Sparse Sampling and Completion for Light Transport in VPL-based Rendering

Yuchi Huo  Rui Wang  Xinguo Liu  Hujun Bao

Abstract

The many-light formulation provides a general framework for rendering various illumination effects using hundreds of thousands of virtual point lights (VPLs). To efficiently gather the contributions of the VPLs, lightcuts and its extensions cluster the VPLs, which implicitly approximates the lighting matrix with some representative blocks similar to vector quantization. In this paper, we propose a new approximation method based on the previous lightcut method and a low-rank matrix factorization model. As many researchers pointed out, the lighting matrix is low rank, which implies that it can be completed from a small set of known entries.

We first generate a conservative global light cut with bounded error and partition the lighting matrix into slices by the coordinate and normal of the surface points using the method of lightslice. Then we perform two passes of randomly sampling on each matrix slice. In the first pass, uniformly distributed random entries are sampled to coarsen the global light cut, further clustering the similar light for the spatially localized surface points of the slices. In the second pass, more entries are sampled according to the possibility distribution function estimated from the first sampling result. Then each matrix slice is factorized into a product of two smaller low-rank matrices constrained by the sampled entries, which delivers a completion of the lighting matrix. The factorized form provides an additional speedup for adding up the matrix columns which is more GPU friendly. Compared with the previous lightslice method, our approach approximates the lighting matrix with some signal specialized bases via factorization. The experimental results show that we can achieve significant acceleration than the state-of-the-art many-light methods.

1. Introduction

Rendering realistic images with global illumination evolves integrating the rendering equation, which is a recursively defined expression because the incoming lighting is to be determined by the same equation. Various efforts are attributed to high-quality rendering [37, 36, 14, 13, 12, 5, 7, 16, 15, 11], or to achieve a balance between quality, performance, and flexibility [20, 23, 1, 27, 42, 22]. Keller [19] introduced an intermediate representation, called virtual point light (VPL), to break the recursive rendering equation, yielding a fast realistic rendering algorithm and inspiring a number of followups called 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 illumination results of the VPLs. The rendering results can be also expressed as summing the columns of a matrix, called the lighting matrix, where each row corresponds to a pixel and each column corresponds to a light. However, the cost of computing the lighting matrix will become prohibitively expensive when there are hundreds of thousands of VPLs required for faithful approximation of the lighting effects, which is unfortunately very common in high-quality rendering applications.

Recently, many extensions and accelerations of the VPL-based rendering have been developed, among which a class of work, called the lightcut based method, focused on efficiently solutions based on cutting/clustering [34, 33, 10, 26]. The lightcut method [34] showed that, while the number of lights is huge, they can be aggressively clustered as some representatives without degrading the perceptual quality of the result image. The followup work, row-column sampling method [10], finds the same set of representative lights for the whole image for further speedup, and the lightslice method [26] refine the clusters for preserving the details by slicing the image into smaller blocks.

From the view of the lighting matrix, the lightcut based methods approximate the lighting matrix with some vectorized blocks. In this paper, we propose to approximate the light transport matrix by a product of two low-rank factorized blocks. As the lightslice method did, we first partition the matrix into slices and generate a lightcut for each slice. Then, instead of computing all entries, we sparsely sample the corresponding sub matrix for each slice and complete it using a low-rank factorization method designed for sparse matrix [41]. In order to capture the matrix structure with as few as possible samples, we design an importance function the lights and a possibility distribution function for sampling the lighting matrix. As shown in Figure 1, the lighting matrix can be faithfully recovered using 10% sampled entries for rendering the final images. As an byproduct benefit, the low-rank production form allows for faster inte-
gration to get the final image, since we can sum the columns of lighting matrix without explicitly recovering the matrix.

The contributions of this work are:

- Introduce a sparse factorization method for the lighting matrix in many-light problem.
- Propose a high quality lightcut generation algorithm using sparsely sampled lighting and visibility information.
- Propose an efficient, GPU friendly sampling and completion algorithm for the lighting matrix.

2. Related Work

We first review some work on virtual point light, lightcut and some realistic rendering methods related to them, then review some work on matrix factorization.

**Many-light methods** Keller [19] introduced Instant Radiosity (IR) method, which is the first realistic rendering algorithm using virtual point lights. It traces a number of virtual point lights (VPLs) from light sources, and then computes the indirect global illumination in the scene as the direct illumination of the virtual point lights. This idea is very attractive, and has inspired many follow up works, which manipulate the rendering equation in a many-light framework based on VPL. A key issue in many-light framework is the expensive cost dealing with a large number of lights, which is an order of millions and even more in high quality realistic rendering. Walter et al. [34] introduced a lightcut method to reduce the pixel-light computation cost from linear to sub linear by clustering the lights into a few hundreds of representatives. The multidimensional lightcuts method [33] extended lightcut to handle high dimensional integrations for rendering volume scattering, depth of field or motion blur, etc. Hasan et al. [10] proposed a sampling method to generate a global light cut optimized for the whole image, which greatly reduce the cost for cutting the light tree for every pixels. To further exploit the spatial coherence and adapt the image details, Ou et al. [26] partition the image into slices and refine the global lightcut for each slice. Note that visibility test is the most expensive step in computing the light transport entries. The work of Ritschel et al. [28] shows that approximate visibility test with imperfect shadow map can effectively reduce the cost for indirect illumination.

An limitation of VPL based rendering lies in difficulties for handling glossy materials. In order to handle glossy materials and subsurface scattering, Walter et al. [35] introduced a new intermediate representation, called virtual point sensors, to compensate the bias due to VPL clamping. Hasan et al. [9] extended VPL to virtual spherical light (VSL) to avoids the clamping bias. However it may bring blurry artifacts on sharp features. Davidovic et al. [6] introduces local VPLs to compensate the loss of clamping and offers better approximation for sharp glossy reflections.

**Path tracing methods** Since the pioneer work of Whitted [40], path tracing has been standard framework for simulating global illumination. Bidirectional ray tracing combines the light path tracing and eye path tracing to inherit the specific advantages of path tracing as well as light tracing [31]. Monte Carlo path tracing is unbiased, but suffers from low convergence speed and perceptible noise. Photon mapping pioneered by Jensen [17] traces photons in the scene and then caches and reuses them for final gathering of paths from eye. However, photon mapping usually requires a large number of photons to reduce noise and reproduce sharp details, especially for glossy materials and caustics. The memory requirement is proportional to the number of photons, which is thus very high. Progressive photon mapping [8] alleviates this problem by multi-pass processing.

**Cache-based methods** Irradiance caching [39] progressively caches diffuse irradiance samples as an octree, and reuses them along the computation. Radiance caching [21]
extends it by recording directional radiance using spherical harmonics. Bala et al. [3] presented a general approach to exploiting both spatial and temporal coherence of rays for radiance interpolation. Okan Arikan et al. [2] introduced a fast approximation to global illumination by decomposing radiance fields into far- and near-field components, which are computed separately to improve efficiency. For glossy interreflections, Krivanek et al. [32] proposed a spatial directional cache method for glossy to glossy reflections. The GPU accelerated irradiance caching by Wang et al. [38] handles only diffuse interreflections.

**Matrix factorization** Matrix recovery has a wide range of applications in image processing, graphics, vision and machine learning. A recovery of a partial matrix is a specific choice of values for the unspecified entries. Given a few assumptions on the nature of the matrix, e.g. the matrix is low rank and the observed entries are randomly sampled, the missing entries can be recovered by various algorithms [4]. In this paper, we chose an nonnegative factorization method [41] to recover the sparsely sampled lighting matrix. There have been a few applications of matrix factorization in rendering, e.g. separable BRDF[18], environment mapping[25], precomputed radiance transfer[29]. However, none of them touched sparsely sampled matrix with a lot of missing entries.

### 3. Overview

This section gives an overview of the presented rendering algorithm. We adopt the matrix formulation [10, 26] for the many-light problem to describe the sampling and rendering algorithms.

Suppose that there are \( m \) surface points each of which corresponds to a pixel in the rendered image, and \( n \) virtual point lights generated for rendering the scene. Let \( A \) be the lighting matrix, where each entry \( A(i, j) \) represents the illumination contribution from light \( j \) to surface point \( i \). Then the rendered image can be expressed as \( \Sigma_A \), the sum of the columns of matrix \( A \).

Since the number of the virtual point lights is very high, naively computing all of the entries in matrix \( A \) is impractical. The lightcut based methods cluster the virtual point lights into some representative lights, which merges the columns corresponding to the lights in the same cluster, yielding a reduced lighting matrix. In this paper, we present new techniques to improve the clustering procedure, and further save the computation by sparsely sampling the reduced lighting matrix.

Given the VPLs in the scene and the surface points to be shaded, our rendering algorithm proceeds in 5 main steps as follows:

1. **Global Lightcut**: build a light tree for the VPLs, and generate a global cut \( g \) of the light tree.

![Image](image_url)

We adopt the lightcut method in [33] to generate the global lightcut. The global lightcut is error-bounded for the whole image. It is also conservative, because the visibility function is overestimated. In our experiments, the global lightcut contains about 800 ~ 900 nodes of lights.

2. **Matrix Slicing**: partition the matrix into slices by clustering the surface points according to their coordinates and normal.

Considering both efficiency and stability, smaller sized and lower ranked matrices are preferred for low-matrix factorization methods. The original lighting matrix \( A \) usually have several hundreds of thousands of rows, and several hundreds of columns corresponding to the node number in the lightcuts, which makes matrix \( A \) difficult to factorize. Therefore, we need to partition the matrix into small ones. We adopt the lightslice method [26] to partition the matrix, and set the slice size to be about 800 pixels. As many research pointed out, the lighting matrix of nearby pixels has much lower rank [24, 26]. Therefore, it is beneficial to perform low-rank factorization on smaller slices instead of on the whole image.

3. **Light Coarsening**: for each slice, coarsen the global lightcut \( g \) to merge more light node. The surface points in each slice are more closely located, then more lights are likely to be distant and can be merged safely without introducing perceptible artifacts. Therefore, we can cut more nodes in the light tree for each slice, so as to further save computational cost. We design a random sampling method and a cost function for coarsening the global lightcut. The details about light coarsening is presented in Section 4.

4. **Matrix Sampling and Completion**: for each matrix slice, sample a sparse set of entries and perform a low-rank factorization.

Performing low-rank analysis via SVD on the lighting matrix is mentioned by Hassan et al.[10]. However, naive SVD is impractical for efficient rendering, since the lighting matrix takes long time to compute. Instead, we sample a small percentage of entries in the lighting matrix, and then recovery it by performing low-rank factorization with the known entries as constraints. The detail of the sampling and factorization procedure is presented in Section 5.

5. **Image Rendering**: At last, the final image is rendered slice by slice. Consider a matrix slice \( s \), and let \( M \) be the lighting matrix of slice \( S \), and its factorization result be \( M = X \cdot Y \). Then the rendered image restricted on slice \( s \) can be rewritten as follows:

\[
I(s) = M \cdot e = (X \cdot Y) \cdot e = X \cdot (Y \cdot e),
\]
where e is a 1D column vector whose elements are all 1s. Therefore, the recovered matrix slices are not necessarily stored for producing the final image, as long as the factor matrices X and Y are kept.

The methods in step 1 and 2 are adopted from the matrix row-column sampling paper [33] and the lightslice paper [26], where the readers can find the details. Since step 5 is straightforward, we will present the methods for step 3 and 4 in the next two sections.

4. Light Coarsening

Consider two sibling leaf nodes, $L_a$ and $L_b$, in the conservative global lightcut $g$, and let $L_f$ be the parent node of $L_a$ and $L_b$. By the definition of lightcut, we have:

$$I_f = I_a + I_b,$$

where $I_{a/b/f}$ be the intensity of light node $a/b/f$. Without loss of generality, suppose that $L_f$ is the same as $L_a$ except their intensity values.

To determine if light $L_a$ and $L_b$ can be merged for rendering slice $s$, we randomly select some pixels (rows) inside slice $s$, and compute the lighting results by light $L_a$ and $L_b$. Since brighter light usually has more impact on the final image, we set the number of the random pixels to be linearly proportional to the total light intensity, i.e. $I_f$, to take more care on significant lights. Once the sampling number is determined, we choose the pixels uniformly distributed inside the image block corresponding to the current slice.

Let $\zeta_f$ be the set of selected pixels, $V_a$ and $V_b$ be vectors of the lighting results by lights $L_a$ and $L_b$ on pixels $\zeta_f$. Then, the vector of the lighting results by light $L_f$ is $V_f = V_a + V_b$. When we merge lights $L_a$ and $L_b$ to coarsen the light tree, the vector of the lighting result is computed as: $V_fI_f/T_f$ (recall that $L_f$ is the same as $L_a$ except intensity value). Therefore, for pixels in $\zeta_f$, the error of cutting light node $L_a$ and $L_b$ is:

$$\varepsilon(L_f) = \| (V_a + V_b) - V_a'I_a/T_a \|_\infty = \| V_b - V_a'I_a/T_a \|_\infty.$$  

To take into account the accumulation error, we define the following cost function for cut the light tree at node $L_f$:

$$\text{cost}(L_f) = \varepsilon(L_f) + \text{cost}(L_b),$$  

and we define the cost function as 0 for the leaf nodes in the global light tree $g$.

Now, we can repeatedly cut two sibling leaf nodes as long as the cost function is less than a prespecified error bound. When proceed with this coarsening procedure, the following case must happen: the two sibling leaf nodes in testing are the parent nodes of two previous coarsening operations. In this case, instead of set $\zeta_f$ as randomly selected pixels, we set $\zeta_f$ to be the following union:

$$\zeta_f = \zeta_a \cup \zeta_b.$$  

The union method in above has two benefits. First, it allows for reusing the lighting results computed in the previous coarsening operations. Second, it gives the light at upper level more sampling, which is naturally consistent with our idea to set the number of sampled pixels to be proportional to the intensity of the merged light. Though the sampling number is doubled when the coarsening operation proceeds to the upper level, the total number is bounded, since the upper level has only about half number of nodes than the next level.

We can also perform the coarsening procedure until a specified number lights is reached. For this purpose, we order the pairs of sibling nodes by their cost functions, then proceed the coarsening from the pair with the least cost.

**Coarsening v.s. Refinement** In contrast, the lightslice method [26] generate the per slice lightcut by refining a coarse global lightcut. It is not easy to tell which one is more efficient than the other one, because they have different sampling strategy for evaluating the cost.

Since our cost function is defined recursively for taking into account the accumulation error, it is not suitable for refinement. But there is no obviously benefit of refinement v.s. coarsening, even if we can drop the accumulation error term. Therefore, we preferred coarsening rather than refinement in this work. However, it is worth to point out, when the initial lightcut to start with is sufficiently close to the optimal lightcut, it would be better to support lightcut optimization in both directions by refinement and coarsening, such that we can take advantage of the coherence among the nearby slices to save computation. We leave this point as a future work.

5. Matrix Sampling and Completion

After generating the lightcuts, we have actually sparsely sampled the lighting matrices for all slices in the meanwhile. There are two choice to complete the light matrices. One choice is to directly compute them, as the lightslice method did [26]. The other one is to complete them from the sparsely sampled values, since they have been proven to be low rank matrices [24]. We take the second choice.

However, our experiments showed that the random samples computed in the light coarsening step is too sparse to faithfully recovery the lighting matrix. Therefore, we need sample more entries for completion.

Again, consider a slice $s$ with $m'$ pixels and $n'$ representative lights. Let $M \in \mathbb{R}^{m' \times n'}$ be the corresponding lighting matrix. Before sampling the lighting matrix $M$, we need a possibility distribution function (pdf) for determine the indices of the sampling entries. A naive one is
uniform distribution. But we would like to design better pdf for sampling, since we have already obtained a very sparse observation of the lighting matrix in the previous step for light coarsening. To capture the lighting matrix as much as possible with prescribed number of samplings, we prefer to sample important entries. Though it is not clear what kinds of entries are important, the entries corresponding to important light and important pixel are likely to be important by intuition.

Let \( f(i) \) be the importance function of pixel \( i \) and \( g(j) \) be the importance function of light \( j \), then we define the pdf on the lighting matrix as follows:

\[
pdf(i, j) \propto f(i)g(j).
\]

A light can be regarded as important if it produces various lighting results over all pixels, neither consistent dark nor consistent bright. Therefore, we can measure its importance by the variance of its lighting results. Let \( C_j \) be the observed column vector corresponding to light \( j \) in the lighting matrix. Then, we define the following importance function for light \( j \):

\[
g(j) = \max(C_j) - \min(C_j).
\]

For pixel importance function, we simply treat all pixel equally, i.e. define \( f(i) = 1 \) for all pixels, because no useful information is found from the observed lighting matrix for now.

Therefore, to sample a new matrix element, we first select the index for the light (or column) by the light importance function, then select the index for the pixel (or row). If the element of the sampled indices is empty, then compute the lighting results of the corresponding light-pixel pair. The sampling procedure is repeated until a prescribed number of elements is reached.

To complete the lighting matrix with the sparsely sampled entries, we choose a nonnegative matrix factorization method [41] based on the algorithm of alternating direction method of multipliers. A brief introduction of the iteration procedure can be found in the appendix. This factorization method gives high-quality solutions within a few iterations. In experiments, we stop the iteration when the error is below a specified tolerance, or the iteration exceeds a specified number. We set the maximum iteration as 100.

After factorization, we have a completed lighting matrix for slice \( s \):

\[
M = X \cdot Y,
\]

where \( X \in R^{m' \times q} \) and \( Y \in R^{q \times n'} \) are both low rank (\( \leq q \)) factor matrices. \( q \) is user specified parameter as a rank estimation for the input matrix. Smaller \( q \) is faster for iteration, but may lead to large factorization error, and vice versa. We experimented with several values \( q = 8, 16 \) and 32, and found that \( q = 16 \) delivers satisfactory results regarding both efficiency and accuracy. We implemented the factorization method in graphics hardware using Nvidia’s Cuda BLAS to accelerate the matrix computation of the iterations.

6. Implementation and Experimental Results

We have implemented the proposed algorithms, and accelerated them with graphics hardware using Nvidia’s CUDA sdk. In step 1, the global lightcut is generated by splitting the light tree, and the splitting test is parallelized for all nodes level by level in GPU. In step 2, the surface points are represented by 6D vectors (coordinates and normal), and are recursively split into a binary tree. We parallelize the computation on the tree nodes in GPU. In step 3, the coarsening operation is implemented in GPU by parallelizing the computation for all pairs of leaf nodes. In step 4, the matrix factorized is implemented with Cuda BLAS. The final rendering in step 5 is done GPU too. The presented algorithms frequently require shooting rays to compute the pixel-light visibility. We implement a ray-casting method in GPU accelerated by SBVH[30]. To compare with the previous lightcut based methods, we integrate the CPU implementation code by the authors of the lightslice work [26] (which is public available from http://www.cs.dartmouth.edu/fabio/publication.php?id=lightslice11) into our rendering system, and accelerate the visibility computation procedure using our SBVH implementation in GPU.

In the following of this section, we will present some rendering results on 3 testing scenes, and compare our algorithms with the state of the art lightcut based methods. The reference images of the testing scenes are shown in Figure 2. The statistics and timings of our experiments are summarized in Table 1.

| Scene   | Bedroom | TeaRoom | Museum |
|---------|---------|---------|--------|
| #Tris   | 88K     | 3115K   | 1604K  |
| #VPLs   | 50K     | 231K    | 165K   |
| m       | 1024 × 768 | 1024 × 768 | 800 × 600 |
| #Nodes  | 569     | 345     | 609    |
| Time(s) | 8       | 9       | 18     |

Table 1. Summary of the scene size and the rendering performance. #Tris is the number of triangles in the scenes, #Nodes is the averaged number of the leaf nodes in the coarsened light tree. Time is measured in seconds.

Figure 1 shows some results by changing the sampling rate in the step of matrix sampling and completion. The results shows that 10% sampling rate is sufficient to deliver faithfully rendering results. And extremely low sampling rate, 1%, can deliver a reasonable good result for preview. These experiments verify that the lighting matrix is low rank, and can be sparse sampled and completed.
Figure 2. Some testing scenes. The images are rendered by brute-force rendering using all of the VPLs, and will be used as reference for later evaluation.

Figure 3. The Museum scene rendered by our method.

Figure 3 shows a large complex scene with heavy occlusions. For this scene, our rendering takes about 3GB GPU memory. Figure 4 shows some rendering results of a bedroom under a smooth lighting environment. It shows the our method can produce the best result using the smallest number of sampling rays per pixel. The smooth transition of the soft shadows and shadings are well reconstructed. One the contrary, the side wood of the bed exhibits perceptible band artifacts in the images rendered by the lightslice method. The reference image of this scene is shown Figure 2.

Figure 5 compares some rendering results of complex indoor scene illuminated by 24 area light sources. In this scene, there are many casted shadows on the wall, and there are many regions receiving very strong local lightings. With bounded error control, the lightcut method can well preserve edge of the hard shadow (shown in the middle Figure 5(a)). Our method can reconstruct correct light effects and complex shadows using only 10% sampling rate. The soft shadow boundaries shown in the middle Figure 5(c) is due to the low-rank approximation. Figure 5(b) shows that the lightslice method may make mistakes on the shadows because insufficient number of rows are sampled when refine the lightcut for the slices with complex strong local lighting.

7. Conclusion and Discussion

We have presented an efficient rendering algorithm for the many-light problem based upon the idea of lightcut. The presented algorithm is featured by sparse sampling and two novel techniques based on sparse sampling. One technique is the coarsening method for generating lightcuts by sparsely sampling the lighting matrix, and the other is the completion algorithm for the lighting matrix with a few sparsely sampled entries. Thanks to the sparse sampling strategy, the presented algorithm achieves significant acceleration compared with the state of the art many-light rendering algorithms.

Compared with the original lightcut method [34], the success of our coarsening method owns to using the sparsely sampled visibility information. Since the original lightcut overestimated the visibility, it wastes many node splitting budget on the occluded nodes, leading to insufficient splitting budget on the visible nodes and more rendering errors. Our methods avoid visibility overestimation by computing sparse visibility.

The row-column sampling method [10] also generates a global light cut using the visibility information. It randomly selects a set of pixels and computes the lighting result from each lights. Since it uses the full visibility information regarding all lights, the number of selected pixels is limited to several hundreds for speedup. The lightslice method [26] partitions the image space into slices and refines the global light cut for each slice. However, only 1 pixel per slice is considered for the refinement. Our method is similar to the lightslice method in using slices, but ours computes the visibility information of a set of pixels in each slice, which can better capture the lighting characteristics.

The major difference between our method and the previous lightcut based methods lies in the matrix sampling and
Figure 4. The Bedroom scene rendered by various methods. The bottom row shows the difference images multiplied by 8.

Figure 5. The Tearoom scene rendered by various methods. The bottom row shows the difference images multiplied by 8.
completion step. Instead of computing the full lighting matrix for rendering, ours sparsely samples it and reconstructs it by matrix factorization.

One limitation of our sparse sampling and factorization is the lack of coherence, leading to flicking artifacts when moving the light and/or the camera. One possible direction to address this issue is to take advantage of the idea in multidimensional lightcut [33]. As another future work, it is interesting to exploit some image space features to guide the sampling for the lighting matrix factorization, while the pixels are uniformly random sampled in our current implementation.

Acknowledgements

Appendix A – Nonnegative Matrix Factorization

Let $M \in \mathbb{R}^{m \times n}$ be a nonnegative matrix, and $\Omega$ be a set of entry indices at which the value of matrix $M$ is known. The nonnegative factorization of $M$ is defined as minimizing the following Frobenius norm restricted on the known entries:

$$\arg \min_{X,Y} \| P_\Omega (XY - M) \|_F^2,$$

where $X \in \mathbb{R}^{m \times q}$ and $Y \in \mathbb{R}^{n \times q}$, and $X, Y > 0$. To use the alternating direction method (ADM) [41], the above form is converted into the following equivalent form:

$$\arg \min_{U,V,X,Y,Z} \frac{1}{2} \| XY - Z \|_F^2,$$

$$\text{s.t.} \left\{ \begin{array}{l}
X = U, Y = V,
U \geq 0, V \geq 0,
P_\Omega (Z - M) = 0
\end{array} \right\}$$

Then, applying ADM for the augmented Lagrangian in above, we have the following successive minimization steps:

$$X_{k+1} = (Z_k Y_k^T + \alpha U_k - \Lambda_k) (Y_k Y_k^T + \alpha I)^{-1},$$

$$Y_{k+1} = (X_{k+1}^T X_{k+1} + \beta I)^{-1} (X_{k+1}^T Z_k^T + \beta V_k - \Pi_k),$$

$$Z_{k+1} = X_{k+1} Y_{k+1} + P_\Omega (M - X_{k+1} Y_{k+1}),$$

$$U_{k+1} = P_\Omega (X_{k+1} Y_{k+1} + \Lambda_k / \alpha),$$

$$V_{k+1} = P_\Omega (X_{k+1} Y_{k+1} + \Pi_k / \beta),$$

$$\Lambda_{k+1} = \Lambda_k + \gamma \alpha (X_{k+1} - U_{k+1}),$$

$$\Pi_{k+1} = \Pi_k + \gamma \beta (Y_{k+1} - V_{k+1}),$$

where $\Lambda \in \mathbb{R}^{m \times q}$ and $\Pi \in \mathbb{R}^{n \times q}$ are Lagrangian multipliers, $\alpha > 0$ and $\beta > 0$ are penalty parameters, $\gamma \in (0, 1.618)$ is another parameter, operator $P_\Omega$ set the entries at $\Omega$ to be 0s, and operator $P_+$ clamps the negative entries by 0.

References

[1] G. An, Y. Huo, and S.-E. Yoon. Hypergraph propagation and community selection for objects retrieval.

[2] O. Arikan, D. A. Forsyth, and J. F. O’Brien. Fast and detailed approximate global illumination by irradiance decomposition. ACM Trans. Graph., 24(3):1108–1114, 2005.

[3] K. Bala, J. Dorsey, and S. Teller. Radiance interpolants for accelerated bounded-error ray tracing. ACM Trans. Graph., 18(3):213–256, 1999.

[4] E. Candès and B. Recht. Exact matrix completion via convex optimization. Found. Comput. Math., 9(6):717–772, Dec. 2009.

[5] I.-Y. Cho, Y. Huo, and S.-E. Yoon. Weakly-supervised contrastive learning in path manifold for monte carlo image reconstruction. ACM Transactions on Graphics (TOG), 40(4):38–1, 2021.

[6] T. Davidovič, J. Krivánek, M. Hašan, P. Slusallek, and K. Bala. Combining global and local virtual lights for detailed glossy illumination. ACM Trans. Graph., 29:143:1–143:8, December 2010.

[7] H. Fan, R. Wang, Y. Huo, and H. Bao. Real-time monte carlo denoising with weight sharing kernel prediction network. In Computer Graphics Forum, volume 40, pages 15–27. Wiley Online Library, 2021.

[8] T. Hachisuka, S. Ogaki, and H. W. Jensen. Progressive photon mapping. In SIGGRAPH Asia ’08: ACM SIGGRAPH Asia 2008 papers, pages 1–8, New York, NY, USA, 2008. ACM.

[9] M. Hašan, J. Krivánek, B. Walter, and K. Bala. Virtual spherical lights for many-light rendering of glossy scenes. ACM Trans. Graph., 28:143:1–143:6, December 2009.

[10] M. Hašan, F. Pellacini, and K. Bala. Matrix row-column sampling for the many-light problem. ACM Trans. Graph., 26(3):26:1–10, 2007.

[11] Y. Huo. Extension–adaptive sampling with implicit radiance field. arXiv preprint arXiv:2202.00855, 2022.

[12] Y. Huo, S. Jin, T. Liu, W. Hua, R. Wang, and H. Bao. Spherical gaussian-based lightcuts for glossy interreflections. In Computer Graphics Forum, volume 39, pages 192–203. Wiley Online Library, 2020.

[13] Y. Huo, R. Wang, T. Hu, W. Hua, and H. Bao. Adaptive matrix column sampling and completion for rendering participating media. ACM Transactions on Graphics (TOG), 35(6):1–11, 2016.

[14] Y. Huo, R. Wang, S. Jin, X. Liu, and H. Bao. A matrix sampling-and-recovery approach for many-lights rendering. ACM Transactions on Graphics (TOG), 34(6):1–12, 2015.
[15] Y. Huo, R. Wang, R. Zheng, H. Xu, H. Bao, and S.-E. Yoon. Adaptive incident radiance field sampling and reconstruction using deep reinforcement learning. *ACM Transactions on Graphics (TOG)*, 39(1):1–17, 2020.  

[16] Y. Huo and S.-e. Yoon. A survey on deep learning-based monte carlo denoising. *Computational Visual Media*, 7(2):169–185, 2021.  

[17] H. W. Jensen. *Realistic image synthesis using photon mapping*. A. K. Peters, Ltd., 2001.  

[18] J. Kautz and M. D. McCool. Interactive rendering with arbitrary brdfs using separable approximations. In *Rendering Techniques*, pages 247–260, 1999.  

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

[20] S. Kim, Y. Huo, and S.-E. Yoon. Single image reflection removal with physically-based training images. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5164–5173, 2020.  

[21] J. Krivánek and P. Gautron. Radiance caching for efficient global illumination computation. *IEEE Trans. Visualization and Computer Graphics*, 11(5):550–561, 2005.  

[22] S. Li, R. Wang, Y. Huo, W. Zheng, W. Hua, and H. Bao. Automatic band-limited approximation of shaders using mean-variance statistics in clamped domain. In *Computer Graphics Forum*, volume 39, pages 181–192. Wiley Online Library, 2020.  

[23] S. Li, C. Zheng, R. Wang, Y. Huo, W. Zheng, H. Lin, and H. Bao. Multi-resolution terrain rendering using summed-area tables. *Computers & Graphics*, 95:130–140, 2021.  

[24] D. Mahajan, I. K. Shlizerman, R. Ramamoorthi, and P. Belhumeur. A theory of locally low dimensional light transport. *ACM Trans. Graph.*, 26(3), July 2007.  

[25] M. D. McCool, J. Ang, and A. Ahmad. Homomorphic factorization of brdfs for high-performance rendering. In *SIGGRAPH*, pages 171–178, 2001.  

[26] J. Ou and F. Pellacini. Lightslice: matrix slice sampling for the many-lights problem. *ACM Trans. Graph.*, 30(6):179:1–179:8, Dec. 2011.  

[27] H. Park, Y. Huo, and S.-E. Yoon. Meshchain: Secure 3d model and intellectual property management powered by blockchain technology. In *Computer Graphics International Conference*, pages 519–534. Springer, 2021.  

[28] T. Ritschel, T. Grosch, M. H. Kim, H.-P. Seidel, C. Dachsbacher, and J. Kautz. Imperfect shadow maps for efficient computation of indirect illumination. *ACM Trans. Graph.*, 27(5):129:1–8, 2008.  

[29] P.-P. J. Sloan, J. D. Hall, J. C. Hart, and J. Snyder. Clustered principal components for precomputed radiance transfer. *ACM Trans. Graph.*, 22(3):382–391, 2003.  

[30] M. Stich, H. Friedrich, and A. Dietrich. Spatial splits in bounding volume hierarchies. In *High Performance Graphics*, pages 7–13, 2009.  

[31] E. Veach. *Robust monte carlo methods for light transport simulation*. PhD thesis, Stanford, CA, USA, 1998. AAI9837162.  

[32] K. B. V.ˇeclav Gassenbauer, Jaroslav Krivanek. Spatial directional radiance caching. In *Computer Graphics Forum*, page 1189’C1198, 2009.  

[33] B. Walter, A. Arbree, K. Bala, and D. P. Greenberg. Multidimensional lightcuts. *ACM Trans. Graph.*, 25(3):1081–1088, 2006.  

[34] B. Walter, S. Fernandez, A. Arbree, K. Bala, M. Donikian, and D. P. Greenberg. Lightcuts: a scalable approach to illumination. *ACM Trans. Graph.*, 24(3):1098–1107, 2005.  

[35] B. Walter, P. Khungurn, and K. Bala. Bidirectional lightcuts. *ACM Trans. Graph.*, 31(4):59:1–59:11, July 2012.  

[36] R. Wang, Y. Huo, Y. Yuan, K. Zhou, W. Hua, and H. Bao. Implementation details of gpu-based out-of-core many-lights rendering.  

[37] R. Wang, Y. Huo, Y. Yuan, K. Zhou, W. Hua, and H. Bao. Gpu-based out-of-core many-lights rendering. *ACM Transactions on Graphics (TOG)*, 32(6):1–10, 2013.  

[38] R. Wang, R. Wang, K. Zhou, M. Pan, and H. Bao. An efficient gpu-based approach for interactive global illumination. *ACM Trans. Graph.*, 28:91:1–91:8, July 2009.  

[39] G. J. Ward, F. M. Rubinstein, and R. D. Clear. A ray tracing solution for diffuse interreflection. In *Proc. SIGGRAPH ’88*, pages 85–92, 1988.  

[40] T. Whitted. An improved illumination model for shaded display. *Commun. ACM*, 23:343–349, June 1980.  

[41] Y. Xu, W. Yin, Z. Wen, and Y. Zhang. An alternating direction algorithm for matrix completion with non-negative factors. *CoRR*, abs/1103.1168, 2011.
[42] Y. Zhang, R. Wang, Y. Huo, W. Hua, and H. Bao. Power-ernet: Learning-based real-time power-budget rendering. *IEEE Transactions on Visualization and Computer Graphics*, 2021.
