CGI2022 manuscript No.
(will be inserted by the editor)

Fast Probe-Leaking Elimination Using Mask Decomposition

Jixiang Zhou · Yanzhen Chen · Yuanheng Li · Shun Cao · Yu Wu · Xiaogang Jin

Abstract *Light leaking* in Probe GI is typically solved by visibility tests, which cannot benefit from hardware-aided tri-linear sampling. We present *Mask Decomposition*, which decomposes the visibility into probe-group indicators and their corresponding masks, making it possible to use tri-linear sampling in its reconstruction. We prove that the rendering overhead is significantly reduced with the help of Mask Decomposition, making the rendering at least 3× faster than the state-of-the-art visibility-test Probe GI, and even as fast as the original leaking probe GI. We also present an efficient algorithm to solve the Mask Decomposition problem and a simple compression method to minimize the spatial overhead of the mask textures, which is much lower than in compression methods like Moving Basis Decomposition (MBD).

Keywords computer graphics · rendering · global illumination · compression

1 Introduction

Probe GI has suffered from *light leaking* for a long time. Light leaking appears when the geometric occlusion of the scene leads to such great discontinuity of the light field that the light field cannot be well reconstructed with linear interpolation between probes. Light leaking can cause severe rendering artifacts, which impair rendering quality and reduce the authenticity of the virtual space.

In practice, to reduce artifacts caused by light leaking, game developers usually place bounding boxes manually in the scene to indicate the influence range of probes, which is time-consuming and inflexible. Most recently proposed leaking-free probe GI frameworks [5][6][3] tend to use visibility tests to exclude probes that are invisible from the shading point during interpolation. However, the culling procedure cannot be performed by hardware directly, thus requires 4˜8 "loading" operations instead of 1 "sampling" operation, which multiplies the time cost in rendering. So far there is no such a method that can eliminate light leaking completely without introducing significant manual workload or rendering overhead.

We address the light leaking problem in probe GI using a new approach, Mask Decomposition (MD). MD automatically decomposes probes into different volume textures using baked masks, enabling the masked light field to be directly sampled using hardware-aided tri-linear interpolations. With probes decomposed into volumes, any shading point only needs to sample probe volumes whose masks are marked at the position of the shading point, which significantly reduces the sampling cost for culling out probes. Experiments show that MD is at least 3× faster than the most optimized visibility based GI framework,which makes it comparable to the original light leaking GI. MD can be applied to any probe-based GI system, so long as the geometry structure of the scene does not change at run time.

In particular, we make the following contributions:

1. We introduce Mask Decomposition, a lightweight framework that addresses the light leaking problem

Jixiang Zhou and Yanzhen Chen contributed equally to this research.

Xiaogang Jin, jin@cad.zju.edu.cn

Jixiang Zhou · Yanzhen Chen · Xiaogang Jin
State Key Lab of CAD&CG, Zhejiang University, Hangzhou 310058, China

Yuanheng Li · Shun Cao · Yu Wu
Tencent, Unit Two, Building C, Kexing Science Park, Kejizhongsan Avenue, Nanshan District, Shenzhen 518057, China

in probe-based GI systems without introducing significant rendering overhead.
2. We develop an algorithm based on simulated annealing to efficiently assign masks to probes base on their visibility to each other with a high precision.
3. We present a compression method that can significantly compress the dense mask volume texture we use in our framework into a sparse representation of a smaller size.

2 Related Works

Global Illumination (GI) is a longstanding topic in physically-based rendering, especially in real-time applications such as modern video games. There are mainly two approaches to implement GI in real-time: performing ray-tracing with the support of ray-tracing accelerated GPUs; or caching the precomputed GI information (e.g. light transport) of the scene and sampling it at run time.

To achieve the best performance, methods like Screen Space GI (SSGI) [9] are proposed to approximately solve ray-tracing problems in screen space. Nowadays, full real-time ray-tracing is also available with the help of GPUs that support hardware-accelerated ray-tracing. However, for mobile devices, real-time ray-tracing is still too expensive and currently not feasible. Therefore, we focus on the other type of approaches — light field caching — in this paper.

2.1 Probe GI

Light probe was first introduced by McGuire et al. [7] as a simplification of Irradiance Caching (IC) [4]. Afterwards, probe methods became widely used in irradiance caching, especially for diffuse lighting.

The lighting probes are points scattered in the scene, storing the precomputed GI information at each position. At the shading time, the diffuse light field is reconstructed by interpolation of the probes. Due to its low cost and acceptable shading results, probe-based GI technology has been generally used in mobile video games.

In 2020, NVIDIA proposed Diffuse Global Illumination (DDGI) [5], which utilizes RTX ray-tracing accelerated GPUs to update light probes with ray-tracing at run-time.

Probes can not only be organized in a uniform grid layout, but also in a hierarchy architecture [2] to compress the light field of an extremely large scale. Wang et al. [12] leveraged a sparse voxel octree to organize the probes, and used gradient descent to minimize the number of probes. These methods introduce significant changes to the pipeline and take several times of time in querying,

2.2 Probe Visibility Test

The original probe-based GI framework suffers from light leaking. NVIDIA's RTXGI [6] can leverage the depth information and Variance Shadow Maps (VSM) [1] to represent visibility fields around the probes. As a result, it can prevent light leaking artifacts. However, this method has technical limitations on hardware and cannot be applied to mobile devices.

The light leaking problem can also be eliminated using signed distance fields (SDFs) [3]. By representing the scene using SDFs, which can be considered as a stand-in of conventional triangle meshes, it is able to accelerate the computation of GI.

2.3 Light Field Compression

Another class of light field caching approaches directly compress the light field itself into a compact representation.

One approach to compress light fields is Clustered Principal Component Analysis (CPCA)([11]). [8] concluded that the key idea of compression is to leverage the spatial redundancy of the signals and then presented clustered Blockwise Principal Component Analysis (BPCA) . Moving Basis Decomposition (MBD) [10] is a general framework on the compression of spatial signals. It decomposes high frequency and high dimensional signals into the dot product of low frequency, high dimensional basis, and high frequency, low dimensional coefficients. They use the result of PCA for initialization and solve the MBD problem by Quasi-Newton iteration.

3 Methodology

In this section, we will formulate our key idea of mask decomposition (Section 3.2), consider the related graph coloring problem (Section 3.3), and present the full algorithm (Section 3.4).

Figure 1 illustrates the framework of our Mask Decomposition method. In a typical pipeline of Probe GI with visibility test, each probe has a depth map, which is similar to the variance shadow map (VSM) [1], and the visibility test is just a "depth test". In our pipeline, we calculate mask vectors for the whole space, and the

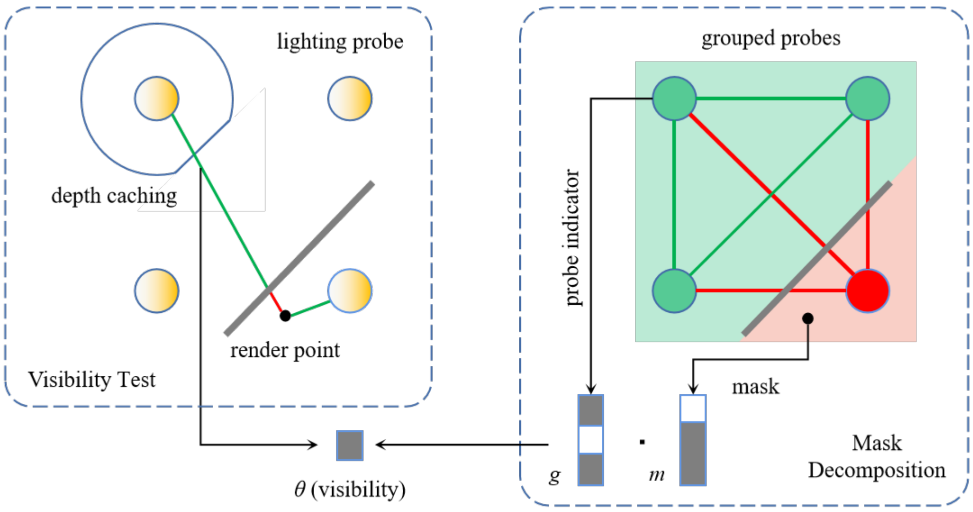


Fig. 1 Difference between Visibility Test and Mask Decomposition. Visibility test methods use the variance shadow map (VSM) to cache the depth of each probe in each direction. Mask Decomposition divides the space into several clusters (2 clusters in this example colored by red and green) and uses mask vectors to indicate the category of each spatial position. The dot product of mask vectors denotes the approximate visibility between two positions.

dot product of mask vectors of two close points implies the visibility of these two points. Such a representation is differentiable and interpolatable.

3.1 Background

A typical probe-based GI system discretizes one light field into probes (sample point) at the baking time, then reconstructs the light field by linearly interpolating among probes at run time. The probes are usually organized into one 3D lattice grid, which can be stored as a volume texture.

3.1.1 Notation

Light fields are usually defined as 3D fields of *Spherical Harmonics* (SH) or *Light Transport Matrices* (LTM, the linear operators on SH). We write one light field as $L(x) : \mathbb{R}^3 \to \mathbb{L}$, where $x \in \mathbb{R}^3$ is the position vector in world space, and $\mathbb{L}$ is the mapped SH or LTM space.

Given the precomputed (diffuse) light field P being discretized by a uniform grid of probes, we write $i = (i_i, i_2, i_3) \in \mathbb{I}$ as the index of this grid ($\mathbb{I} \subset \mathbb{N}^3$ is the index space), and $x(i)$ as its world position. For brevity, we use the subscript (or superscript when a subscript exists) for indexing (e.g. P_i) or field sampling (e.g. P_x), instead of using function invocation forms.

The sampling of probes is accomplished by interpolation of probes, and we adopt the kernel form:

$$P_x := \sum_i \phi_i(x) P_i,$$

where $\phi : \mathbb{R}^3 \times \mathbb{I} \to [0, 1]$, is usually a tri-linear weighting function. Such tri-linear weighting interpolation is performed by the hardware when sampling the volume texture.

3.1.2 Light Field Reconstruction

Most of the leaking-free probe-based GI systems, such as RTXGI[6], leverage a visibility test to prevent light leaking. As a result, the sampling is no longer able to be written in a kernel form of interpolation. We have:

$$L_{\text{ProbeGI}}(x) = P_x, \quad (1)$$

$$L_{\text{RTXGI}}(x) = \sum_i \theta_i(x)\phi_i(x)P_i, \quad (2)$$

where $\theta : \mathbb{R}^3 \times \mathbb{I} \to [0, 1]$ is the visibility function from position x to probe i. To determine the visibility of each probe, we have to perform at least 8 loading operations (P_i) to reconstruct the light field, and cannot use hardware-accelerated tri-linear interpolation.

3.2 Mask GI

We want to make leaking-free probe-based GI to be *hardware-interpolatable*, so that the number of sampling operations can be reduced. Instead of using one volume texture to represent all probes, we partition probes into K clusters using an one-hot cluster indicator $g : \mathbb{I} \to \{0, 1\}^K$. Then the light field can be constructed by

$$L_{\text{MaskGI}}(x) = m_x \cdot (g \otimes P)_x. \quad (3)$$

where $m : \mathbb{R}^3 \to [0,1]^K$ is a normalized mask value specifying the visibility of one cluster from the shading position x. The mask value m can be computed using various approaches, in our implementation, we perform visibility tests by casting rays from x to the point, line or plane formed by probes in the bounding cell of x and in the target cluster as well, then we average the visibility values to get the final mask value.

Given the light field being discretized by one uniform grid of probes, the maximum value for K is 8, indicating that every probe in one grid cell will be partitioned into a different cluster. In such case, Equation (3) is close enough to Equation (2), by defining $g(i)$ as follow:

$$g^k(i) = \begin{cases} 1 \text{ if } \bigwedge_{d=1}^{3}[\mathrm{Bit}(k,d) = \mathrm{Bit}(i_d, 1)], \\ 0 \text{ otherwise,} \end{cases} \tag{4}$$

where $\mathrm{Bit}(k,d)$ is the d-th digit in reverse order of the binary representation of k. With this definition, each pair of probes i,j within a single cell should have different indicators $g(i) \neq g(j)$, which means that the visibility of all 8 probes with non-zero weights in (2) can be modeled separately using different mask values.

The number K is chosen by the user according to the performance budget for the GI algorithm. Since K maps directly to the number of clusters (3D textures) and the maximum number of sampling operations to perform, a smaller K value saves memory and shading time, but may cause sufficient precision to model the visibility discontinuity for some cells. If K is limited to a number smaller than 8, we solve an optimization problem by defining a loss term as the distance between Equation (3) and Equation (2):

$$\begin{aligned} R &= \int \left\| \sum_i \theta_x^i \phi_x^i P_i - \sum_k^K m_x^k \sum_i \phi_x^i g_i^k P_i \right\|_F \mathrm{d}x \\ &= \int \left\| \sum_i \phi_x^i P_i (\theta_x^i - m_x \cdot g_i) \right\|_F \mathrm{d}x \\ &\leq \int \sum_i \phi_x^i \|P_i\|_F \left|\theta_x^i - m_x \cdot g_i\right| \mathrm{d}x. \end{aligned}$$

The optimization is approximately achieved when the simplified residual $\hat{R}$ is minimized:

$$\hat{R} = \sum_{i,x} \phi_x^i \left|\theta_x^i - m_x \cdot g_i\right|, \tag{5}$$

which is not correlated to the light field value. Note that we do not distinguish between discrete summation and integration in the rest of this paper for convenience.

3.3 Graph Coloring

Assume that $g(i) = g(j)$, residual $\hat{R}$ will increase by:

$$E(i,j) := \sum_x \min(\phi_x^i, \phi_x^j) \cdot \left|\theta_x^i - \theta_x^j\right|, \tag{6}$$

which is defined as the *edge-wise* loss penalizing *node-wise* difference. Consider the probe grid as a graph G whose edges are weighted by $E(i,j)$, and the optimization problem of Equation (5) becomes equivalent to a graph coloring problem. Then, we employ simulated annealing to solve this graph coloring problem. Our loss function is:

$$\mathcal{L}(g) := \sum_{i,j} \mathbf{1}_{g_i = g_j} \cdot E(i,j). \tag{7}$$

3.3.1 Regularization.

To improve the sparsity of $m(x)$, we add a regularization term into the loss to penalize filling adjacent nodes with $E = 0$ in different colors:

$$\mathcal{L}_{\mathrm{reg}}(g) := \frac{\lambda}{2} \sum_i \sum_{j \in \mathcal{N}(i)} \mathbf{1}_{g_i \neq g_j}, \tag{8}$$

and we empirically choose $\lambda = 0.001$ in our experiments.

3.3.2 Light-aware Optimization.

Since the formulation of Mask Decomposition is similar to Moving Basis Decomposition, we directly optimize the reconstructed light field. Given g, we solve m by minimizing:

$$\begin{aligned} \mathcal{L}_{\mathrm{opt}}(g) := \sum_x \left\| L_{\mathrm{MaskGI}}(x) - L(x) \right\|^2 \\ + \lambda \sum_{x,k} \mathbf{1}_{m_{x,k}^{\mathrm{initial}} = 0} |m_{x,k}|. \end{aligned} \tag{9}$$

This optimization may decrease the sparsity of m and increase the rendering time overhead, so we also add a regularization to penalize filling zero channels of initial m_x with non-zero values.

3.4 Algorithm

3.4.1 Mask Decomposition

We present the full algorithm of Mask Decomposition as follow:

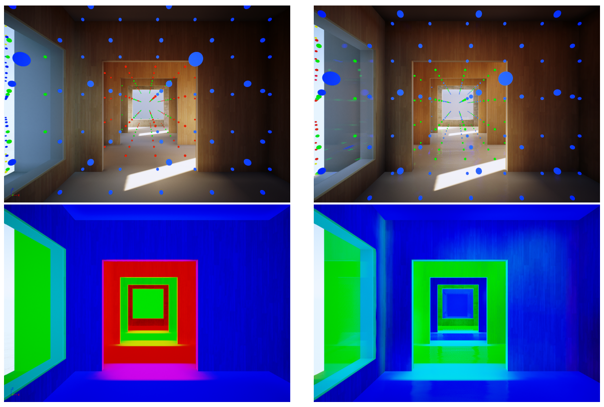

(a) Simple Mask (b) Light-aware Optimized Mask

Fig. 2 Colored probes and baked maskmaps. Probes are divided into different clusters and filled with different colors. Every pixel in maskmaps is colored according to colors of probes visible from its position.

1. Calculate the ground-truth $\theta_i(x)$ (e.g. the visibility using VSM). x is discretized by a refined grid (e.g. 6 times the resolution of the probe grid).
2. Calculate $E(i, j)$ using Equation (6), within a single traversal.
3. Solve $g(i)$ by simulated annealing. We initialize g with a convex division of the original probe grid. Each part of the division forms a convex hull, whose nodes have the same initial color. This method works better than random initialization.
4. Given g, we initialize $m(x)$ on the refined grid with $\theta_i(x)$ that we have stored in step (1), by optimizing Equation (5) locally.
5. (Optional) We scale $m(x)$ to directly minimize the reconstruction loss of Equation 9 by gradient descent, just similar to compression methods. This is the only step that uses the content of the probes.
6. Finally, we filter m with a Gaussian kernel.

3.4.2 Mask Compression.

We also developed an algorithm that efficiently compresses mask volume textures used in our framework. We noticed that data in mask volume textures are highly sparse, so we divide mask data into cubes formed by 8 probes, and use one indirection volume texture to index cubes. In this way, cubes with the same mask data can be merged and thus the mask volume texture can be significantly reduced. Experiment shows that after compression, the texture size is reduced to about 1/16 of the original size, at the cost of only one additional sampling call at run time.

Table 1 Base-Pass time cost measurements

Methods	Computer 1	Computer 2
GI Disabled	0.07 ms	0.31 ms
Leaking Probe GI (Baseline)	0.09 ms	0.37 ms
Mask GI (Ours)	0.11 ms	0.43 ms
Visibility Test without culling	0.38 ms	0.99 ms
Visibility Test with culling	0.25 ms	1.01 ms

4 Results

In this section, we present the experimental result of our approach. We implement our algorithm in Unreal Engine 4.26. As a comparison, we also adopt the official RTXGI plug-in for UE4 to shade visibility GI result, as well as the native volumetric lightmap system in UE4 to shade the baseline (leaking probe GI) result.

4.1 Space Division

Figure 2 demonstrates the results of our space-division and mask baking algorithm. The algorithm performs 500˜1000 simulated annealing iterations in 5˜10 seconds, producing a satisfactory result that is suitable for most application cases. Our space-division algorithm converges fast and behaves stably in multiple runs. In most cases, setting the number of clusters to 3 will be enough, while using more unnecessary colors will instead decrease the sparsity of the result. However, some complex meshes in the scene (like trees with abundant leaves) may cause unexpected color waste. Although such waste will not affect the correctness of the algorithm, we suggest excluding such meshes when baking masks or replacing them with simpler proxy meshes.

4.2 Reconstruction Quality

Figure 3 compares our result with the baseline (Light-leaking GI), RTXGI, and the ground-truth (static lightmap without probe GI). In the experiment, we constructed one scene of one house with multiple rooms and windows. We set the probe grid resolution to 16x32x16 for all probe-based approaches from (a) to (d), and the cluster number K to 4 for our mask GI approaches (c) and (d), then followed UE4 convention to represent the light field using Spherical Harmonics. Lightmap textures for static objects were used to store pre-baked maskmaps in our approach (a) and (d), so no lightmap lighting contributed to our result. Shading errors were measured by pixel-wise standard deviations between

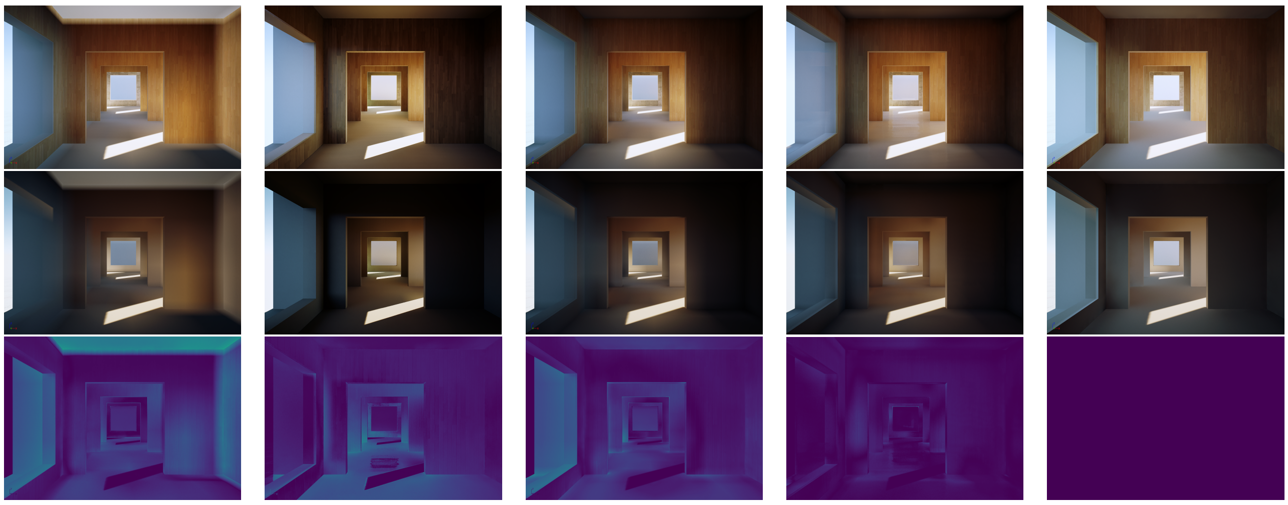


Fig. 3 Comparison. We present the rendering result (Top), the lighting visualization (Middle) and the pixel-wise error (Bottom) of different implementations. a) Baseline (Probe GI without leaking elimination). b) RTXGI (Visibility-based Probe GI). c) Mask Decomposition without light information. d) Mask Decomposition with light-aware optimization. e) The Ground-truth illuminated using lightmaps.

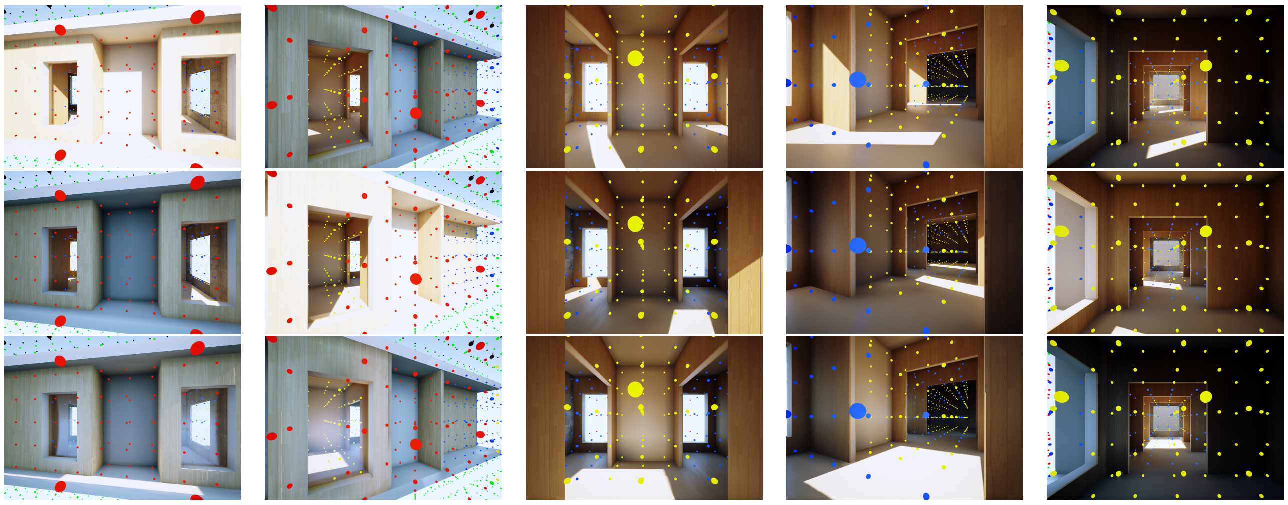


Fig. 4 Decoupling of Mask Decomposition and lighting environment. From top to bottom, the lighting environment changes without re-dividing of probe volumes or re-baking of masks (of the version without lighting-aware optimization). The rendering result is convincing from each point of view (in different columns). The scene becomes slightly darker since we do not scale the masks to fit the lighting environment. However, the light leaking is still perfectly eliminated.

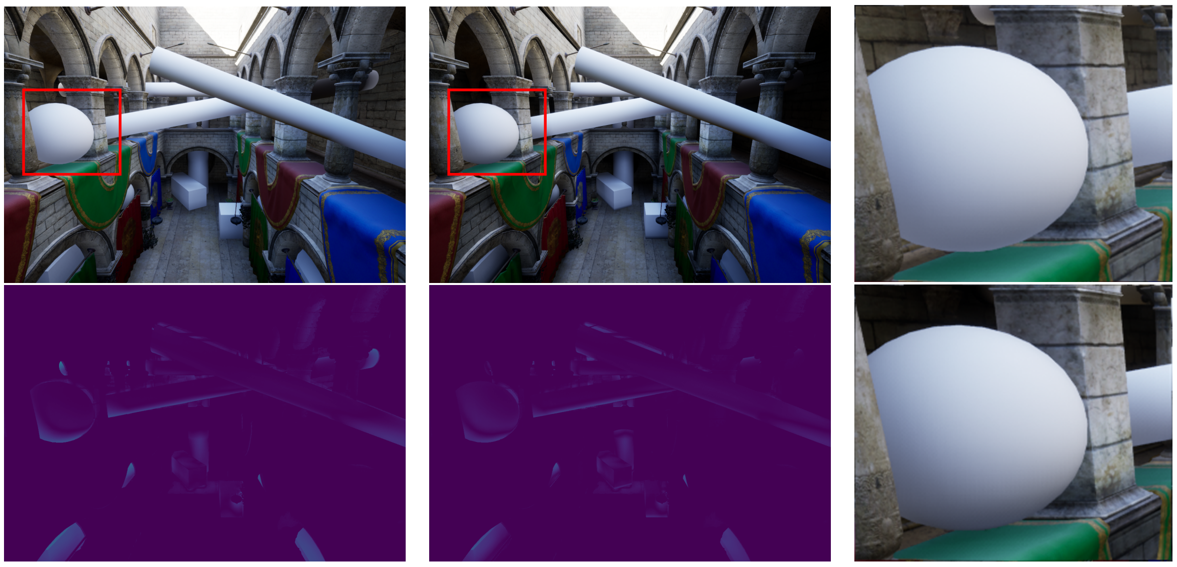
the current shading result and the ground-truth:

$$E(x) := \sqrt{\frac{1}{3}\sum_{n}^{RGB}(C_n(x) - C'_n(x))^2}$$

The pixel-wise error values are then visualized by heatmaps ranging from 0 to 1 in the third row of Figure 3. We also computed the root mean square error (RMSE) value for each image comparison:

$$E_{RMSE} := \sqrt{\frac{1}{s}\sum_{i}^{s}(C(i) - C'(i))^2}$$

where s is the number of pixels in the image. Color values were represented in normalized (0 to 1) range for all error measurements. We can see from Figure 3 that the shading quality of our approach is very close to RTXGI, but with a much lower rendering cost.



(a) Leaking Probe GI (b) Ours (c) Detail Comparison

Fig. 5 Dynamic object rendering. Figure (b) shows dynamic object rendering using mask volume in our approach. Compared to leaking probe GI in figure (a), our approach also reduces light leaking for dynamic objects. Figure (c) highlights one of the incorrect brighter areas on the 2nd floor in figure (a) and (b).

Table 2 Spatial cost measurements

Type	Probe	K-Probes	Mask m	Flag g	Indirection Texture	Compressed Mask
Voxel Bytes	28	112	1	1	4	1
Volume Size	$16 \times 32 \times 16$	$16 \times 32 \times 16$	$16 \times 32 \times 16 \times 8^3$	$16 \times 32 \times 16$	$16 \times 32 \times 16$	$252 \times 252 \times 6$

Table 3 Precomputing performance measurements

Entry	MBD	Ours
Coef/Mask Resolution	$64 \times 64 \times 64$	$96 \times 192 \times 96$
Basis/Probe Resolution	$9 \times 9 \times 9$	$16 \times 32 \times 16$
Basis/Probe Channels	324	27
Hardware Requirements	GPU Concurrent	CPU Serial
Time cost (full bake)	199s	183s
Complexity	Non-linear	Linear

Figure 5 demonstrates the shading result of our approach for dynamic objects. We placed several white spheres, cubes, and cylinders in the Sponza scene, and marked them as dynamic objects, so no maskmap is baked for them. Figure 5 shows that light leaking is also well addressed for dynamic objects.

4.3 Spatial and Temporal Performance

Table 1 compares the temporal cost of the "BasePass" shading stage in UE4 using different GI strategies. We use two computers to render the same sample room scene presented in Figure 3, from the same camera view. Computer 1 is equipped with Intel i9-9900K CPU, 32GB of system memory, and NVIDIA RTX 3070 GPU; computer 2 is equipped with AMD Ryzen 7 1800X CPU, 32GB of system memory, and NVIDIA GTX 1080 Ti GPU.

For visibility GI, we implemented both with/without culling versions. For the culling version, we only load one probe when its visibility term is not close to 0, while for the version without culling, we always load 8 adjacent probes for every sample point. When implementing visibility GI, we cache the visibility from every probe to adjacent probes in precomputation, and load them directly when shading to further increase performance. Even with these optimizations, the average sample/load count for mask GI is still much lower than visibility GI, which makes it at least 2x faster.

Table 2 displays the spatial cost for mask GI in our experiment. Mask GI uses one volume texture per probe cluster, every volume texture takes 0.21MB to store all probes in it, summing to 0.84MB for 4 clusters. For mask volumes, we choose 8x8x8 as the sample density between every 8 probes that form one cube. Since the mask volume is sparse, we then compress the mask volume, merging cubes with the same local data to one cube entry in the mask atlas volume, then address the cube using one separate indirection volume texture. After compression, the mask volume size is reduced to 1.08MB, about 10$\times$ smaller than the original size, while RTXGI would store at least 16×16 depth textures for every probe, about 4.00MB in total, to achieve the same precision. Therefore, even with 4$\times$size growth in probe volumes, the total volume size is not bigger than that of RTXGI.

Table 3 demonstrates the precomputing performance of Mask GI in our experiment. Compared to MBD, our approach costs less time even with a higher resolution and without any concurrency. Although our approach only stores 27-dimension SH rather than 324-dimension LTM, our mask decomposition algorithm is not related to the probe data itself, but to the probe filtering by their visibility, thus the number of dimensions for each probe data is not relevant to the performance comparison of the algorithms.

4.4 Decoupling of Visibility and Lighting

Figure 5 shows the decoupling of decomposed masks and the lighting information content of probes. In this case, we do not scale masks to fit the lighting intensity for the complete decoupling of mask and lighting, so the scene may look darker than average. We change the lighting environment along the rows of Figure 5 without updating the baked masks, and the rendering result is still convincing from each point of view.

4.5 Discussion

In this section, we will discuss the relationship between *Mask Decomposition* and *Moving Basis Decomposition* (MBD)[10].

MBD is a generalization of space-related compression algorithms, which includes *Mask Decomposition* (MD). In their formulation, mask decomposition is a specialization of MBD with the number of bases equal to 4.

[10] mentioned that applications of MBD framework could have their own task-related kernels, which is what Mask Decomposition does. Mask Decomposition assumes that all discontinuation of the signal comes from spatial occlusion, which is tenable only in precomputed light field rendering semantics.

Our approach utilizes this assumption and considerably increases the sparsity of B and c term until they only have non-zero values in 1 channel, making its performance close to the MBD result with $K = 1$.

In addition, since our algorithm is mostly decoupled with the light data stored in probes, the mask may not need to be recomputed even when the scene light condition changes, so long as the scene geometry structure does not changed, which is also an attribute that MBD does not have.

The limitation of our approach is the lack of generalization ability. But since Mask Decomposition keeps all premises that Probe GI has, it is capable of handling all cases that Probe GI supports.

5 Conclusion

We present a lightweight framework, Mask Decomposition (MD), to deal with the leaking problem of Probe GI in mobile games. Mask Decomposition is based on Leaking-free Probe GI with visibility test and resolves its problem of interpolation. We prove that the Mask Decomposition problem has a 0-loss solution with $K = 8$. Then, we present an algorithm based on graph coloring to solve Mask Decomposition problems efficiently.2 Finally, we compare the result of Mask Decomposition with visibility methods and show that Mask Decomposition performs well both in the usage of time and space while maintaining an approximately close render quality to visibility methods.

6 Limitation and Future Works

The Mask Decomposition is a static method and does not adopt to the geometry changing at run time. One run time system may be implemented to update masks when geometry changes, or multiple sets of masks can be prepared and switched at run time if all scene changes can be determined at baking time. We are still exploring approaches to adopt MD for dynamic scene changing.

Currently, the choice of the probe grid resolution in MD is strongly limited. An 8-probe cage should neither be larger than the static geometries nor smaller than the dynamic geometries, otherwise causing unacceptable artifacts. Apparently the best value can be determined automatically by analyzing the scene structure. We are developing algorithms to choose the suitable value for every unique scene.

The Mask Decomposition renders much faster than visibility-based GI algorithms, making it suitable for performance-limited devices like mobile phones. We're migrating our framework to mobile platforms to test its performance on such platforms.

References

1. Donnelly, W., Lauritzen, A.: Variance shadow maps. In: Proceedings of the 2006 symposium on Interactive 3D graphics and games, pp. 161–165 (2006)
2. Garcia, K., Lindqvist, A., Brinck, A.: " hustle by day, risk it all at night" the lighting of need for speed heat in frostbite. In: Special Interest Group on Computer Graphics and Interactive Techniques Conference Talks, pp. 1–2 (2020)
3. Hu, J., Yip, M.K., Alonso, G.E., Gu, S., Tang, X., Jin, X.: Efficient real-time dynamic diffuse global illumination using signed distance fields. The Visual Computer pp. 1–13 (2021)
4. Krivanek, J., Gautron, P.: Practical global illumination with irradiance caching. Synthesis lectures on computer graphics and animation **4**(1), 1–148 (2009)
5. Majercik, Z., Guertin, J.P., Nowrouzezahrai, D., McGuire, M.: Dynamic diffuse global illumination with ray-traced irradiance fields. Journal of Computer Graphics Techniques Vol **8**(2) (2019)
6. Majercik, Z., Marrs, A., Spjut, J., McGuire, M.: Scaling probe-based real-time dynamic global illumination for production. arXiv preprint arXiv:2009.10796 (2020)
7. McGuire, M., Mara, M., Nowrouzezahrai, D., Luebke, D.: Real-time global illumination using precomputed light field probes. In: Proceedings of the 21st ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games, pp. 1–11 (2017)
8. Nishino, K., Nayar, S.K., Jebara, T.: Clustered blockwise pca for representing visual data. IEEE Transactions on Pattern Analysis and Machine Intelligence **27**(10), 1675–1679 (2005)
9. Ritschel, T., Grosch, T., Seidel, H.P.: Approximating dynamic global illumination in image space. In: Proceedings of the 2009 symposium on Interactive 3D graphics and games, pp. 75–82 (2009)
10. Silvennoinen, A., Sloan, P.P.: Moving basis decomposition for precomputed light transport. In: Computer Graphics Forum, vol. 40, pp. 127–137. Wiley Online Library (2021)
11. Sloan, P.P., Hall, J., Hart, J., Snyder, J.: Clustered principal components for precomputed radiance transfer. ACM Transactions on Graphics (TOG) **22**(3), 382–391 (2003)
12. Wang, Y., Khiat, S., Kry, P.G., Nowrouzezahrai, D.: Fast non-uniform radiance probe placement and tracing. In: Proceedings of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games, pp. 1–9 (2019)