Real-time All-frequency Global Illumination with Radiance Caching

Youxin Xing*
Shandong University
China
youxinxing@mail.sdu.edu.cn

Gaole Pan*
Nanjing University of Science and Technology
China
pangaole@njust.edu.cn

Xiang Chen
Shandong University
China
xiangchen@mail.sdu.edu.cn

Ji Wu
Shandong University
China
jiwu@mail.sdu.edu.cn

Lu Wang†
Shandong University
China
luwang_hcivr@sdu.edu.cn

Beibei Wang†
Nanjing University of Science and Technology
China
beibei.wang@njust.edu.cn

Abstract

Global Illumination (GI) plays a crucial role in rendering realistic results for the virtual exhibition, *e.g.*, the virtual car exhibition. These scenarios usually include all-frequency bidirectional reflectance distribution functions (BRDFs), although the geometry and the light configuration might be static. Rendering all-frequency BRDFs in real-time is still challenging due to the complex light transport. Existing approaches, including precomputed radiance transfer, light probes, or the most recent path tracing-based approaches (ReSTIR PT), can not satisfy both quality and performance requirements at the same time. In this paper, we propose a practical hybrid global illumination approach, combining ray tracing and cached GI by caching the incoming radiance with wavelets. Our approach can produce close results to offline renderers at the cost of only about 17 ms at runtime and is robust over all-frequency BRDFs. Our approach is designed for applications with static lighting and geometries, like the virtual exhibition.

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, which enhances the renderings' realism. However, the computation of these effects is time-consuming, especially for real-time rendering.

Our method mainly aims at 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 assumption in our paper.

In the real-time rendering domain, precomputed radiance transfer (PRT) [1, 2, 3] and light probes [4, 5] are widely used. The approaches based on PRT support all-frequency shadows [3], glossy reflections, and dynamic lighting at the cost of expensive storage. Most of the light probes-based approaches only focus on diffuse materials. Rodriguez *et al*. [6] can handle glossy interactions, but it isn't suitable for real-time rendering. Majercik *et al*. [7] can provide real-time GI effects on glossy objects by forcing second-order glossy reflections to maximum roughness.

Recently, path tracing combined with advanced sampling strategies and denoising has become a possible solution for real-time global illumination. The advanced sampling strategies include the resampled importance sampling (RIS) [8] for direct illumination [9] or global illumination [10]. Both of them don't work well for low-roughness materials. Recently, Lin *et al*. [11] enabled all-frequency material rendering, thanks to the generalized resampled importance sampling (GRIS), but the results are not noise-free.

This paper aims to achieve real-time global illumination effects with all-frequency shadows and interreflections on glossy objects. For that, we propose a practical hybrid global illumination solution, which combines ray tracing

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

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

Table 1. Notations.

and cached GI. The cached GI is responsible for direct illumination from non-delta light sources (*e.g.*, area lights, environment maps) and the indirect illumination of objects with low-frequency to intermediate-frequency BRDFs from all light sources. The ray tracing handles the indirect illumination of specular or near-specular materials from all light sources and direct illumination from delta light sources (*e.g.*, point lights, directional lights).

In our cached GI approach, we use wavelets to represent the incoming radiance and BRDFs at a precomputation step and perform an efficient convolution during rendering. In particular, we cache irradiance for the diffuse materials. In the end, our method is able to provide high-quality results, which match the references, with only about 17 ms.

2. Previous work

Precomputed radiance transfer (PRT). Sloan *et al*. [1] use the spherical harmonic (SH) basis to restore soft shadows, reflections, and caustic effects. Ng *et al*. [3] replace SH with wavelets, so their method can represent all-frequency shadows and reflections. Wang *et al*. [12] introduce PRT and 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 into real-time rendering to sample direct illumination [9], low-frequency GI [10], and high-frequency GI [11]. The recent generalized form of RIS (GRIS) allows for glossy indirect illumination. Müller *et al*. [13] further introduce the neural radiance caching into real-time path tracing to accelerate rendering by learning and rendering the radiance distribution online with a neural network.

Light probes. The irradiance volume [14, 15] or light probes-based approaches have been widely used in the video game industry due to their efficiency. They subdivide scenes into discretized points, represent the irradiance distribution at these points during the precomputation, and query the light probes during rendering. Majercik *et al*. [5] proposed a dynamic solution for diffuse global illumination (DDGI). They update both the irradiance and visibility with ray tracing at runtime. Most of these works only focus on diffuse materials, except one of the recent studies by Rodriguez *et al*. [6], which allows for glossy interreflections. Unfortunately, it's time-consuming and not suitable for real-time rendering. Majercik *et al*. [7] have also extended the DDGI to glossy materials, but they force second-order glossy reflections to maximum roughness leading to unfaithful 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 a geometric proxy of the geometry, so they can be rasterized to solve the visibility. PBGI has also been extended to real-time rendering [17, 18], but they focus on diffuse global illumination. They can not handle glossy interreflections and caustics since the radiance is represented with SH. Later, Wang *et al*. [19] replace SH with wavelets, allowing for non-diffuse light transport, but their work is too heavy for real-time rendering.

3. Background and overview

We first introduce the rendering equation and PRT in this section. Then we give an overview of our approach.

3.1. Rendering equation

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

$$L_o(\mathbf{x}, \boldsymbol{\omega_o}) = \int_\Omega L_i(\boldsymbol{\omega_i}) f_r(\mathbf{x}, \boldsymbol{\omega_i}, \boldsymbol{\omega_o})(\mathbf{n} \cdot \boldsymbol{\omega_i}) \mathrm{d}\boldsymbol{\omega_i}, \quad (1)$$

where $\boldsymbol{\omega_i}$ is the incident direction, $\boldsymbol{\omega_o}$ is the outgoing direction (also called a view direction), and $\mathbf{n}$ is the surface normal. L_o is the outgoing radiance at a position $\mathbf{x}$ from the view direction $\boldsymbol{\omega_o}$, L_i is the lighting and f_r is the BRDF.

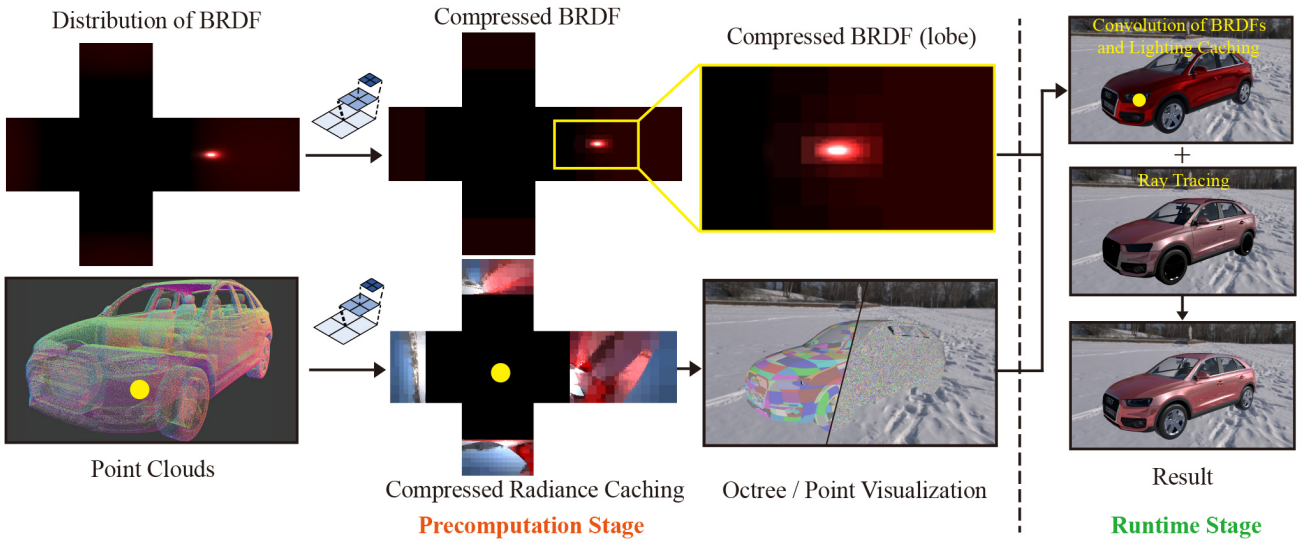


Figure 1. Overview of our hybrid global illumination method on a Vehicle scene. The precomputation is shown on the left, and the runtime rendering is on the right. In the precomputation stage, each BRDF is precomputed as a half cube map and compressed with Haar wavelets (left top). The scene is discretized 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 get 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, then combine with the cached GI to get the final rendered results.

3.2. Precomputed radiance transfer

The rendering equation can be reformulated to achieve real-time rendering by caching a part of its composition, like PRT [1, 2, 3].

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

$$T(\mathbf{x}, \boldsymbol{\omega_i}, \boldsymbol{\omega_o}) = f_r(\mathbf{x}, \boldsymbol{\omega_i}, \boldsymbol{\omega_o}) V(\mathbf{x}, \boldsymbol{\omega_i})(\mathbf{n} \cdot \boldsymbol{\omega_i}). \quad (2)$$

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

$$L_o(\mathbf{x}, \boldsymbol{\omega_o}) = \int_\Omega L_i(\boldsymbol{\omega_i}) T(\mathbf{x}, \boldsymbol{\omega_i}, \boldsymbol{\omega_o}) \mathrm{d}\boldsymbol{\omega_i}. \quad (3)$$

To achieve real-time frame rates, L_i and T are precomputed and represented with the appropriate orthonormal basis function $\Psi_j(\boldsymbol{\omega_i})$:

$$L_i(\boldsymbol{\omega_i}) = \sum_j l_j \Psi_j(\boldsymbol{\omega_i}), \quad (4)$$

$$T(\mathbf{x}, \boldsymbol{\omega_i}, \boldsymbol{\omega_o}) = \sum_k t_k \Psi_k(\boldsymbol{\omega_i}), \quad (5)$$

resulting in the following final formation:

$$\begin{aligned} L_o(\mathbf{x}, \boldsymbol{\omega_o}) &= \int_\Omega (\sum_j l_j \Psi_j(\boldsymbol{\omega_i}))(\sum_k t_k \Psi_k(\boldsymbol{\omega_i})) \mathrm{d}\boldsymbol{\omega_i} \\ &= \sum_j l_j t_j = \mathbf{L} \cdot \mathbf{T}. \end{aligned} \quad (6)$$

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

Compared with PRT, our method can compute and store the radiance by highly compressed wavelet coefficients at each point. Aided by the octree structure for query acceleration and the 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 the efficient convolution of these two components during rendering. For the specular or near-specular effects and the direct illumination of delta light sources that are too sharp for the cached GI, we compute them 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. Then we 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 perform a product of the wavelet coefficients for the lighting and the BRDF, as shown in Figure 1.

4. BRDFs and Lighting precomputation

In the precomputation step, each mesh is discretized into a point cloud (Section 4.1). Then, we precompute all the 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 clouds generation

For each mesh in the scene, we generate a point cloud with Poisson disk sampling [21]. Given a desired point count M, we first generate more points with random sampling, where the number of the points for each mesh triangle is with respect to its area. In practice, we generate $5 \times M$ random-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 decide which sample point to eliminate, we measure the closeness of each point to its neighboring points with *weight*. W_{ij} is the *weight* between sample point i and j $(i \neq j)$:

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

where d_{ij} is the distance between sample point i and j. $r_{\max}$ is the maximum radius, set as $\sqrt{A_2/(2\sqrt{3}n)}$, where A_2 is an area of the sampling area [21]. n is a desired number of samples after elimination, and α is an exponential constant used to control the *weight*, set as 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 from sample point i.

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

4.2. BRDFs precomputation and compression

In our paper, we focus on three types of materials: diffuse, glossy, and clear coat BRDFs.

As for the diffuse, we use the Lambertian diffuse material:

$$f_d(\boldsymbol{\omega_i}, \boldsymbol{\omega_o}) = \frac{1}{\pi} c_{\text{base}}, \tag{8}$$

where c_{base} is the diffuse color.

Then, we use a Cook-Torrance model [22] for the glossy material, which includes both a diffuse term f_d defined by Eq. (8) and a 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 is a typical microfacet model [23]:

$$f_s(\boldsymbol{\omega_i}, \boldsymbol{\omega_o}) = \frac{D(h)F(\boldsymbol{\omega_o}, h)G(\boldsymbol{\omega_i}, \boldsymbol{\omega_o}, h)}{4(\mathbf{n} \cdot \boldsymbol{\omega_i})(\mathbf{n} \cdot \boldsymbol{\omega_o})}, \tag{10}$$

where D is the normal distribution function (NDF) [24], F is a 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.

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

$$\begin{aligned} f_{\text{clearcoat}}(\boldsymbol{\omega_i}, \boldsymbol{\omega_o}) = f_g(\boldsymbol{\omega_i}, \boldsymbol{\omega_o})(1 - F_c) \\ + f_c(\boldsymbol{\omega_i}, \boldsymbol{\omega_o}), \end{aligned} \tag{11}$$

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 in the material, which controls the strength of the clear coat effect.

Precomputation. The diffuse material is computed at runtime, which is shown in Section 5.2. For each glossy material, we precompute the distribution of the BRDF. We sample the outgoing direction $\boldsymbol{\omega_o}$ uniformly by using the equal cosine sampling in a hemisphere space. Then, for each sampled $\boldsymbol{\omega_o}$, we represent the distribution of the incoming direction $\boldsymbol{\omega_i}$ with 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. Finally, we compute the BRDF value according to each $\boldsymbol{\omega_o}$ and $\boldsymbol{\omega_i}$ at each pixel of the half cube map. The resolution of the cube map is set to 128×128 in practice.

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 fashion. 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 with the mother scaling coefficients of its child nodes, defined by c_j $(j = 0, 1, 2, 3)$:

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

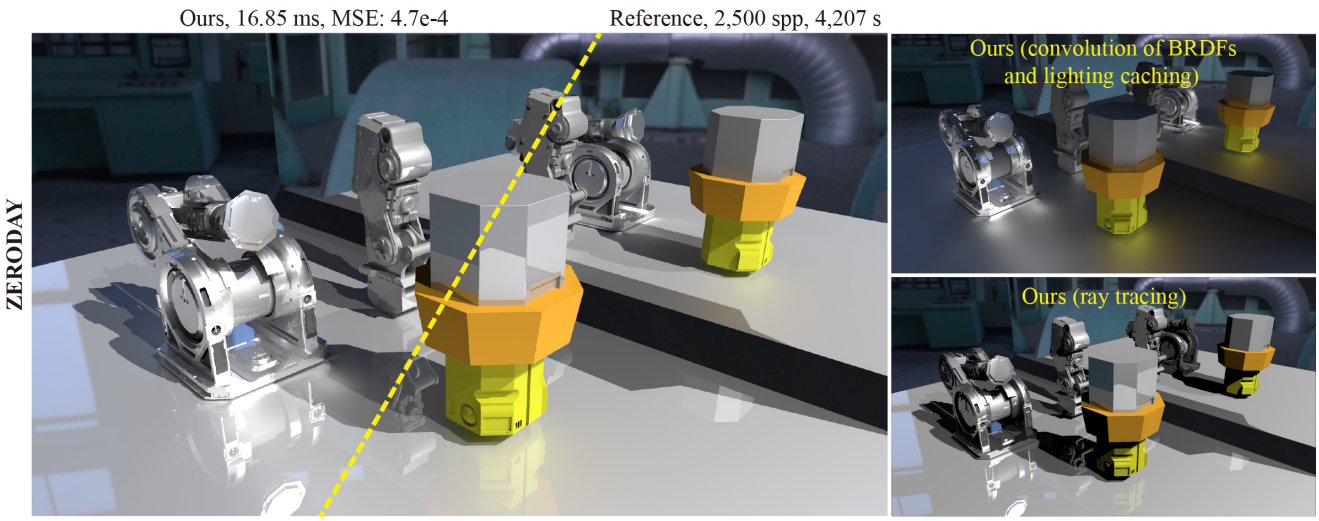


Figure 2. Results of ZeroDay using our method. The comparison between our final result and the reference is shown on the left. The result of the convolution of BRDFs and lighting caching is shown on the right (top). The ray tracing result is shown on the right (bottom).

Table 2. Resolution of radiance cube map under different roughness.

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

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

4.3. Lighting caching and octree construction

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

Lighting caching. Object with different material type is treated differently. We store the irradiance rather than incoming radiance for the object with the diffuse material since it is view-independent:

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

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

Different from the diffuse material, we precompute the incoming radiance for the object with the glossy material, regardless of whether it has a clear coat or not. During caching, we locate a half cube map around each point and compute the incoming radiance for the half cube map with 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, which is shown in Table 2. Then we compress each half cube map with wavelets in a quadtree form, the same as the BRDFs. The discard threshold β is set to 1,000.

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

We construct the octree in an up-bottom fashion. Starting from the axis-aligned bounding box (AABB) of the whole mesh, each node is subdivided uniformly into eight child nodes according to the AABB. This subdivision for each node continues when the point count at each leaf node is larger than a certain threshold (set as 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 the normal. The leaf node's bounding box is computed by the bounding box of its points, while the non-leaf node's bounding box is computed by the union of the bounding boxes of its child nodes. The normal of each node is set as an average of the normals in its points or child nodes.

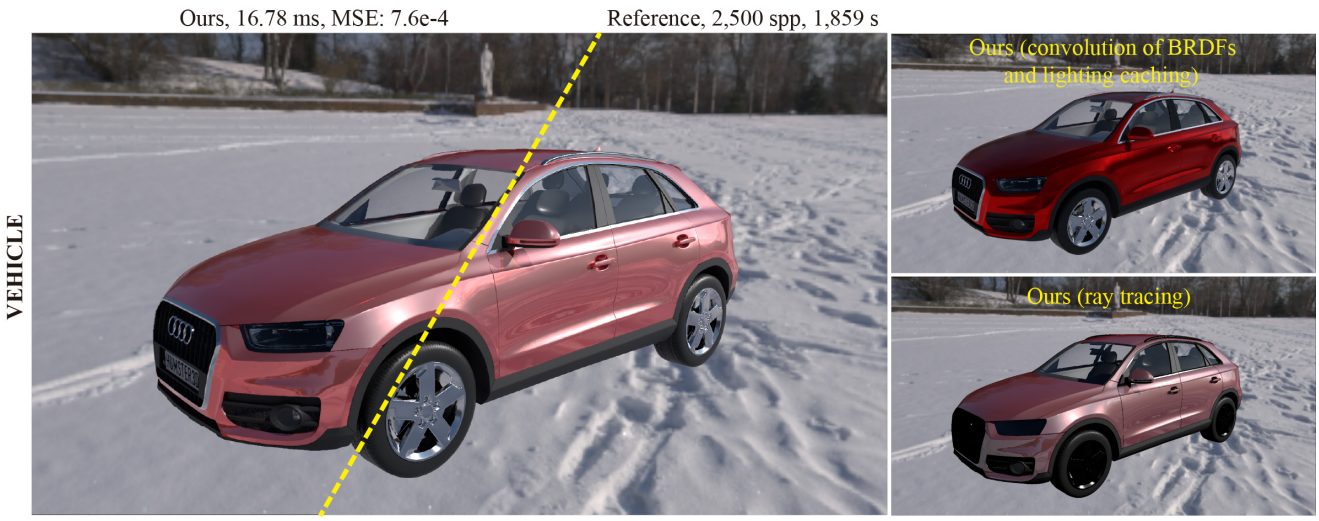


Figure 3. Results of Vehicle using our method. The comparison between our final result and the reference is shown on the left. The result of the convolution of BRDFs and lighting caching is shown on the right (top). The ray tracing result is shown on the right (bottom).

5. Real-time global illumination

During runtime, we compute the result of ray tracing first. Then we search and convolute the BRDFs and lighting caching. At last, we merge them to get the final real-time global illumination.

5.1. Ray tracing

As for ray tracing, we compute the direct illumination of delta lights and clear coat effects. And we also trace rays to compute the indirect illumination for low-frequency materials. For diffuse materials, the equation is as follows:

$$L_o(\mathbf{x}, \boldsymbol{\omega_o}) = f_d(\boldsymbol{\omega_i}, \boldsymbol{\omega_o}) L_i(\mathbf{x}, \boldsymbol{\omega_i})(\boldsymbol{\omega_i} \cdot \mathbf{n}). \tag{15}$$

For glossy materials, the equation is as follows:

$$L_o(\mathbf{x}, \boldsymbol{\omega_o}) = f_g(\boldsymbol{\omega_i}, \boldsymbol{\omega_o}) L_i(\mathbf{x}, \boldsymbol{\omega_i})(\boldsymbol{\omega_i} \cdot \mathbf{n}). \tag{16}$$

For glossy materials with clear coat effects, the equation is as follows:

$$\begin{aligned} L_o(\mathbf{x}, \boldsymbol{\omega_o}) = {} & f_g(\boldsymbol{\omega_i}, \boldsymbol{\omega_o}) L_i(\mathbf{x}, \boldsymbol{\omega_i})(\boldsymbol{\omega_i} \cdot \mathbf{n})(1 - F_c) \\ & + f_c(\boldsymbol{\omega_i}, \boldsymbol{\omega_o}) L_i(\mathbf{x}, \boldsymbol{\omega_i})(\boldsymbol{\omega_i} \cdot \mathbf{n}). \end{aligned} \tag{17}$$

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

$$L_o(\mathbf{x}, \boldsymbol{\omega_o}) = F(\boldsymbol{\omega_o}, h) L_{\text{env}}(\mathbf{x}, \boldsymbol{\omega_i}), \tag{18}$$

where F is a 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 the convolution of them in the quadtree form. The BRDFs coefficients are queried through the material index (ID) and the view direction $\boldsymbol{\omega_o}$ in the hemispherical space. The lighting caching is queried by three steps. First, we find the octree by the mesh ID of the shading point. Then we search for leaf nodes that contain the target point in the octree by a distance threshold. Finally, we select the most appropriate point for the shading point in the leaf nodes according to a mixed weight of the position and normal direction.

After searching, for diffuse materials, we compute the result by the diffuse color c_{base} and irradiance $L_{\text{irradiance}}$.

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

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

$$L_o(\mathbf{x}, \boldsymbol{\omega_o}) = \mathbf{C}_{\text{BRDF}} \otimes \mathbf{C}_{\text{radiance}}. \tag{20}$$

As for the glossy material with a clear coat, we additionally multiply by the ratio $(1 - F_c)$ as follows:

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

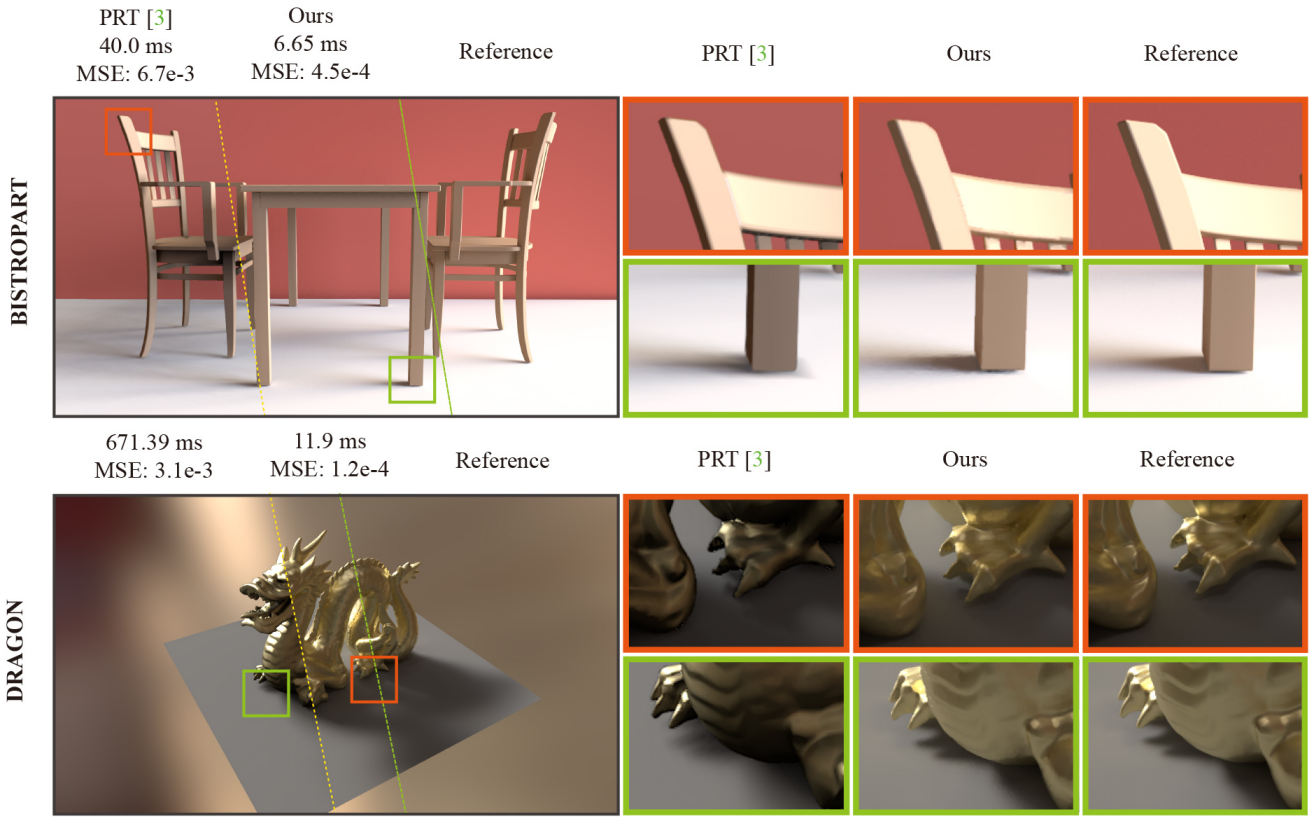


Figure 4. Comparison between PRT [3] and our method on the BistroPart (top) and Dragon (bottom) scenes.

The clear coat term is computed by ray tracing.

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

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 dispatch the tasks to 16,384 GPU threads at once for GPU acceleration. And the lighting caching tasks are dispatched to 8,192 GPU threads in practice. We have implemented our method on 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 categorize the BRDFs with similar properties (diffuse color, etc.) into groups and then perform BRDF precomputation for each group by treating it as a single BRDF. Note that the indices of these groups are stored in the alpha channel of the BRDF texture.

When computing the half cube maps for BRDFs, for a ω_o, we apply stratified sampling to ω_i in the half cube map to avoid the stripe artifacts in the rendering results. During compression, we store the coefficients of BRDFs and cached radiance in the half data type to reduce the storage size, which is sufficient to obtain the same rendering results as the float type.

Lighting caching and octree construction. While computing the lighting, we divide the radiance in each radiance half cube map by the solid angle in the corresponding direction [29] for energy conservation. After computation, as for diffuse materials, we add up all the incident radiance stored in the half cube map to get the irradiance for each point.

We construct an octree for each mesh to improve the query accuracy for the cached points at the intersection of meshes. The reflections and shadows of objects with diffuse materials or high-roughness glossy materials are view-independent. Thus, on the objects with low-frequency reflections and shadows, we compute the result with sparse points for lighting caching, which has a similar result with the dense points (*e.g.*, the inner shadow on the floor in Figure 4 bottom). In practice, during construction, we remove the points that have minor energy differences with neighboring points by the probability based on the statistics of the

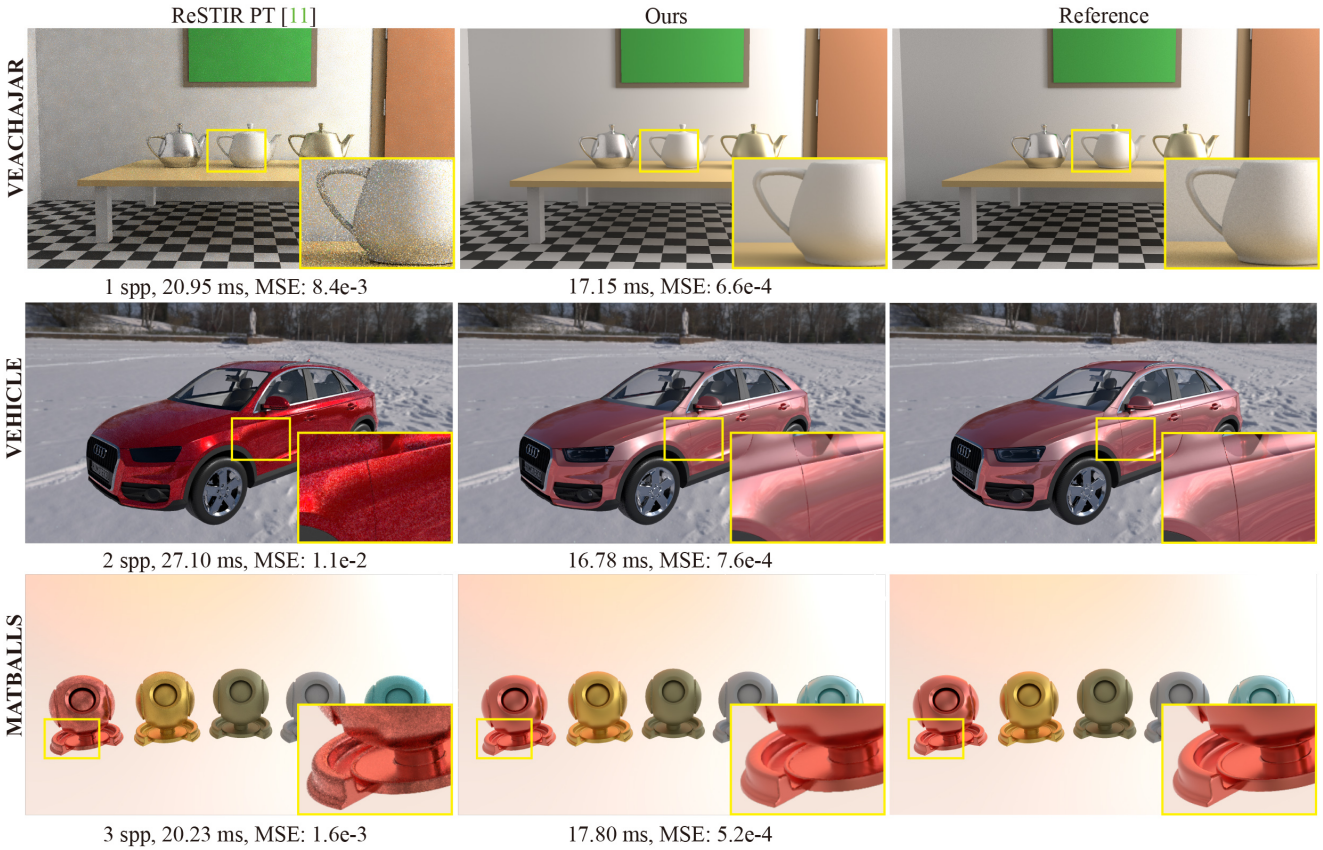


Figure 5. Comparison between ReSTIR PT [11] and our method on the scenes including VeachAjar (top), Vehicle (middle), Matballs (bottom).

average energy differences in the whole point cloud. The points elimination rate is about 47%.

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

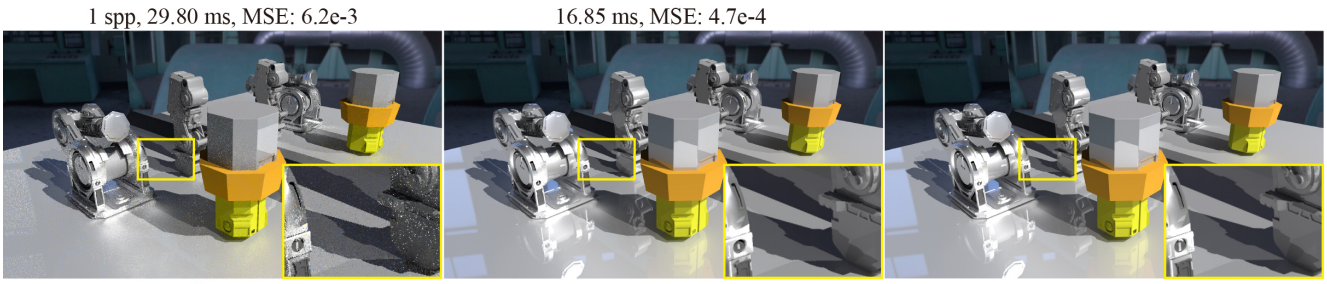
While searching BRDFs, first, we get four BRDFs coefficient quadtrees from the four neighboring directions of the view direction ω_o. Then we perform the bilinear interpolation on the four quadtrees to reduce the discontinuity of the rendering results and the storage of BRDFs coefficients while using a sparse ω_o sampling density. After the interpolation, the new quadtree's depth is the same as the lowest one in the four quadtree depths.

When querying the lighting caching in an octree, we control the maximum number of searching points in all searched nodes by a parameter m to improve the search speed. Besides, we also discard the point whose energy is lower than a set threshold. Because it 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 control the traversal layers of quadtrees to balance performance and quality. Besides, we reduce the computation and offer a way to simulate the effect of the high-roughness glossy materials by using a low quadtree depth. However, it inevitably brings some artifacts. Therefore, we make a tradeoff in practice.

7. Results

We first compare our global illumination method with PRT [3], ReSTIR PT [11], and DDGI [5]. Next, we show the performance measures of our method in different test scenes and the parameter analysis in the Dragon scene. We implement ReSTIR PT from the released code and reimplement PRT by us. References are computed by standard path tracing with multiple samples per pixel (spp). We quantify the error by the mean squared error (MSE). All the results



(a) ReSTIR PT [11] (b) Ours (c) Reference

Figure 6. Compared to ReSTIR PT [11] (a) on the ZeroDay scene, our approach (b) is noise-free and has similar quality with reference (c), in about 17 ms per frame.

Table 3. Performance measures of our method in different test scenes. The precomputed storage includes the caching of BRDFs and lighting.

Scenes	Points	Materials	Precomputed time (h)		Precomputed storage (GB)		Runtime memory cost		Runtime
			BRDFs	Lighting	BRDFs	Lighting	Main (GB)	GPU (GB)	time cost (ms)
Dragon	81,579	2	2	4	2.55	0.57	3.8	4.4	11.9
BistroPart	400,000	3	0	10	0	0.02	0.678	1.01	6.65
VeachAjar	1,317,000	11	6	31	6.73	6.43	11.4	13.54	17.15
Vehicle	600,000	14	5	30	3.38	15.9	19.55	20.3	16.78
ZeroDay	692,000	6	2.5	20	4.39	1.61	6.66	7.99	16.85
MatBalls	500,000	5	2.5	13.5	6.28	7.75	14.71	15.2	17.8

are 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 (Figure 4 top) contains three diffuse materials and an environment map. In this scene, we have cached the irradiance at 400,000 points only. PRT has cached the visibility distribution at 34,325 vertexes, the lighting distribution from the environment map, and the BRDFs distribution of three materials. Compared with the 1.61 GB caching storage of PRT, our method has an advantage in storage which is only 0.02 GB. And it is almost 6 times faster than PRT at runtime.

In the Dragon scene (Figure 4 bottom), there are two glossy materials and an environment map. The caching of our method consists of the radiance distribution at 81,579 points and the BRDFs distribution of two glossy materials. The caching of PRT includes the visibility distribution at 100,277 vertexes, the lighting distribution from the environment map, and the BRDFs distribution of two materials. Our method's caching size is 3.12 GB which is larger than PRT's (2.47 GB) because the BRDFs distribution of glossy materials takes up most of the caching storage. At runtime, the performance of our method is about 56 times faster than PRT.

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

Comparison with ReSTIR PT. We compare the results of our method with ReSTIR PT in the VeachAjar, Vehicle, MatBalls, and ZeroDay scenes, as shown in Figure 5 and Figure 6. All the materials in the scenes are glossy. Besides, the Vehicle (car shell), Matballs (cyan ball), and ZeroDay (floor and canons) scenes have the clear coat effect.

Different from ReSTIR PT, the results of our method are basically noise-free by the benefit of ray tracing and the convolution of BRDFs and lighting caching without sampling. Our results have no color bias. The results of ReSTIR PT have an obvious color bias, especially on objects with a clear coat and low-frequency BRDFs (*e.g.*, the car shell). Because it is limited to the shift mapping strategies of the path reuse. At runtime, the time cost of our method is less than ReSTIR PT with 1-3 spp in the test scenes.

Comparison with DDGI. We compare the indirect illumination results of our method with DDGI [5] in the Table scene, as shown in Figure 7. The Table scene has two diffuse materials and an environment map. We precompute the irradiance at 400,000 points, and the caching of DDGI contains $25 \times 25 \times 25$ light probes. With the equal time cost

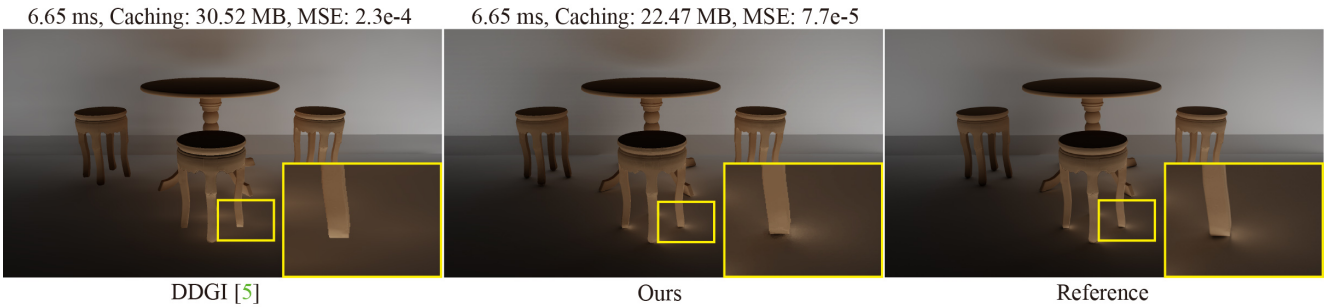


Figure 7. Comparison of the indirect illumination between DDGI [5] and our method on the Table scene.

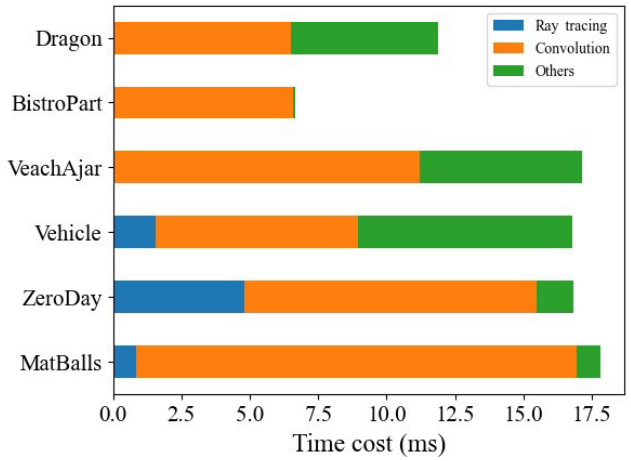


Figure 8. Time cost during runtime using our method. Note that we compute global illumination with irradiance instead of the convolution of BRDFs and lighting caching in the BisroPart scene.

at runtime, our method has less caching storage (22.47 MB) than DDGI (30.52 MB). Besides, our results preserve richer indirect lighting details than DDGI during rendering.

7.2. Performance and storage

We show the performance measures of our method in Table 3 for different test scenes. Our method takes up about 17 ms per frame at runtime in all test scenes, with almost no quality loss compared with the reference. But our method requires up to several hours of precomputation time. Fortunately, as long as the scene does not change (with static objects and lighting), the precomputation only needs to do once.

Table 3 also shows the storage of cached information and runtime memory of our method. In precomputation stage, the storage of lighting caching is influenced by the geometry complexity of the meshes, which directly determines the points number. Besides, it is also affected by the complexity of the lighting distribution at the points. Such as the VeachAjar scene, both the geometry and light distribution are complex. Thus a large number of points are needed to store the lighting information. The storage of BRDFs caching is mainly determined by the number of glossy materials and the distribution complexity of the BRDFs, which is related to their roughness. The scene with diffuse materials (*e.g.*, the BistoPart scene) has less storage than the scene with glossy materials (*e.g.*, the Dragon scene). The main memory and GPU memory at runtime depend mainly on the total storage of BRDFs and lighting caching.

Figure 8 shows our method's time cost of ray tracing, convolution, and others. The others include V-Buffer computing, primary ray shooting, intersecting, etc. The VeachAjar scene only contains an area light source behind the door. The other scenes all have an environment map. For the Vehicle and ZeroDay scenes, we additionally provide the delta lights. In Figure 8, the scenes of Dragon, BistroPart, and VeachAjar have no time cost of ray tracing. Because we compute the global illumination for non-delta lights with BRDFs and lighting caching instead of ray tracing. When computing the convolution, the 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 takes up the largest time cost.

7.3. Parameter analysis

Varying number of points. We generate point clouds with different points number for the Dragon scene and show

Table 4. Performance measures of our method with different points number in the Dragon scene.

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

the performance measures of our method in Table 4. With more points, the caching storage of lighting increases significantly, leading to more time cost during rendering. However, the quality of our results has also been improved, while the MSE between our results and the references gets lower.

Varying roughness. In Figure 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 effect) in the MatBalls scene. Our method obtains the same results as the offline renderer in a wide roughness range with the 17.8 ms time cost at runtime.

7.4. Limitations and discussions

We recognize two limitations. First, to simplify the query of BRDFs, we discretize the ω_o in the hemispherical space and compute the distribution of BRDFs with ω_i. This inevitably leads to a lot of storage. We consider organizing ω_o and ω_i from a four-dimensional perspective to obtain a more compact representation. Second, for BRDFs defined with textures, the excessive number of colors in the texture map results in a huge number of BRDFs, which lead to large caching storage.

8. Conclusion and future work

In this paper, we have presented a new hybrid real-time global illumination method that combines ray tracing and the convolution of BRDFs and lighting caching. During the precomputation, we offer a point clouds generator to compute the points that conform to the Poisson distribution, a new wavelet compression structure in the quadtree form for BRDFs and cached radiance, and a compact spatial hierarchy for lighting caching. At runtime, our method has results close to 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. As for scenes with diffuse materials, our method is almost 6 times faster than PRT and has less caching storage at runtime. It is suitable for applications that allow static lighting and geometric scenes, primarily virtual exhibitions.

In the future, we will combine deep learning to provide a more compact representation and avoid the interpolation for lighting and BRDFs. Furthermore, our approach might be a good proxy for path guiding.

Acknowledgements

We thank the reviewers for their valuable suggestions. This work has been partially supported by the National Key R&D Program of China under grant No.2020YFB1709203, the National Natural Science Foundation of China under grant No.62272275 and 62172220, the Shandong Provincial Natural Science Foundation of China under grant No.ZR2020LZH016. This work is also partially supported by funds from Huawei Cloud Computing Technologies Co., Ltd.

Compliance with ethical standards

The authors have no competing interests to declare that are relevant to the content of this article. This study does not contain any studies with human or animal subjects performed by any of the authors.

References

[1] P. P. Sloan. Precomputed radiance transfer for real-time rendering in dynamic, low-frequency lighting environments. *Proc. of SIGGRAPH 2002*, 2002. 1, 2, 3

[2] R. Ng, R. Ramamoorthi, and P. Hanrahan. All-frequency shadows using non-linear wavelet lighting approximation. In *ACM SIGGRAPH 2003 Papers*, pages 376–381. 2003. 1, 3

[3] R. Ng, R. Ramamoorthi, and P. Hanrahan. Triple product wavelet integrals for all-frequency relighting. In *ACM SIGGRAPH 2004 Papers*, pages 477–487. 2004. 1, 2, 3, 7, 8

[4] J. Kontkanen and S. Laine. Sampling precomputed volumetric lighting. *Journal of Graphics Tools*, 11(3):1–16, 2006. 1

[5] Z. Majercik, J.-P. Guertin, D. Nowrouzezahrai, and M. McGuire. Dynamic diffuse global illumination with ray-traced irradiance fields. *Journal of Computer Graphics Techniques (JCGT)*, 8(2):1–30, June 2019. 1, 2, 8, 9, 10

[6] S. Rodriguez, T. Leimkühler, S. Prakash, C. Wyman, P. Shirley, and G. Drettakis. Glossy probe reprojection for interactive global illumination. *ACM Transactions on Graphics (TOG)*, 39(6):1–16, 2020. 1, 2

[7] Z. Majercik, A. Marrs, J. Spjut, and M. McGuire. Scaling probe-based real-time dynamic global illumination for production. *arXiv preprint arXiv:2009.10796*, 2020. 1, 2

[8] J. Talbot, D. Cline, and P. K. Egbert. Importance resampling for global illumination. In *Eurographics Symposium on Rendering*, 2005. 1

[9] B. Bitterli, C. Wyman, M. Pharr, P. Shirley, A. Lefohn, and W. Jarosz. Spatiotemporal reservoir resampling for real-time ray tracing with dynamic direct lighting. *ACM Transactions on Graphics (TOG)*, 39(4):148–1, 2020. 1, 2

[10] Y. Ouyang, S. Liu, M. Kettunen, M. Pharr, and J. Pantaleoni. ReSTIR GI: Path Resampling for Real-Time Path Tracing. *Computer Graphics Forum*, 2021. 1, 2

[11] D. Lin, M. Kettunen, B. Bitterli, J. Pantaleoni, C. Yuksel, and C. Wyman. Generalized resampled importance sampling: foundations of restir. *ACM Transactions on Graphics (TOG)*, 41(4):1–23, 2022. 1, 2, 8, 9

[12] R. Wang, J. Tran, and D. Luebke. All-frequency relighting of glossy objects. *ACM Transactions on Graphics (TOG)*, 25(2):293–318, 2006. 2

[13] T. Müller, F. Rousselle, J. Novák, and A. Keller. Real-time neural radiance caching for path tracing. *ACM Trans. Graph.*, 40(4):36:1–36:16, Aug. 2021. 2

[14] G. Greger, P. Shirley, P. M. Hubbard, and D. P. Greenberg. The irradiance volume. *IEEE Computer Graphics and Applications*, 18(2):32–43, 1998. 2

[15] M. Nijasure, S. Pattanaik, and V. Goel. Real-time global illumination on gpus. *Journal of Graphics Tools*, 10(2):55–71, 2005. 2

[16] P. Christensen. Point-based approximate color bleeding. *Pixar Technical Notes*, 2(5):6, 2008. 2

[17] T. Ritschel, T. Engelhardt, T. Grosch, H.-P. Seidel, J. Kautz, and C. Dachsbacher. Micro-rendering for scalable, parallel final gathering. *ACM Trans. Graph. (Proc. SIGGRAPH Asia)*, 28(5), 2009. 2

[18] M. Hollander, T. Ritschel, E. Eisemann, and T. Boubekeur. Manylods: Parallel many-view level-of-detail selection for real-time global illumination. *Computer Graphics Forum*, 30(4):1233–1240, 2011. 2

[19] B. Wang, X. Meng, and T. Boubekeur. Wavelet point-based global illumination. *Computer Graphics Forum*, 34(4):143–153, 2015. 2

[20] Kajiya and T. James. The rendering equation. *Acm Computer Graphics*, 20(4):143–150, 1986. 2

[21] C. Yuksel. Sample elimination for generating poisson disk sample sets. *Computer Graphics Forum (Proceedings of EUROGRAPHICS 2015)*, 34(2):25–32, 2015. 4

[22] R. L. Cook and K. E. Torrance. A reflectance model for computer graphics. *SIGGRAPH Comput. Graph.*, 15(3):307–316, aug 1981. 4

[23] B. Walter, S. R. Marschner, H. Li, and K. E. Torrance. Microfacet models for refraction through rough surfaces. *Rendering techniques*, 2007:18th, 2007. 4

[24] T. Trowbridge and K. P. Reitz. Average irregularity representation of a rough surface for ray reflection. *JOSA*, 65(5):531–536, 1975. 4

[25] E. Heitz. Understanding the masking-shadowing function in microfacet-based brdfs. *Journal of Computer Graphics Techniques*, 3(2):32–91, 2014. 4

[26] C. Schlick. An inexpensive brdf model for physically-based rendering. *Computer Graphics Forum*, 13(3):233–246, 1994. 4

[27] M. A. Romain Guy. Physically based rendering in filament. https://google.github.io/filament/Filament.html, Feb. 2019. 4

[28] N. Benty, K.-H. Yao, P. Clarberg, L. Chen, S. Kallweit, T. Foley, M. Oakes, C. Lavelle, and C. Wyman. The Falcor rendering framework, 08 2020. https://github.com/NVIDIAGameWorks/Falcor. 7

[29] Cubemap texel solid angle. https://www.rorydriscoll.com/2012/01/15/cubemap-texel-solid-angle/. Accessed: 2022-8-25. 7