Computational Visual Media
https://doi.org/10.1007/s41095-023-0367-z Vol. 10, No. 5, October 2024, 923–936

Research Article

Real-time all-frequency global illumination with radiance caching

Youxin Xing[1,*], Gaole Pan[2,*], Xiang Chen[1], Ji Wu[1], Lu Wang[1] (✉), and Beibei Wang[2] (✉)

Abstract Global illumination (GI) plays a crucial role in rendering realistic results for virtual exhibitions, such as virtual car exhibitions. These scenarios usually include all-frequency bidirectional reflectance distribution functions (BRDFs), although their geometries and light configurations may be static. Rendering all-frequency BRDFs in real time remains challenging due to the complex light transport. Existing approaches, including precomputed radiance transfer, light probes, and the most recent path-tracing-based approaches (ReSTIR PT), cannot satisfy both quality and performance requirements simultaneously. Herein, we propose a practical hybrid global illumination approach that combines ray tracing and cached GI by caching the incoming radiance with wavelets. Our approach can produce results close to those of offline renderers at the cost of only approximately 17 ms at runtime and is robust over all-frequency BRDFs. Our approach is designed for applications involving static lighting and geometries, such as virtual exhibitions.

Keywords real-time global illumination; all-frequency BRDFs; Haar wavelets; radiance caching

1 Introduction

The effects of realistic materials, all-frequency shadows, and global illumination are significant for photorealistic rendering, enhancing its realism. However, the computation of these effects is time-consuming, especially for real-time rendering.

Our method focuses mainly on virtual exhibitions, e.g., virtual car exhibitions. The scenarios usually have dynamic views in such an application and might cover all-frequency bidirectional reflectance distribution functions (BRDFs), but with static lighting and geometries. Therefore, we make the same assumptions in our study.

In the real-time rendering domain, the precomputed radiance transfer (PRT) of Sloan et al. [1], Ng et al. [2, 3], and light probes [4, 5] are widely used. Approaches based on PRT support all-frequency shadows, glossy reflections, and dynamic lighting at the cost of expensive storage. Most light probe-based approaches focus only on diffuse materials, such as dynamic diffuse global illumination (DDGI) [5].

Recently, path tracing, combined with advanced sampling strategies and denoising, has become a possible solution for real-time global illumination. Advanced sampling strategies include resampled importance sampling (RIS) [6] for both direct illumination [7] and global illumination [8]. Neither of these methods work well for low-roughness materials. Recently, ReSTIR PT [9] enabled all-frequency material rendering owing to generalized resampled importance sampling (GRIS); however, the results are not noise-free.

This study aimed to achieve real-time global illumination effects with all-frequency shadows and interreflections on glossy objects. For this purpose, we propose a practical hybrid global illumination solution that combines ray tracing and cached GI. The cached GI is responsible for direct illumination from non-delta light sources (e.g., area lights and environment maps) and indirect illumination of objects with low-to-intermediate frequency BRDFs from all light sources. Ray tracing handles the indirect illumination of

* Youxin Xing and Gaole Pan contributed equally to this work.

1 School of Software, Shandong University, Jinan 250101, China. E-mail: Y. Xing, youxinxing@mail.sdu.edu.cn; X, Chen, xiangchen@mail.sdu.edu.cn; J. Wu, jiwu@mail.sdu.edu.cn; L. Wang, luwang_hcivr@sdu.edu.cn (✉).

2 School of Computer Science and Engineering, Nanjing University of Science and Technology, Nanjing 210094, China. E-mail: G. Pan, pangaole@njust.edu.cn; B. Wang, beibei.wang@njust.edu.cn (✉).

Manuscript received: 2023-02-22; accepted: 2023-07-01

specular or near-specular materials from all light sources and direct illumination from delta light sources (e.g., point and directional lights).

In our cached GI approach, we used wavelets to represent the incoming radiance and BRDFs at a precomputation step and performed efficient convolution during rendering. In particular, we cache irradiance for diffuse materials. Finally, our method can provide high-quality results that match the references in only approximately 17 ms.

Table 1 Notations

Ψ_j, Ψ_k	Orthonormal basis functions
γ	Clear-coat parameter
$\boldsymbol{\omega}_i, \boldsymbol{\omega}_o$	Incident and outgoing directions
c_{base}	Diffuse color defined for the material
C	Mathor scaling Coeffs
$\boldsymbol{C}_{\text{BRDF}}$	Coeffs vectors of the glossy BRDF
$\boldsymbol{C}_{\text{radiance}}$	Coeffs vectors of cached radiance
D_i	Detail wavelet Coeffs
f_c	Clear-coat term of the clear-coat BRDF
f_d, f_g	BRDFs of diffuse and glossy materials
f_r	BRDF
f_s	Specular term of glossy materials' BRDF
F_c	Fresnel term of the clear-coat BRDF
l_j, t_k	Coeffs of light and light transport
$\boldsymbol{L}, \boldsymbol{T}$	Coeffs vectors of light and light transport
L_i	Light source
$L_{\text{irradiance}}$	Irradiance of a point
L_o	Outgoing radiance
T	Transport operator
V	Binary visibility
$\boldsymbol{x}, \boldsymbol{n}$	Position and normal of shading point

2 Previous work

Precomputed radiance transfer (PRT). Sloan et al. [1] used the spherical harmonic (SH) basis to restore soft shadows, reflections, and caustic effects. Ng et al. [3] replaced the SH with wavelets; therefore, their method can represent all-frequency shadows and reflections. Wang et al. [10] introduced PRT and a separable BRDF approximation, allowing for the rendering of glossy objects in complex and dynamic lighting environments. PRT-based approaches can handle global illumination effects with dynamic lighting at the cost of expensive storage.

Real-time Monte Carlo path tracing. Recently, RIS has been introduced to real-time rendering to sample direct illumination, low-frequency GI, and high-frequency GI. The recent generalized form of RIS allows for glossy indirect illumination. Müller et al. [11] further introduced neural radiance caching to real-time path tracing to accelerate rendering by learning and rendering the radiance distribution online using a neural network.

Light probes. Irradiance volume [12, 13] and light probe-based approaches have been widely used in the video game industry owing to their efficiency. They subdivided scenes into discretized points, represented the irradiance distribution at these points during precomputation, and queried light probes during rendering. Majercik et al. [5] proposed a dynamic solution for diffuse global illumination. They updated both irradiance and visibility using ray tracing at runtime. Most of these studies focused only on diffuse materials, except for one recent study by Rodriguez et al. [14], which allowed for glossy interreflections. However, this method is time-consuming and unsuitable for real-time rendering. Majercik et al. [15] also extended DDGI to glossy materials; however, they forced second-order glossy reflections to maximum roughness, leading to inaccurate results.

Point-based global illumination. The point-based global illumination (PBGI) is first proposed by Christensen [16] for color bleeding in the offline rendering domain. The point cloud and hierarchical structure are treated as geometric proxies of the geometry; therefore, they can be rasterized for visibility. PBGI has also been extended to real-time rendering [17, 18]; however, it focuses on diffuse global illumination. It cannot handle glossy interreflections and caustics because the radiance is represented by SH. Later, Wang et al. [19] replaced SH with wavelets, allowing for non-diffuse light transport, but their work was too heavy for real-time rendering.

3 Background and overview

In this section, we first introduce the rendering equation and PRT. We then provide an overview of our approach.

3.1 Rendering equation

The rendering equation [20] is the core of global illumination:

$$L_o(\boldsymbol{x}, \boldsymbol{\omega}_o) = \int_{\Omega} L_i(\boldsymbol{\omega}_i) f_r(\boldsymbol{x}, \boldsymbol{\omega}_i, \boldsymbol{\omega}_o)(\boldsymbol{n} \cdot \boldsymbol{\omega}_i) \mathrm{d}\boldsymbol{\omega}_i \quad (1)$$

where $\boldsymbol{\omega}_i$ is the incident direction, $\boldsymbol{\omega}_o$ is the outgoing direction (also called the viewing direction), and $\boldsymbol{n}$ is the surface normal. L_o is the outgoing radiance at

position $\boldsymbol{x}$ from the viewing direction $\boldsymbol{\omega}_o$, L_i is the lighting, and f_r is the BRDF.

3.2 Precomputed radiance transfer

The rendering equation can be reformulated to achieve real-time rendering by caching part of its composition, such as in PRT.

Ng et al. [3] defined a transport operator T, which includes both the BRDF and a visibility term V:

$$T(\boldsymbol{x}, \boldsymbol{\omega}_i, \boldsymbol{\omega}_o) = f_r(\boldsymbol{x}, \boldsymbol{\omega}_i, \boldsymbol{\omega}_o) V(\boldsymbol{x}, \boldsymbol{\omega}_i)(\boldsymbol{n} \cdot \boldsymbol{\omega}_i) \quad (2)$$

Then, Eq. (1) is transformed into the integral of the product of the incoming light and light transport operator:

$$L_o(\boldsymbol{x}, \boldsymbol{\omega}_o) = \int_{\Omega} L_i(\boldsymbol{\omega}_i) T(\boldsymbol{x}, \boldsymbol{\omega}_i, \boldsymbol{\omega}_o) \mathrm{d}\boldsymbol{\omega}_i \quad (3)$$

To achieve real-time frame rates, L_i and T are precomputed and represented by an appropriate orthonormal basis function $\Psi_j(\boldsymbol{\omega}_i)$. j and k are serial numbers of the basis functions. For a given shading point with fixed values for $\boldsymbol{x}$ and $\boldsymbol{\omega}_o$,

$$L_i(\boldsymbol{\omega}_i) = \sum_j l_j \Psi_j(\boldsymbol{\omega}_i) \quad (4)$$

$$T(\boldsymbol{\omega}_i) = \sum_k t_k \Psi_k(\boldsymbol{\omega}_i) \quad (5)$$

This results in the final formation in Eq. (6):

$$\begin{aligned} L_o &= \int_{\Omega} \left(\sum_j l_j \Psi_j(\boldsymbol{\omega}_i)\right) \left(\sum_k t_k \Psi_k(\boldsymbol{\omega}_i)\right) \mathrm{d}\boldsymbol{\omega}_i \\ &= \sum_j l_j t_j = \boldsymbol{L} \cdot \boldsymbol{T} \end{aligned} \quad (6)$$

Finally, the integral is converted into a dot product of the coefficients vectors $\boldsymbol{L}$ and $\boldsymbol{T}$.

Compared with PRT, our method can compute and store radiance using highly compressed wavelet coefficients at each point. Aided by an octree structure for query acceleration and a wavelet quadtree structure for fast convolutions, our method can obtain all-frequency shadows and interreflections at runtime.

3.3 Overview

The crucial insight of our approach is to cache the lighting and BRDFs in a precomputed step and then perform efficient convolution of these two components during rendering. The specular or near-specular effects and direct illumination of delta light sources that are too sharp for the cached GI are computed by ray tracing.

More specifically, in the precomputation step (Section 4), we generate the point clouds and represent the materials (BRDFs) with Haar wavelets first. We then represent the lighting by caching the incoming radiance distribution on point clouds with wavelets and organizing the point clouds into a spatial hierarchy constructed by octrees. During rendering (Section 5), we search the hierarchy to find the incoming radiance distribution and then obtain the product of the wavelet coefficients for the lighting and BRDF, as shown in Fig. 1.

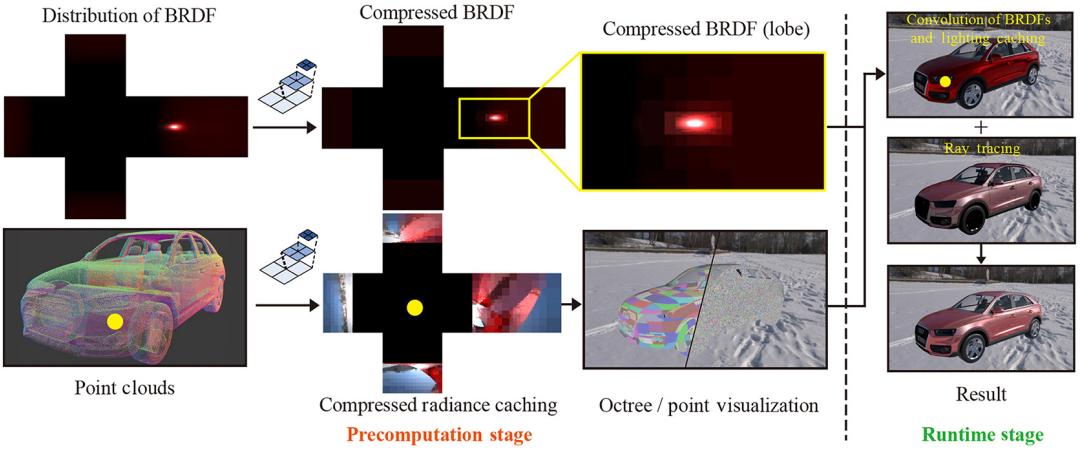

Fig. 1 Overview of our hybrid global illumination method on a Vehicle scene. The precomputation is shown on the left, and the runtime rendering is on the right. In the precomputation stage, each BRDF is precomputed as a half-cube map and compressed with Haar wavelets (top-left). The scene is discretized into several point clouds by the mesh index (ID). The incoming radiance distribution of each point in point clouds is also precomputed as a half-cube map and represented with Haar wavelets. Then, the whole spatial hierarchy (octrees) is constructed to organize the lighting of the point clouds. Note that the wavelets for each half-cube map are organized in a quadtree manner. During rendering, given a shading point, we traverse the spatial hierarchy to obtain a cached point and then perform the convolution of BRDFs and lighting caching, resulting in partial results. We compute the direct illumination of delta lights directly without caching and shoot rays on the specular or near-specular surfaces and then combine them with the cached GI to obtain the final rendered results.

4 BRDFs and lighting precomputation

In the precomputation step, each mesh is discretized into a point cloud (Section 4.1). Then, we precompute all BRDFs in the scene and represent each of them with wavelets (Secion 4.2). Next, we precompute and compress the incoming radiance distribution of each point and organize them into octrees (Section 4.3).

4.1 Point cloud generation

For each mesh in the scene, we generate a point cloud using Poisson disk sampling [21]. Given the desired point count M, we first generate more points using random sampling, where the number of points for each mesh triangle corresponds to its area. In practice, we generate $5\times M$ randomly distributed points. Then, we eliminate the points iteratively to obtain a uniformly distributed point cloud. Note that the sample points are organized by a KD-Tree and organized into a heap structure for efficient elimination.

To determine which sample point to eliminate, we measure the closeness of each point to its neighboring points using a *weight*. W_{ij} is the *weight* between the sample points i and j $(i \neq j)$:

$$W_{ij} = \left(1 - \frac{\min(d_{ij}, 2r_{\max})}{r_{\max}}\right)^{\alpha} \tag{7}$$

where d_{ij} is the distance between sample points i and j. $r_{\max}$ is the maximum radius, set as $\sqrt{A_2/(2\sqrt{3}n)}$, where A_2 is the sampling area. n is the desired number of samples after elimination, and α is an exponential constant used to control the *weight*, which is set to 8 in practice. The *weight* W_i of sample point i is the sum of W_{ij} corresponding to sample point j within $2r_{\max}$ distance of sample point i.

At each iteration, we eliminate the sample point with the highest *weight* and dynamically adjust the *weights* of remaining sample points. The iteration stops when the number of sample points meets the input point count. In this way, the distribution of the remaining points becomes uniform.

4.2 BRDFs precomputation and compression

In our study, we focused on three types of materials: diffuse, glossy, and clear-coat BRDFs.

For the diffuse material, we used the Lambertian diffuse material:

$$f_d(\boldsymbol{\omega}_i, \boldsymbol{\omega}_o) = \frac{1}{\pi} c_{\text{base}} \tag{8}$$

where c_{base} denotes the diffuse color.

We then use the Cook–Torrance model [22] for the glossy material, which includes both the diffuse term f_d defined by Eq. (8) and the specular term f_s:

$$f_g(\boldsymbol{\omega}_i, \boldsymbol{\omega}_o) = f_d(\boldsymbol{\omega}_i, \boldsymbol{\omega}_o) + f_s(\boldsymbol{\omega}_i, \boldsymbol{\omega}_o) \tag{9}$$

where f_s denotes a typical microfacet model [23]:

$$f_s(\boldsymbol{\omega}_i, \boldsymbol{\omega}_o) = \frac{D(h)F(\boldsymbol{\omega}_o, h)G(\boldsymbol{\omega}_i, \boldsymbol{\omega}_o, h)}{4(\boldsymbol{n}\cdot\boldsymbol{\omega}_i)(\boldsymbol{n}\cdot\boldsymbol{\omega}_o)} \tag{10}$$

where D is the normal distribution function (NDF) [24], F is the Fresnel term, and G is the masking-shadowing function [25]. We use GGX as our NDF and Schlick's approximation [26] for the Fresnel term.

Finally, we add a clear-coat term from the Google Filament engine [27] for the glossy material:

$$f_{\text{clearcoat}}(\boldsymbol{\omega}_i, \boldsymbol{\omega}_o) = f_g(\boldsymbol{\omega}_i, \boldsymbol{\omega}_o)(1 - F_c) + f_c(\boldsymbol{\omega}_i, \boldsymbol{\omega}_o) \tag{11}$$

where f_c is the clear-coat BRDF modeled with a typical microfacet model, and F_c is the Fresnel term of the clear-coat BRDF:

$$F_c = [0.04 + 0.96(1 - \boldsymbol{\omega}_o \cdot h)^5]\gamma \tag{12}$$

where γ is a clear-coat parameter of the material that controls the strength of the clear-coat effect.

Precomputation. The diffuse material is computed at runtime, as described in Section 5.2. For each glossy material, we precompute the distribution of the BRDF. We sample the outgoing direction $\boldsymbol{\omega}_o$ using uniform hemisphere sampling with sampling density set to 172×1080. Then, for each sampled $\boldsymbol{\omega}_o$, we represent the distribution of the incoming direction $\boldsymbol{\omega}_i$ using a half-cube map. Here, we use the local coordinate system by aligning the z-axis of the cube map with the surface shading normal. In practice, for each $\boldsymbol{\omega}_o$, we compute the BRDF value according to $\boldsymbol{\omega}_i$ at each pixel of the half-cube map, which is set to 128×128.

Compression. We use the quadtree form wavelet transform to project the half-cube map of BRDFs onto Haar bases. The quadtree is constructed in a bottom–up manner. Each node contains a mother scaling coefficient C, three detail wavelet coefficients D_i $(i = 0, 1, 2)$, and four child indices. The coefficients of the current node are computed using the mother scaling coefficients of its child nodes, which are defined by c_j $(j = 0, 1, 2, 3)$:

$$\begin{aligned} C &= \frac{c_0 + c_1 + c_2 + c_3}{2}, \; D_0 = \frac{c_0 - c_1 + c_2 - c_3}{2} \\ D_1 &= \frac{c_0 + c_1 - c_2 - c_3}{2}, \; D_2 = \frac{c_0 - c_1 - c_2 + c_3}{2} \end{aligned} \tag{13}$$

If the child node is a leaf node, c_i is the pixel color in the half-cube map. During compression, when all coefficients of a node and its child nodes are below a certain threshold (β), we discard them. Thus, we can control the coefficient quadtree's degree of compression using this threshold. In practice, we set β to 0.2 when the roughness value is greater than 0.2; otherwise, it is 0.1.

4.3 Lighting caching and octree construction

For each point in the point clouds, we precompute the incoming radiance distribution or irradiance and organized each point cloud into an octree.

Lighting caching. Objects of different material types were treated differently. We store the irradiance rather than the incoming radiance for an object with a diffuse material because it is view-independent.

$$L_{\text{irradiance}}(\boldsymbol{x}) = \int_{\Omega} L_i(\boldsymbol{x}, \boldsymbol{\omega}_i)(\boldsymbol{n} \cdot \boldsymbol{\omega}_i)\mathrm{d}\boldsymbol{\omega}_i \tag{14}$$

where $L_{\text{irradiance}}$ is the irradiance at position $\boldsymbol{x}$ of each point in the point cloud.

In contrast to the diffuse material, we precompute the incoming radiance for the object with the glossy material, regardless of whether it has a clear coat. During caching, we locate a half-cube map around each point and compute the incoming radiance for the half-cube map using path tracing (the sample count is set as 128 for each path). The resolution of the radiance cube map is determined by the roughness of the object material, as shown in Table 2. Then, we compress each half-cube map with wavelets in a quadtree form, as with the BRDFs. The discard threshold β is set to 1000.

Table 2 Resolution of radiance cube map under different roughness values

Roughness	Resolution
[0.10, 0.35]	128×128
(0.35, 0.55]	64×64
(0.55, 0.65]	32×32
(0.65, 1.0]	16×16

Octree construction. We organize the point clouds into octrees, where each point stores the lighting caching, including the irradiance or incoming radiance, represented by wavelet coefficients in quadtree form.

We construct the octree in a top–down manner. Starting from the axis-aligned bounding box (AABB) of the entire mesh, each node is uniformly subdivided into eight child nodes according to the AABB. This subdivision of each node continues when the point count at each leaf node is greater than a certain threshold (set to 30 in practice). Finally, the leaf nodes of the octree contain all the cached points.

After constructing the octrees for all meshes, we update the information of each node in a bottom–up manner, including the bounding box and normal. The bounding box of the leaf node is computed using the bounding box of its points, while the bounding box of non-leaf nodes is computed using the union of the bounding boxes of its child nodes. The normal of each node is set as the average of the normals in its points or child nodes.

5 Real-time global illumination

During runtime, we first compute the results of ray tracing. Then, we search and convolute the BRDFs and lighting caching. Finally, we merge them to obtain the final real-time global illumination.

5.1 Ray tracing

For ray tracing, we computed the direct illumination of delta lights and clear-coat effects. We also trace rays to compute the indirect illumination for low-frequency materials. For diffuse materials, the equation is as Eq. (15):

$$L_o(\boldsymbol{x}, \boldsymbol{\omega}_o) = f_d(\boldsymbol{\omega}_i, \boldsymbol{\omega}_o)L_i(\boldsymbol{x}, \boldsymbol{\omega}_i)(\boldsymbol{\omega}_i \cdot \boldsymbol{n}) \tag{15}$$

For glossy materials, the equation is as Eq. (16):

$$L_o(\boldsymbol{x}, \boldsymbol{\omega}_o) = f_g(\boldsymbol{\omega}_i, \boldsymbol{\omega}_o)L_i(\boldsymbol{x}, \boldsymbol{\omega}_i)(\boldsymbol{\omega}_i \cdot \boldsymbol{n}) \tag{16}$$

For glossy materials with clear-coat effects, the equation is as Eq. (17):

$$\begin{aligned} L_o(\boldsymbol{x}, \boldsymbol{\omega}_o) =& f_g(\boldsymbol{\omega}_i, \boldsymbol{\omega}_o)L_i(\boldsymbol{x}, \boldsymbol{\omega}_i)(\boldsymbol{\omega}_i \cdot \boldsymbol{n})(1 - F_c) \\ &+ f_c(\boldsymbol{\omega}_i, \boldsymbol{\omega}_o)L_i(\boldsymbol{x}, \boldsymbol{\omega}_i)(\boldsymbol{\omega}_i \cdot \boldsymbol{n}) \end{aligned} \tag{17}$$

We trace an extra clear-coat ray along the direction of the original ray's specular reflection to compute the clear-coat term. The maximum tracing depth is set to 3.

For objects with low-frequency materials, when a ray (including the clear-coat ray) hits a shading point with a roughness of less than 0.1, it is treated as a mirror. In addition, we continue to trace the ray along the direction of the perfect specular reflection until it hits the environment map or reaches its maximum tracing depth. When the ray hits the environment map, we compute the indirect illumination with ambient light L_{env}:

$$L_o(\boldsymbol{x}, \boldsymbol{\omega}_o) = F(\boldsymbol{\omega}_o, h)L_{\text{env}}(\boldsymbol{x}, \boldsymbol{\omega}_i) \tag{18}$$

where F denotes the Fresnel term. In practice, the maximum tracing depth is set to 4.

5.2 Convolution of BRDFs and lighting caching

During rendering, we search for the caching of BRDFs and lighting and then compute their convolution in quadtree form. The BRDF coefficients are queried through the material index (ID) and viewing direction $\boldsymbol{\omega}_o$ using inverse uniform hemisphere sampling in the hemispherical space. The lighting caching is queried in three steps. First, we determine the octree using the mesh ID of the shading point. We then search for leaf nodes that contain the target point in the octree using a distance threshold. Finally, we select the most appropriate shading point in the leaf nodes according to the mixed weight of the position and normal direction.

After searching for diffuse materials, we computed the result using the diffuse color c_{base} and irradiance $L_{\text{irradiance}}$:

$$L_o(\boldsymbol{x}, \boldsymbol{\omega}_o) = \frac{1}{\pi} c_{\text{base}} L_{\text{irradiance}}(\boldsymbol{x}) \tag{19}$$

For glossy materials, we convolve the coefficients of BRDFs $\boldsymbol{C}_{\text{BRDF}}$ and cached radiance $\boldsymbol{C}_{\text{radiance}}$ in the quadtree form to obtain the final result:

$$L_o(\boldsymbol{x}, \boldsymbol{\omega}_o) = \boldsymbol{C}_{\text{BRDF}} \otimes \boldsymbol{C}_{\text{radiance}} \tag{20}$$

For a glossy material with a clear coat, we additionally multiply by the ratio $(1 - F_c)$ as Eq. (21):

$$L_o(\boldsymbol{x}, \boldsymbol{\omega}_o) = (\boldsymbol{C}_{\text{BRDF}} \otimes \boldsymbol{C}_{\text{radiance}})(1 - F_c) \tag{21}$$

The clear-coat term is computed using ray tracing.

We consider the ZeroDay and Vehicle scenes as examples to show the results of ray tracing and convolution of BRDFs and lighting caching in Figs. 2 and 3, respectively.

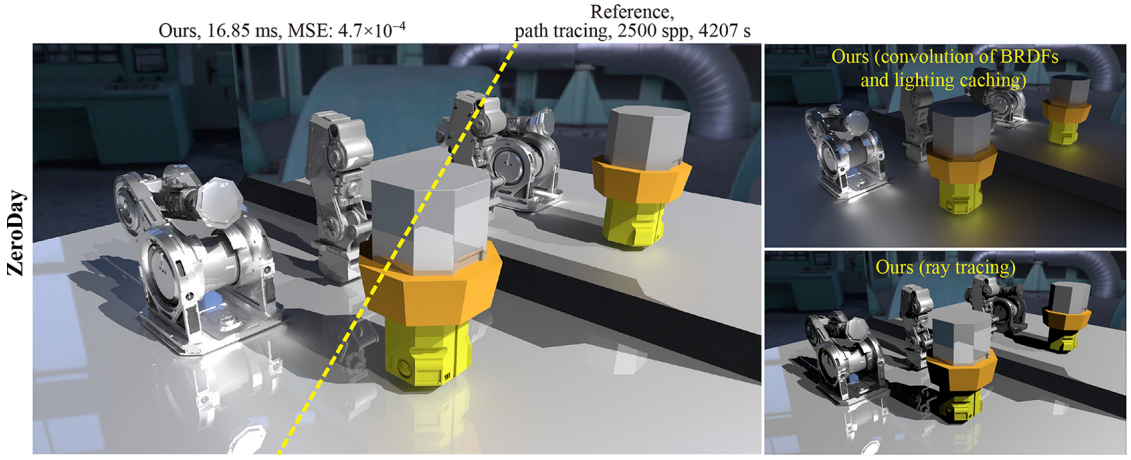

Fig. 2 Results of ZeroDay using our method. The comparison between our final result and the reference is shown on the left. The result of the convolution of BRDFs and lighting caching is shown on the top right. The ray tracing result is shown on the bottom right.

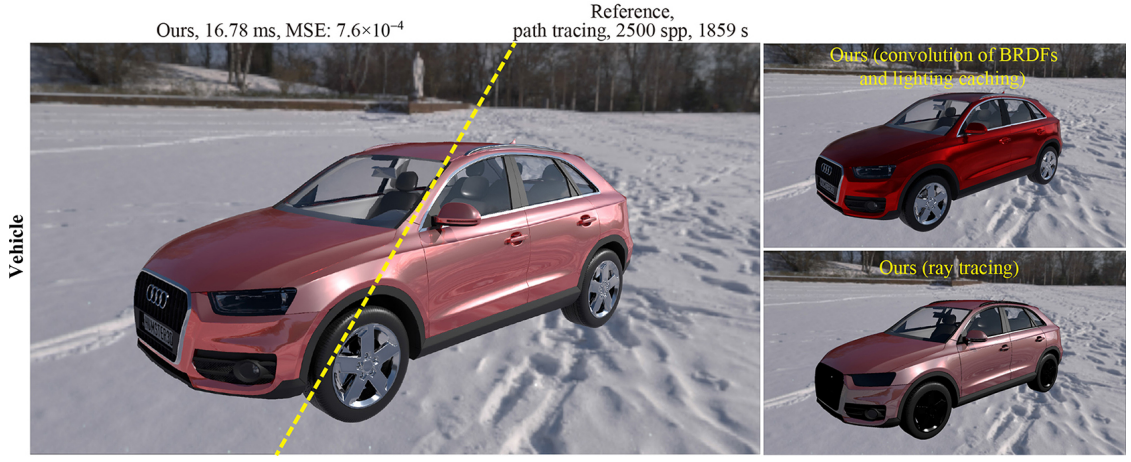

Fig. 3 Results of Vehicle using our method. The comparison between our final result and reference is shown on the left. The result of the convolution of BRDFs and lighting caching is shown on the top right. The ray tracing result is shown on the bottom right.

6 Implementation details

In this section, we focus on the details of the BRDFs precomputation, lighting caching, octree construction, and runtime rendering. While precomputing the BRDFs, we simultaneously dispatched the tasks to 16,384 GPU threads for GPU acceleration. The lighting caching tasks were dispatched to 8192 GPU threads in practice. We implemented our method using the Falcor GPU rendering framework (4.3 version) [28] for runtime rendering.

BRDFs precomputation. Before precomputation, for BRDFs defined with textures (e.g., diffuse map), we categorized BRDFs with similar properties (diffuse color) into groups, and then performed BRDF precomputation for each group by treating it as a single BRDF. The indices of these groups were stored in the alpha channel of the BRDF texture.

When computing the half-cube maps for BRDFs for $\boldsymbol{\omega}_o$, we applied stratified sampling to $\boldsymbol{\omega}_i$ in the half-cube map to avoid stripe artifacts in the rendering results. During compression, we stored the coefficients of BRDFs and cached the radiance in the half-precision floating-point type to reduce storage size, which was sufficient to obtain the same rendering results as the floating-point type.

Lighting caching and octree construction. While computing the lighting, we divided the radiance in each radiance half-cube map by the solid angle in the corresponding direction [29] for energy conservation. After computation, for diffuse materials, we added all incident radiances stored in the half-cube map to obtain the irradiance for each point.

We constructed an octree for each mesh to improve the query accuracy for the cached points at mesh intersections. The reflections and shadows of objects with diffuse or high-roughness glossy materials are view-independent. Thus, for objects with low-frequency reflections and shadows, we computed the result with sparse points for lighting caching, which had a similar result as with the dense points (e.g., the inner shadow on the floor in Fig. 4(bottom)). In practice, during construction, we removed the points that had minor energy differences from neighboring points by probability based on the statistics of the average energy differences in the entire point cloud. The point elimination rate was approximately 47%.

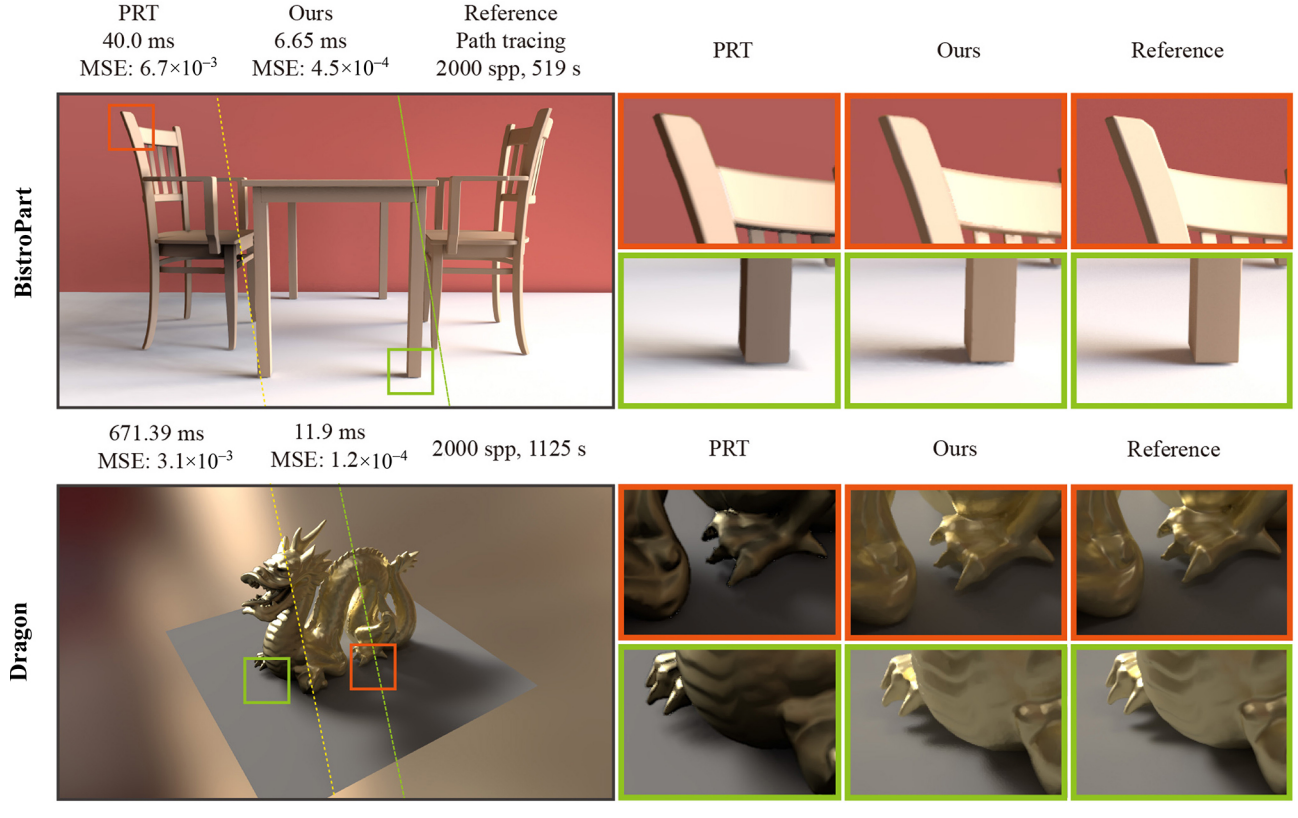

Fig. 4 Comparison between PRT and our method on the BistroPart (top) and Dragon (bottom) scenes.

Runtime rendering. We implemented the proposed method in three passes: *V-Buffer*, *GI*, and *TAA* pass. In the V-Buffer pass, we generated the initial V-Buffer to cache the instance type, instance index, primitive index, and barycentric coordinates of each shading point. The GI pass is the main pass used to compute ray tracing and the convolution of BRDFs and lighting caching. Finally, the results were merged to obtain the final global illumination. We computed mipmap levels for normal maps in this pass to reduce flicker artifacts. The TAA pass is used for anti-aliasing.

While searching BRDFs, we first obtained four BRDF coefficient quadtrees from the four neighboring directions of the viewing direction $\boldsymbol{\omega}_o$ using inverse uniform hemisphere sampling. We then performed bilinear interpolation on the four quadtrees to reduce the discontinuity of the rendering results and storage of the BRDFs coefficients while using a sparse $\boldsymbol{\omega}_o$ sampling density. After interpolation, the new quadtree's depth was the same as the lowest of the four quadtree depths.

When querying the lighting caching in an octree, we controlled the maximum number of search points in all searched nodes using the parameter m to improve the search speed. In addition, we discarded the points whose energy was lower than a set threshold because this is probably the point under the object surfaces or in the folds and might influence the search of other points.

During the convolution of BRDFs and lighting caching, we controlled the traversal layers of quadtrees to balance performance and quality. Moreover, we reduced the computation time and offered a method to simulate the effect of high-roughness glossy materials using a low quadtree depth. However, this inevitably resulted in artifacts. Therefore, we made a tradeoff in practice.

7 Results

First, we compared our global illumination method with PRT (Ng et al. [3]), ReSTIR PT, and DDGI. Next, we showed the performance measures of our method in different test scenes and the parameter analysis in the Dragon scene. We implemented ReSTIR PT using the released code and reimplemented PRT by us. References were computed using standard path tracing with multiple samples per pixel (spp). We quantified the error using mean squared error (MSE). All results were rendered on a high-end desktop machine (i9 10900k and RTX 3090) at a resolution of 1920×1080.

7.1 Comparison with previous work

Comparison with PRT. The BistroPart scene (Fig. 4(top)) contains three diffuse materials and an environment map. In this scene, we cached the irradiance at 400,000 points only. PRT cached the visibility distribution at 34,325 vertices, lighting distribution from the environment map, and the BRDF distribution of three materials. Compared with the 1.61 GB caching storage of PRT, our method has the advantage of storage, which is only 0.02 GB. It was almost six times faster than PRT at runtime.

In the Dragon scene (Fig. 4(bottom)), there are two glossy materials and an environment map. The caching in our method consists of the radiance distribution at 81,579 points and BRDF distribution of two glossy materials. The caching of PRT includes the visibility distribution at 100,277 vertices, lighting distribution from the environment map, and BRDF distribution of two materials. Our method's caching storage is 3.12 GB, which is larger than PRT's (2.47 GB) because the BRDF distribution of glossy materials accounts for most of the caching storage. At runtime, the performance of the proposed method was approximately 56 times faster than PRT.

In comparison, PRT precomputes the lighting distribution from an environment map for every vertex, whereas our method precomputes an independent lighting distribution for each point. Thus, our method can compute all-frequency shadows and reflections that have more details than PRT, especially in the high-frequency parts.

Comparison with ReSTIR PT. We compared the results of our method with those of ReSTIR PT in the VeachAjar, Vehicle, Matballs, and ZeroDay scenes, as shown in Figs. 5 and 6. All materials in these scenes are glossy. In addition, the Vehicle (car shell), Matballs (cyan ball), and ZeroDay (floor and canons) scenes exhibit a clear-coat effect.

In contrast to ReSTIR PT, the results of our method are essentially noise-free owing to the benefits of ray tracing and the convolution of BRDFs and lighting caching without sampling. No color bias was observed in the results. The results of ReSTIR PT had an obvious color bias, especially for objects with a clear coat and low-frequency BRDFs (e.g., car

Fig. 5 Comparison between ReSTIR PT and our method on the scenes including VeachAjar (top), Vehicle (middle), and Matballs (bottom).

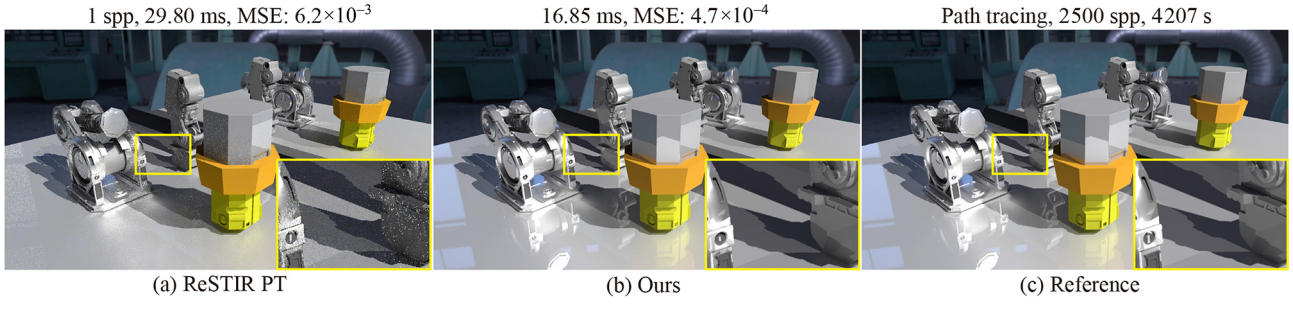

Fig. 6 Compared to ReSTIR PT (a) on the ZeroDay scene, our approach (b) is noise-free and has similar quality with reference (c), in approximately 17 ms per frame.

shell). This is because it is limited to shift mapping strategies for path reuse. At runtime, the time cost of our method was less than that of ReSTIR PT with 1–3 spp in the test scenes.

Comparison with DDGI. We compared the indirect illumination results of our method with those of DDGI [5] in the Table scene, as shown in Fig. 7. The Table scene contains four diffuse materials, a point light, and a directional light. We precomputed the irradiance at 400,000 points, and the caching of DDGI contained $45 \times 8 \times 45$ light probes. With an equal time cost at runtime, our method required less caching storage (22.80 MB) than DDGI (31.64 MB). In addition, our results preserved richer indirect lighting details and had a lower MSE than DDGI during rendering.

7.2 Performance and storage

We present the scene scale (number of triangles) and performance measures of our method in Table 3 for different test scenes. For general complex scenarios, our method requires approximately 17 ms per frame at runtime, with almost no quality loss compared to the reference. However, our method requires up to several hours of precomputation time. Nonetheless, as long as the scene does not change (with static

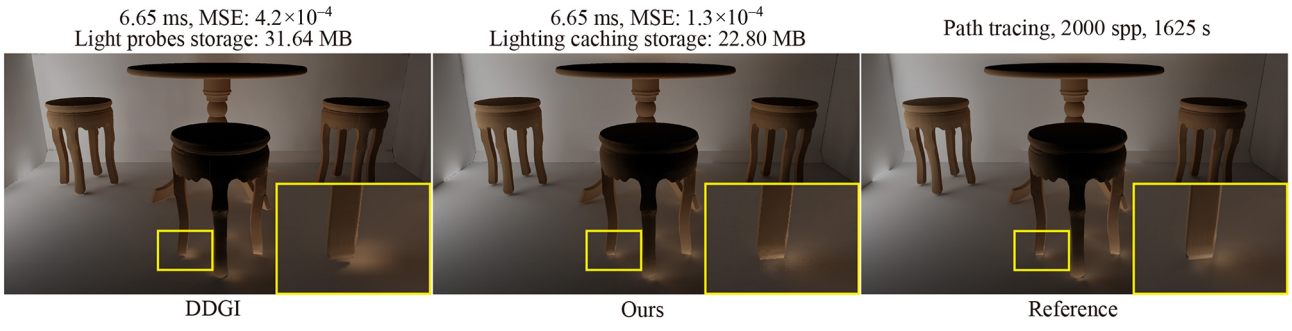

Fig. 7 Comparison of the indirect illumination between DDGI and our method on the Table scene.

objects and lighting), the precomputation only needs to be performed once.

Table 3 also shows the storage of the cached information and runtime memory of our method. In the precomputation stage, the storage of lighting caching is influenced by the geometric complexity of the meshes, which directly determines the number of points. In addition, it is affected by the complexity of the lighting distribution at these points. For example, in the VeachAjar scene, both the geometry and lighting distribution are complex, many points are required to store the lighting information. Furthermore, we cached the lighting distribution with a larger half-cube map size for scenes with low-roughness materials, as shown in Table 2. For those two scenes of VeachAjar and Vehicle, we can see that the VeachAjar scene has more points but large roughness values (0.17–1.0), while the Vehicle scene has fewer points, and its car shell has a low roughness value (0.11). Thus, the Vehicle scene requires more than twice the lighting caching storage compared with the former. The storage of BRDF caching is mainly determined by the number of glossy materials and the distribution complexity of the BRDFs, which are related to the roughness. A scene with diffuse materials (e.g., the BistoPart scene) requires less storage than a scene with glossy materials (e.g., the Dragon scene). The main memory and GPU memory at runtime depend primarily on the total storage of BRDFs and lighting caching.

Figure 8 shows our method's time cost of ray tracing, convolution, and others. Other methods include V-Buffer computing, primary ray shooting, intersecting, etc. The VeachAjar scene contains only the area of the light source behind the door. All other scenes include an environment map. Additional delta

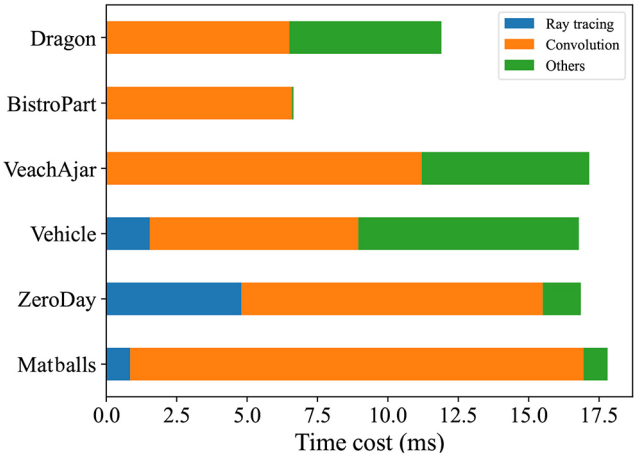

Fig. 8 Time cost during runtime using our method. Note that we computed global illumination with irradiance instead of the convolution of BRDFs and lighting caching in the BisroPart scene.

Table 3 Performance measures of our method in different test scenes. The precomputed storage (pre-storage) includes the caching of BRDFs and lighting. The precomputed time (pre-time) includes the time cost of BRDFs and lighting

Scene	No. of triangles	No. of points	No. of materilas	Pre-time (h)		Pre-storage (GB)		Runtime memory		Runtime time (ms)
				BRDFs	Lighting	BRDFs	Lighting	Main (GB)	GPU (GB)	
Dragon	132,768	81,579	2	2	4	2.55	0.57	3.8	4.40	11.90
BistroPart	38,938	400,000	3	0	10	0	0.02	0.678	1.01	6.65
VeachAjar	750,900	1,317,000	11	6	31	6.73	6.43	11.4	13.54	17.15
Vehicle	889,241	600,000	14	5	30	3.38	15.90	19.55	20.30	16.78
ZeroDay	419,615	692,000	6	2.5	20	4.39	1.61	6.66	7.99	16.85
Matballs	203,640	500,000	5	2.5	13.5	6.28	7.75	14.71	15.20	17.80

lights were provided for the Vehicle and ZeroDay scenes. In Fig. 8, the scenes of Dragon, BistroPart, and VeachAjar have no time cost for ray tracing because we computed the global illumination for non-delta lights using BRDFs and lighting caching instead of ray tracing. When computing the convolution, GPU parallelism is influenced by the different traversal depths of the coefficient quadtrees, which depend on the BRDF roughness. The roughness values of the materials in the Matballs scene cover a large range from 0.2 to 0.8, which makes the traversal difficult, so the Matballs scene's convolution of BRDFs and lighting caching accounts for the largest time cost.

Our method also supports normal maps with the same storage as BRDFs for flat materials. Figure 9 shows our results of different normal maps in the CarInterior scene. Regardless of how bumpy the normal maps are, we only precomputed BRDFs without the normal maps in the local space. Then, for a certain type of normal map, while searching the BRDFs using $\boldsymbol{\omega}_o$ at runtime, we transformed $\boldsymbol{\omega}_o$ from world space to local space with different normal vectors. Therefore, the normal maps influence only the coordinate system transformation of $\boldsymbol{\omega}_o$. In addition, we do not need to precompute other new BRDFs. In Fig. 9, our results with and without normal maps both have the same BRDF storage (0.35 GB). For more bumpy normal maps, the GPU takes more time to access the memory randomly to obtain the cached BRDF coefficients at runtime. Therefore, our result with bumpy normal maps consumes 2.76 ms time cost more than that without normal maps.

7.3 Parameter analysis

Varying number of points. We generated point clouds with different numbers of points for the Dragon scene and show the performance measures of our method in Table 4. The corresponding results are shown in Fig. 10. With more points, the caching storage for lighting increases significantly, leading to increased time costs during rendering. However, the quality of our results also improved, while the MSE between our results and the references decreased.

Varying roughness. In Fig. 5, we show the results of five material balls with varying roughness values (0.2, 0.4, 0.6, 0.8, and 0.8 with a clear-coat

Table 4 Performance measures of our method with different point numbers in the Dragon scene

Points	Precomputation oflighting		Runtime	
	Time (h)	Storage (MB)	Time cost (ms)	MSE (10^{-4})
81,579	4	582	11.9	1.2
50,320	1.83	314	11.3	2.1
33,762	1	176	11.2	2.4
19,396	0.5	93.4	11.1	3.6
9,698	0.32	46.9	10.8	6.0

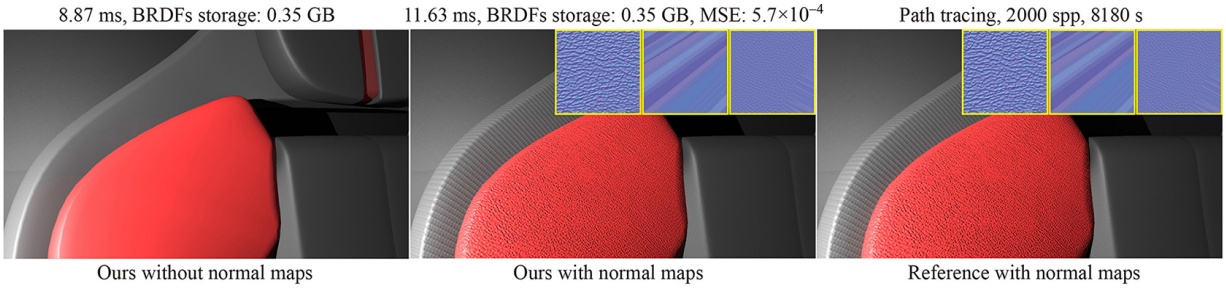

Fig. 9 Comparison of the results without (left) or with (middle) normal maps using our method on the CarInterior scene. The reference with normal maps is shown on the right.

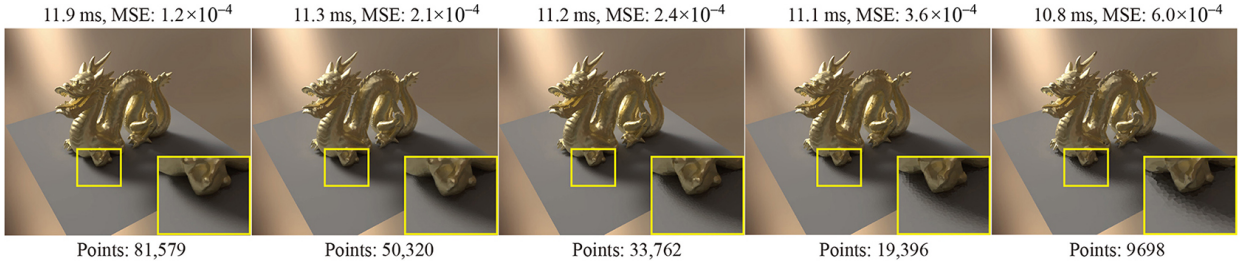

Fig. 10 Comparison of the results with different sampled point amounts using our method on the Dragon scene.

effect) in the Matballs scene. Our method obtained the same results as the offline renderer in a wide roughness range, with a 17.8 ms time cost at runtime.

7.4 Limitations and discussions

We recognize the following limitations of this study. First, to simplify the query of BRDFs, we discretized $\boldsymbol{\omega}_o$ in a hemispherical space and computed the distribution of BRDFs with $\boldsymbol{\omega}_i$. This inevitably resulted in considerable storage capacity. We should consider organizing $\boldsymbol{\omega}_o$ and $\boldsymbol{\omega}_i$ from a four-dimensional perspective to obtain a more compact representation. Second, for BRDFs defined by textures, an excessive number of colors in the texture map results in a large number of BRDFs, which leads to large caching storage.

Our method supports general complex scenarios, such as the Vehicle scene with 889,241 triangles and 95 objects. However, for extremely complex scenarios (e.g., a large-scale forest scene), it remains a challenge for our method to store a large number of points to maintain the lighting details. A feasible solution is to use a neural network to represent the BRDFs and lighting caching, which is an interesting topic for future research.

8 Conclusions and future work

In this paper, we presented a new hybrid real-time global illumination method that combines ray tracing and the convolution of BRDFs and lighting caching. During precomputation, we offer a point cloud generator to compute the points that conform to the Poisson distribution, a new wavelet compression structure in quadtree form for BRDFs and cached radiance, and a compact spatial hierarchy for lighting caching. At runtime, our method produces results close to those of the offline renderer in only 17 ms, based on various optimizations, such as octree-accelerated searching and quadtree-accelerated convolutions. It can compute all-frequency shadows and reflections in static scenes, which is noise-free. For scenes with diffuse materials, our method is almost six times faster than PRT and requires less caching storage at runtime. It is suitable for applications that allow static lighting and geometric scenes, primarily in virtual exhibitions.

In the future, we will introduce deep learning to provide a more compact representation and avoid interpolation of lighting and BRDFs. Furthermore, our approach may serve as a good proxy for path guiding.

Acknowledgements

We thank the reviewers for their valuable suggestions. This work was partially supported by the National Key R&D Program of China under Grant No. 2020YFB1709203, the National Natural Science Foundation of China under Grant Nos. 62272275 and 62172220, and the Shandong Provincial Natural Science Foundation of China under Grant No. ZR2020LZH016.

Declaration of competing interest

The authors have no competing interests to declare that are relevant to the content of this article.

References

[1] Sloan, P. P.; Kautz, J.; Snyder, J. Precomputed radiance transfer for real-time rendering in dynamic, low-frequency lighting environments. In: Proceedings of the 29th Annual Conference on Computer Graphics and Interactive Techniques, 527–536, 2002.

[2] Ng, R.; Ramamoorthi, R.; Hanrahan, P. All-frequency shadows using non-linear wavelet lighting approximation. In: Proceedings of the ACM SIGGRAPH Papers, 376–381, 2003.

[3] Ng, R.; Ramamoorthi, R.; Hanrahan, P. Triple product wavelet integrals for all-frequency relighting. In: Proceedings of the ACM SIGGRAPH Papers, 477–487, 2004.

[4] Kontkanen, J.; Laine, S. Sampling precomputed volumetric lighting. *Journal of Graphics Tools* Vol. 11, No. 3, 1–16, 2006.

[5] Majercik, Z.; Guertin, J. P.; McGuire, M. Dynamic diffuse global illumination with ray-traced irradiance fields. *Journal of Computer Graphics Techniques* Vol. 8, No. 2, 1–30, 2019.

[6] Talbot, J. F.; Cline, D.; Egbert, P. Importance resampling for global illumination. In: Proceedings of the 16th Eurographics Conference on Rendering Techniques, 139–146, 2005.

[7] Bitterli, B.; Wyman, C.; Pharr, M.; Shirley, P.; Lefohn, A.; Jarosz, W. Spatiotemporal reservoir resampling for real-time ray tracing with dynamic direct lighting. *ACM Transactions on Graphics* Vol. 39, No. 4, Article No. 148, 2020.

[8] Ouyang, Y.; Liu, S.; Kettunen, M.; Pharr, M.; Pantaleoni, J. ReSTIR GI: Path resampling for real-time path tracing. *Computer Graphics Forum* Vol. 40, No. 8, 17–29, 2021.

[9] Lin, D.; Kettunen, M.; Bitterli, B.; Pantaleoni, J.; Yuksel, C.; Wyman, C. Generalized resampled importance sampling: Foundations of ReSTIR. *ACM Transactions on Graphics* Vol. 41, No. 4, Article No. 75, 2022.

[10] Wang, R.; Tran, J.; Luebke, D. All-frequency relighting of glossy objects. *ACM Transactions on Graphics* Vol. 25, No. 2, 293–318, 2006.

[11] Müller, T.; Rousselle, F.; Novák, J.; Keller, A. Real-time neural radiance caching for path tracing. *ACM Transactions on Graphics* Vol. 40, No. 4, Article No. 36, 2021.

[12] Greger, G.; Shirley, P.; Hubbard, P. M.; Greenberg, D. P. The irradiance volume. *IEEE Computer Graphics and Applications* Vol. 18, No. 2, 32–43, 1998.

[13] Nijasure, M.; Pattanaik, S.; Goel, V. Real-time global illumination on GPUs. *Journal of Graphics Tools* Vol. 10, No. 2, 55–71, 2005.

[14] Rodriguez, S.; Leimkühler, T.; Prakash, S.; Wyman, C.; Shirley, P.; Drettakis, G. Glossy probe reprojection for interactive global illumination. *ACM Transactions on Graphics* Vol. 39, No. 6, Article No. 237, 2020.

[15] Majercik, Z.; Marrs, A.; Spjut, J.; McGuire, M. Scaling probe-based real-time dynamic global illumination for production. *arXiv preprint* arXiv:2009.10796, 2020.

[16] Christensen, P. H. Point-based approximate color bleeding. *Pixar Technical Notes* Vol. 2, No. 5, 6, 2008.

[17] Ritschel, T.; Engelhardt, T.; Grosch, T.; Seidel, H. P.; Kautz, J.; Dachsbacher, C. Micro-rendering for scalable, parallel final gathering. In: Proceedings of the ACM SIGGRAPH Asia Papers, Article No. 132, 2009.

[18] Hollander, M.; Ritschel, T.; Eisemann, E.; Boubekeur, T. ManyLoDs: Parallel many-view level-of-detail selection for real-time global illumination. *Computer Graphics Forum* Vol. 30, No. 4, 1233–1240, 2011.

[19] Wang, B.; Meng, X.; Boubekeur, T. Wavelet point-based global illumination. *Computer Graphics Forum* Vol. 34, No. 4, 143–153, 2015.

[20] Kajiya, J. T. The rendering equation. In: Proceedings of the 13th Annual Conference on Computer Graphics and Interactive Techniques, 143–150, 1986.

[21] Yuksel, C. Sample elimination for generating Poisson disk sample sets. *Computer Graphics Forum* Vol. 34, No. 2, 25–32, 2015.

[22] Cook, R. L.; Torrance, K. E. A reflectance model for computer graphics. *ACM SIGGRAPH Computer Graphics* Vol. 15, No. 3, 307–316, 1981.

[23] Walter, B.; Marschner, S. R.; Li, H.; Torrance, K. E. Microfacet models for refraction through rough surfaces. In: Proceedings of the 18th Eurographics Conference on Rendering Techniques, 195–206, 2007.

[24] Trowbridge, T. S.; Reitz, K. P. Average irregularity representation of a rough surface for ray reflection. *JOSA* Vol. 65, No. 5, 531–536, 1975.

[25] Heitz, E. Understanding the masking-shadowing function in microfacet-based BRDFs. *Journal of Computer Graphics Techniques* Vol. 3, No. 2, 32–91, 2014.

[26] Schlick, C. An inexpensive BRDF model for physically-based rendering. *Computer Graphics Forum* Vol. 13, No. 3, 233–246, 1994.

[27] Guy, R.; Agopian, M. Physically based rendering in filament. 2019. Available at https://google.github.io/filament/Filament.html

[28] Benty, N.; Yao, K. H.; Clarberg, P.; Chen, L.; Kallweit, S.; Foley, T.; Oakes, M.; Lavelle, C.; Wyman, C. The Falcor rendering framework. 2022. Available at https://github.com/NVIDIAGameWorks/Falcor

[29] CodeItNow. Cubemap texel solid angle. 2022. Available at https://www.rorydriscoll.com/2012/01/15/cubemap-texel-solid-angle/

Youxin Xing is a doctoral student at Shandong University. He received his bachelor degree from Shandong University of Science and Technology in 2020. His research interests include real-time rendering and game development.

Gaole Pan is a master student at Nanjing University of Science and Technology. He received his bachelor degree from the College of Computer Science and Technology at Harbin Engineering University in 2022. His research interest is mainly in rendering.

Xiang Chen is an undergraduate at Shandong University. His research interest is mainly in real-time rendering.

Ji Wu is a master student at Shandong University. He received his bachelor degree from Southwest Jiaotong University in 2022. His research interest is mainly focused on global illumination.

Lu Wang is a professor at the School of Software, Shandong University. She received her Ph.D. degree from the Department of Computer Science and Technology at Shandong University of China in 2009. Her research interests include photorealistic rendering, real-time rendering, material appearance modeling, and high-performance rendering.

Beibei Wang received her Ph.D. degree from Shandong University in 2014 and visited Telecom ParisTech from 2012 to 2014. She is an associate professor at Nanjing University of Science and Technology (NJUST). She worked as a postdoc with Inria from 2015 to 2017. She joined NJUST in March 2017. Her research interests include rendering and game development.