. However, considering that the spatial distance of two segments is large, the similarity distance becomes dominated by the spatial distance. Therefore, in practice, we consider only the spatial neighbors of segments. Each segment is linked to its nearest five spatial neighbors by searching the nearest start or end points of segments in a point KD-tree. Figure 4(b) illustrates a VRL segment graph, in which graph nodes are in orange dot and edges are in black lines.

### 4.2.2 Graph Partitioning

Given a graph, we convert the clustering problem into a graph-partitioning problem to partition graph nodes (segments) into groups. These groups are then used to slice the lighting matrix and build the VRL hierarchy, which is discussed in Section 4.2.3. We use the K-cut graph-partitioning algorithm to partition the graph into K groups while minimizing the sum of the edge weights that connect nodes in different groups. The accurate K-cut graph partitioning is a NP-hard problem, but with several efficient approximation algorithms.

We adopt the mtMetis proposed by LaSalle et al. [2013]. Figure 4(c) illustrates an example, in which the nodes are partitioned into two groups that are denoted as red and white. We select the graph-partitioning algorithm because this approach efficiently preserves segments of the same rays; therefore, the segments of different clusters are not interlaced with other rays. The K-means clustering may fail to preserve the spatial connection because it directly encompasses spatial information into a high-dimensional distance metric, in which spatial connections might be suppressed by other factors. We present further comparisons in Section 6.1.

### 4.2.3 Segment Refinement

The initial segments are randomly generated. Consequently, they may not effectively capture the local coherence of the media. We observe that these segment clusters, after the graph partitioning, expose global information on how segments should be distributed in the media. For example, the boundary among clusters reflects certain changes in the media. Therefore, these segments across the boundary should be divided into two segments, so that such changes among clusters can be better preserved and in-cluster coherence can be improved. We propose a three-step refinement procedure to utilize the underlying information of segment clusters and improve the segments along rays, i.e., find better start or end points of segments.

The first step is to determine the boundary between two neighboring clusters. We adopt the support vector machine (SVM) to find an optimal decision hyperplane between two clusters. We use only the start or end points of boundary segments as the training data in the SVM to simplify the computation. The radial basis function $K(x, y) = e^{-\frac{(x-y)^2}{\sigma^2}}$ with $\gamma = 0.5$ is used as a nonlinear kernel to compute the support vector, where $x$ and $y$ are positions of the points. The nonlinear kernel enables us to compute the non-planar boundary between clusters. Given the boundary, we then split the segments across the boundary and create new segments. Finally, each newly created segment is compared with the connecting segment in the same cluster using the similarity metric (Eq.(8) or (9)). They are merged if they are similar enough, i.e., the similarity metric is larger than a threshold of 2.5.

### 4.3 Slicing Lighting Matrix and Building Light Tree

For VPLs or shading points, we use the methods proposed in previous studies [Ou and Pellacini 2011; Frederickx et al. 2015; Huo et al. 2015] to cluster rows and build a hierarchy of lights.

For eye rays, we directly partition them into $K$ clusters. $K$ is empirically set to 0.008 times the total number of eye rays. With this setting, each group contains approximately 500 to 1000 segments (rows) after the partition. Such a setting experimentally suits our matrix completion algorithm in terms of efficiency. Upon obtaining the clusters of eye ray segments, we use them to slice the lighting matrix into sub-matrices.

For VRLs, we iteratively apply two-cut partitions on the VRL segments until all segments in the same node reach the fixed similarity threshold. At each iteration, a graph is partitioned into two subgraphs. These partitions result in a hierarchy of subgraphs, which directly maps to a virtual light tree.

### 5 Adaptive Sampling Algorithm

With sliced sub-lighting matrices and virtual light trees, the next step is to compute the final illumination of every receiver (point or segment) by sampling and integrating columns of each sub-lighting matrix. In this section, we introduce our sampling and reconstruction algorithm that adaptively samples columns and completes the sampled columns from a small number of elements.

The pseudo code of the entire algorithm is shown in Algorithm 1. The overall procedure of our algorithm exhibits certain similarities to the tree traversal scheme proposed by Lightcuts [Walter et al. 2005]. In particular, we hierarchically traverse the light tree to compute a cut. All tree nodes on the cut are stored in a priority queue, $Queue$, and sorted by errors. At each traversal, one node with the largest error is popped and split into children nodes, where each child node is sampled and evaluated in the $SampleAndEvaluate$ routine. Once the error of this child node is less than a threshold, we accumulate its contribution to receivers and stop the traversal on this node; otherwise, this child node is pushed into the queue.

However, our method fundamentally differs from the standard Lightcuts procedure in several aspects. First, in Lightcuts, a representative light is used to estimate the overall contribution of a
node; in our method, constructing a representative light segment for segments with varying lengths, intensities and scattering coefficients is impractical. Thus, we propose a column sampling method to sample a subset of columns in the node and estimate the overall contribution from them. Second, while sampling the column, our method only uses a small number of rows to complete the entire column instead of fully sampling all rows. Finally, instead of using a conservative error bound, our method computes the error from a number of columns at a tree node because of the difficulty in estimating an error bound on VRSLs. In the following subsections, we first describe the procedures to sample and complete columns and then introduce the step to evaluate the node error.

5.1 Computing Node Illumination

A tree node on the cut contains a set of light segments. Given a tree node and receivers represented in columns and rows of a sub-matrix, integrating of node illumination aims to compute all illuminations from light segments of the node to all rows. Instead of densely sampling all elements, we adopt the idea of adaptive matrix column completion [Krishnamurthy and Singh 2014] that utilizes the sub-column space to reconstruct columns by sampling only a few elements of rows. However, even when the sub-column space is used, still many columns have to be sampled reman. Instead, inspired by the idea proposed by Georgiev et al. [2010] to sample VPLs, we use the Monte Carlo integration that only randomly samples a small number of columns. In the following, we first introduce the column completion step and then describe the Monte Carlo integration step.

5.1.1 Column Completion

An adaptive matrix completion algorithm [Krishnamurthy and Singh 2014] is adopted for our column completion. Technically, we maintain a sub-column space $U$ of the matrix to complete columns. We use $L \in \mathbb{R}^{m \times k}$ to denote a sub-lighting matrix and $U \in \mathbb{R}^{m \times k}$ to denote the sub-column space of $L$, where $k \ll n$. Each column of $U$ is an orthogonal basis of the sub-column space. When sampling a new column, $l$, we first sample a few elements of this new column, where $\Omega \in \mathbb{R}^{[\rho m]}$ denotes uniformly sampled indices ranging from 0 to $m$, and $l_0 \in \mathbb{R}^{[\rho m]}$ is the down-sampled version of $l$, which contains only sampled rows, i.e. the rows indexed by $\Omega$. Then, the entire column $l$ can be reconstructed by projecting the down-sampled column back into the column space $U$ as:

$$l \approx \bar{l} = U(U_0^T U_0)^{-1} U_0^T l_0$$

Figure 5: Illustration of one column recovery.

The pseudo code of this column completion algorithm is provided in Algorithm 2, and an illustration of the recovery of one column is shown in Figure 5. The column completion algorithm draws a small set of row indices from the routine DrawRowIndices and places them in $\Omega$. Rows of unknown columns are sampled according to the indices in $\Omega$ in the routine DrawRowSamples. In line 6, we test whether the down-sampled column $l_0$ is in the down-sampled space $U_0$. If this down-sampled column $l_0$ is in the space, then it is directly projected back into the full column space $U$ (line 10). Otherwise, the entire column $l$ is sampled and added to the column space $U$.

We use $E_0(l_0)$ to represent the operation that projects a down-sampled column $l_0$ into the space $U$ as $E_0(l_0) = (U_0^T U_0)^{-1} U_0^T l_0$, and $e \in \mathbb{R}^k = E_0(l_0)$ to denote the projection of $l_0$ on the sub-column space $U_0$. Such a projection $e$ can be used to recover the entire column $l$ from the down-sampled column $l_0$ as $l \approx \bar{l} = U e$, where $\bar{l}$ is the approximation of the original column $l$. Using these projections and recoveries, we can efficiently perform the integration and error estimation in the sub-column space $U$, in which the column size of sub-column space is much smaller than the original column space, $k \ll n$, and the dimension of the projection $e$ is much smaller than the row size, $pm \ll m$.

5.1.2 Integrating Columns

Based on the sub-column space reconstruction, the integration of the entire sub-lighting matrix is the summation of all columns.
\[ \sum_j U \approx U \sum_j e^j. \] By combining the idea of Monte Carlo sampling in VPLs [Georgiev and Slusallek 2010], we can compute the integration of the node as:

\[ i(n) \approx U e(n) = U \frac{1}{K} \sum_{k=1}^{K} E_{v}(U \cdot t^k_{1(k)}), \quad (14) \]

where \( K \) is the number of sampled columns in the node, \( PDF(U \cdot t^k_{1(k)}) \) is the sampling probability of column \( k \), \( t^k_{1(1)} \) is the randomly sampled column, \( E_{v}(U \cdot t^k_{1(1)}) \) calculates the projection of column \( t^k \) in space \( U \), and \( e(n) \) is the integration of projections. We simply use intensities of virtual lights as the PDF and sample 10 columns per node as default. More details can be found in the supplementary document. By taking Eq.(14) into Eq.(7), we compute the final illumination of the entire sub-lighting matrix by summing all contributions of nodes as:

\[ s = \sum_j U e(n) = U \sum_j e(n), \quad (15) \]

where, we first accumulate the projections in the sub-column space and then project them back to the full column space, which further reduce the computation of integration.

### 5.2 Estimating Node Errors

Two kinds of errors are used in our algorithm to control the procedures of sampling and reconstruction. The first error is the completion error in line 6 of Algorithm 2. This error represents the energy loss in the column completion. If one column \( I \) cannot be well reconstructed by sub-column space \( \hat{U} \), then this column will be fully sampled. Such an error is computed as:

\[ \frac{\|E_v - U \cdot e\|_1}{\|t_1\|_1} \|e\|_1 > \sigma \|e_{all}\|_1, \quad (16) \]

where \( e_{all} \) is the accumulated contribution of all nodes in the current tree cut; \( \sigma \) is a predefined ratio parameter, with 0.05 as default; and \( \frac{\|E_v - U \cdot e\|_1}{\|t_1\|_1} \) estimates the percentage of energy loss for \( I \).

The second error is introduced to evaluate how well the node contribution is estimated by some of its columns (lines 9 and 14 of Algorithm 1). We use an error metric similar to the adaptive clustering metric proposed by Hachisuka et al. [2008]. Specifically, the column sampling error \( r(n) \) for node is computed as:

\[ r(n) = \frac{1}{N} \sum_{n} (e(n)) - \frac{E_v(U \cdot t^k_{1(k)})}{PDF(U \cdot t^k_{1(k)})} \right)^2. \quad (17) \]

The average squared difference between sampled columns and the estimated node contribution in space \( U \) is calculated. If \( r(n) \) exceeds \( \beta e_{all} \), then the estimate of this node is not good enough, and the node is split. We set \( \beta = 0.01 \) as default.

### 6 Results

We implement our method on a PC with two Intel Xeon E5-2630 CPUs and 32GB memory. The matrix computation is conducted using the Intel MKL library. Different virtual lights, including directional, omni and diffuse oriented VPLs [Walter et al. 2005] and standard VRLs [Novák et al. 2012b], are used in test scenes. We compare the quality and performance of our method with those of the original VRL [Novák et al. 2012b] and the recent adaptive LightSlice [Frederickx et al. 2015]. All algorithms are parallelized in our test machine, but the reference images are rendered by a cluster with 32 PCs in tens of hours. The statistics of demo scenes are listed in Table 2. The column sampling ratio is manually set as 0.1 in most cases. The maximum memory consumptions of our method, adaptive LightSlice and VRL are 470, 280 and 200 MB, respectively. Compared with previous methods, the additional memory used in our method is for storing the hierarchies of VPLs and segments. Unless mentioned specifically, we use the same original method of VRL [Novák et al. 2012b] to compute the segment-to-segment illuminations.

#### 6.1 Comparison of Different Partitions of Rays

![Comparison of different partitions of rays. The eye rays are partitioned by three different strategies and rendered with equal sampling ratios. The reference image of this scene is shown in Figure 7(d).](image)

Figure 6 shows the results generated with three ray-partitioning strategies, namely, the proposed graph-based partitioning, K-Means clustering, and no partitioning on rays. The previous approaches [Frederickx et al. 2015] can be regarded as using no partitioning strategy. We visualize the ranks of sub-lighting matrices in Figure 6 (bottom). A lower rank indicates fewer samples to sample and complete these matrices. In Figure 6 (top), we render the scene with an equal sampling ratio. As can be seen, our graph-based partitioning obviously outperforms the other two ray partitioning strategies.

#### 6.2 Results of Column Completion

We show several results that are only produced by column completion without adaptive column sampling (i.e., we fully sample every column but we only complete each column by a few row elements) to better evaluate the completion step in our algorithm. We first show the effect of the error threshold \( \sigma \) on the quality of the column completion algorithm in Figure 7. Large \( \sigma \) results in some loss of lighting details. In our practice, \( \sigma = 0.005 \) is good enough to recover subtle lighting effects; therefore, it is used as default in all our experiments.

We then use a studio scene to show the effect of the sampling ratio \( \rho \) on the quality of the column completion algorithm. In this scene, the major light source comes from the window and directly illuminates the smoke, in which the major illumination of the scene is contributed by indirect VRLs. We forcibly sample every column of the sub-lighting matrix, and test various sampling ratio \( \rho \) of rows. As shown in Figure 9, with only 10% of elements per column, the resultant image is close to the reference. After counting the overhead to conduct the column completion, our column completion achieves over 7 times speedup in generating an image with a similar quality. While the column sampling ratio decreases, blocky
6.3 Results of Anisotropic Materials

We apply our algorithm to render participating media with anisotropic materials, and show the results in Figure 8. From left to right, the Buddha models have different coefficients $g$ of the Henyey-Greenstein (HG) phase function, ranging from purely isotropic 0 to highly anisotropic 0.99. At the top row of Figure 8, the ranks of sub-lighting matrices are presented, in which the rank increases as the anisotropic coefficient increases. We use two sampling ratios, 0.1 and 0.2, to render the scene. As shown in the middle and bottom rows of Figure 8, when the material becomes increasingly anisotropic, the lower sampling ratio tends to produce more noticeable artifacts because of the lack of samples. The increase in rank of the lighting matrix leads to the increase of sampling ratio. Therefore, if the scene is complex both in lighting and material, our algorithm might degrade to the traditional full sampling algorithm.

6.4 Final Results and Comparison

We compare the overall performance of our method with those of VRLs [Novák et al. 2012b] and adaptive LightSlice [Frederickx et al. 2015] by rendering various scenes at equal time. Error images show the relative $L^1$ absolute errors. Specifically, $e = \frac{\|L_o - L\|_1}{\|L_o\|_1}$, where $L_o$ is the intensity of the reference image, and $L$ is the intensity of the test image. We also plot the curves of time versus number of virtual lights for different methods to better compare the scalability of different methods.

The city scene in Figure 1 is the largest scene in our paper. The entire city is covered by heterogeneous smoke and lit by sky lighting from the sun and several street lamps. The sun and the lamps produce light shafts in the smoke. The complex effects require a large number of virtual lights to capture the lighting. At equal time, the VRL method generates serious artifacts because of only a small number of VRLs that can be processed during a limited time. For the same reason, the adaptive LightSlice has some obvious blocky artifacts, especially in the regions lit by the lamps. By contrast, our algorithm works well in this scene because our method samples 10% of the elements per column. In this way, our method can process more columns in the same time.

The Sponza scene in Figure 10 is a small but commonly used scene to test the complex indirect lighting transport in the corridor. At the same processing time, only a small number of virtual lights coming into the yard are captured by the VRL method. As a result, this method produces a large light shaft and spiky lighting artifacts. The adaptive LightSlice captures most lighting features of the scene but still has blocky artifacts as shown in the error map. Our method generates a smoother result at the same processing time. The curves on the bottom right of Figure 10 show the scalabilities of the three methods under the same error threshold. For one millions virtual lights, our method outperforms the adaptive LightSlice over one order of magnitude because of our sparse sampling and completion scheme.

Finally, we test the three methods on a laboratory scene with various participating media, as shown in Figure 11. From left to right, the media are soft drink (Gatorade), jade, soap, juice (grapefruit), skin and milk with different HG phase functions. These material parameters are from previous studies [Gkioulekas et al. 2013b; Jakob 2010; Gkioulekas et al. 2013a] and listed in Table 1. These media comprise anisotropic media (such as jade, beverage and soap) and
Our method, however, has several limitations. First, the sampling ratio increases with the rank of the lighting matrix. Therefore, our method might need a higher sampling ratio to render scenes with a high frequency of light scatterings. Second, we cannot automatically detect the best sampling ratio for different scenes or different regions of the scene. Adaptively estimating the optimal sampling ratios for different matrices is a promising future direction, but it remains a challenge in the research of matrix completion. Third, similar to many other many-light methods, our method does not explicitly support temporal coherence. Preserving temporal coherence involves a series of works that are beyond the scope of our paper, such as ways to distribute virtual point lights to diminish flickering [Dachsacher et al. 2014]. Addressing this limitation is a highly challenging and interesting future task to explore the potential of our method in time domain.

The proposed algorithm is a general framework for other effects that can be formulated as matrix integration problems, including extending this method to a higher dimension, such as motion blur, depth of field, or temporal coherent animation. Applying the adaptive matrix column sampling and completion or the sparse sampling concept to other rendering methods, such as bidirectional volumetric path tracing, and even problems outside rendering, is worth exploring in future.

Acknowledgement

We would like to thank Kavita Bala for the initial discussion and all reviewers for their thoughtful comments. We also would like to thank Shihao Jin and Haohua Liu for preparing demos. This work was partially supported by National Key R&D Program of China (No. 2016YFB1001503), NSFC (No. 61472350), the Fundamental Research Funds for the Central Universities (No. 2016FZA5013).

Table 1: Media parameters in the Lab scene.

<table>
<thead>
<tr>
<th>Media</th>
<th>$\sigma_a$</th>
<th>$\sigma_m$</th>
<th>$\rho$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Drink</td>
<td>0.04,0.04,0.06</td>
<td>0.02,0.02,0.009</td>
<td>0.93,0.93,0.94</td>
</tr>
<tr>
<td>Jade</td>
<td>1.1,1</td>
<td>0.002,0.004,0.007</td>
<td>0.9,0.9,0.9</td>
</tr>
<tr>
<td>Soap</td>
<td>0.2,0.2,0.22</td>
<td>0.001,0.001,0.002</td>
<td>0.96,0.95,0.94</td>
</tr>
<tr>
<td>Juice</td>
<td>0.15,0.15,0.16</td>
<td>0.03,0.17,0.48</td>
<td>0.93,0.93,0.93</td>
</tr>
<tr>
<td>Skin</td>
<td>0.74,0.88,1</td>
<td>0.03,0.17,0.48</td>
<td>0.0,0</td>
</tr>
<tr>
<td>Milk</td>
<td>1,1</td>
<td>0.0026,0.005,0.01</td>
<td>0.82,0.8,0.75</td>
</tr>
</tbody>
</table>

References


Figure 10: Sponza scene

Figure 11: Lab scene


GEORGIEV, I., AND SLUSALLEK, P. 2010. Simple and robust iterative importance sampling of virtual point lights. In In Eurographics short papers.


JAROSZ, W., NOWROUZEZAHRAI, D., SADEGHI, I., AND JENSEN, H. W. 2011. A comprehensive theory of volumetr-
Table 2: The statistics of the scenes and results. The surface points and eye rays are separately counted, on which the column completion time is the pure algebraic computation of matrix completion algorithm. The last two sections are equal-time statistics of the VRL and the adaptive Lightslice.

<table>
<thead>
<tr>
<th>scene</th>
<th>triangles</th>
<th>resolution</th>
<th>VPLs/VRLs</th>
<th>ray partition time(s)</th>
<th>column sample ratio</th>
<th>avg. densely sampled columns</th>
<th>avg. sparsely sampled columns</th>
<th>avg. sampled tree nodes</th>
<th>column completion time(s)</th>
<th>other time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>sponza</td>
<td>2.9K</td>
<td>1600×1200</td>
<td>723K/490K</td>
<td>14</td>
<td>0.1</td>
<td>30</td>
<td>385</td>
<td>31</td>
<td>761</td>
<td>803</td>
</tr>
<tr>
<td>city</td>
<td>153K</td>
<td>1920×1080</td>
<td>3180/342</td>
<td>21</td>
<td>0.1</td>
<td>35</td>
<td>36</td>
<td>342</td>
<td>426</td>
<td>462</td>
</tr>
<tr>
<td>lab</td>
<td>107K</td>
<td>1600×1200</td>
<td>2135/274</td>
<td>18</td>
<td>0.2</td>
<td>42</td>
<td>22</td>
<td>274</td>
<td>255</td>
<td></td>
</tr>
<tr>
<td>studio(ρ = 0.1)</td>
<td>346K</td>
<td>1600×1200</td>
<td>759/436</td>
<td>11</td>
<td>0.1</td>
<td>26</td>
<td>12</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>surface points</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>eye rays</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>equal-time VRL</td>
<td>avg. relative error</td>
<td>total time (s)</td>
<td></td>
<td>27%</td>
<td>1218</td>
<td>1337</td>
<td>840</td>
<td>457</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(no clustering)</td>
<td>VPLs/VRLs</td>
<td></td>
<td>604/1076</td>
<td>927/510</td>
<td>510/225</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>equal-time Lightslice</td>
<td>avg. relative error</td>
<td></td>
<td></td>
<td>14.2%</td>
<td>1098</td>
<td>1107</td>
<td>840</td>
<td>457</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>


