IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received April 17, 2020, accepted May 11, 2020, date of publication May 14, 2020, date of current version May 29, 2020.

*Digital Object Identifier 10.1109/ACCESS.2020.2994597*

# An Approach to Global Illumination Calculation Based on Hybrid Cone Tracing

**TAO LIU[1,2,5], JIN GAO[3], AND ZHENGLING LEI[4]**

[1]College of Transport and Communications, Shanghai Maritime University, Shanghai 201306, China
[2]Hubei Key Laboratory of Inland Shipping Technology, Wuhan 430063, China
[3]China Waterborne Transport Research Institute, Beijing 100088, China
[4]College of Engineering Science and Technology, Shanghai Ocean University, Shanghai 201306, China
[5]School of Data Science and Technology, North University of China, Taiyuan 030051, China

Corresponding author: Zhengling Lei (zllei@shou.edu.cn)

This work was supported in part by the National Natural Science Foundation of China under Grant 61602426, Grant 61672473, and Grant 61702147, in part by the China Postdoctoral Science Foundation under Grant 2017M621932, in part by the Open Subject of the State Key Laboratory of Engines (Tianjin University) under Grant K2019-14, and in part by the Fund of the Hubei Key Laboratory of Inland Shipping Technology under Grant NHHY2019001.

**ABSTRACT** For 3D geographic information systems (GIS) or video game systems, global illumination (GI) effect can greatly improve the understanding of the volumetric structure and the spatial relationships of objects in the scene. However, GI effect is computationally expensive. It requires not only complex visibility computations between arbitrary points but also the integration computation over a large number of directions. These two computations become extremely difficult tasks for the real-time or interactive algorithms of the dynamic scene. This paper proposes a lightweight GI calculation approach built upon a hybrid cone tracing algorithm with which to approximate the GI effect of the open world scene in real time. First, the 3D scene is divided into two types: complex meshes and height-field meshes. Second, the corresponding lightmaps of the two meshes are generated and mipmaped, respectively. GPU hardware acceleration technology is used to do the calculation efficiently. Finally, a hybrid cone tracing method is performed on the GPU to gather the indirect lighting information for each shaded point. The method first carries out the cone-scene intersection test and then carries out the light sampling and accumulation calculation. All tracing calculations are based on the hybrid lightmap representation. In addition, the experiment results show the effectiveness of our method. Compared with the 3D texture VCT method, memory consumption can be reduced by up to 80% in our experiment.

**INDEX TERMS** Global illumination, cone tracing, real-time rendering, hybrid lightmap representation.

## I. INTRODUCTION

GI effects can greatly improve the understanding of the volumetric structure and the spatial relationships of objects in the scene [1], and have been widely investigated since the earliest days of graphics research. For modern 3D GIS or video game systems, there is a growing need to offer a realistic environment, where the GI effect plays an important role. During these applications, almost all operations need the user to interact with the 3D environment, i.e., move objects around, construct buildings, or change the lighting dynamically. This is where dynamic GI comes into play. Compared with static pre-computed algorithms, dynamic GI algorithm does account for highly dynamic environments. Therefore, there is a high interest in developing real-time dynamic GI algorithms.

Lighting algorithms that additionally describe the light at a certain surface point as a function of other geometries in the scene are hence called GI algorithms. GI algorithm can drastically improve the realism of a rendered scene. However, GI effect is computationally expensive. In order to generate plausible results in various applications, including movies, simulations, computer-aided design and computer-aided manufacturing, and video games, a lot of GI algorithms have been proposed. Such algorithms can be classified as either realistic algorithm for a physically plausible result or real-time algorithm for a fast approximation result.

Complete realistic rendering algorithms are not yet practical to compute in real-time for dynamic scenes. For example, path tracing methods require a lot of computational

The associate editor coordinating the review of this manuscript and approving it for publication was Nuno Garcia.

consumption. They usually take a few minutes or even hours to generate ground truth for each frame. With the introduction of new level programmable commodity GPU hardware in 2002, there is an exploration in the number of real-time algorithms [2]. Real-time algorithms which take as their target an art director's 'vision', rather than a particular subset of the physics of light. The representative method is the pre-computed radiance transfer (PRT) technology, which uses the cached light transport to allow new views and/or re-lighting of a 3D scene to be generated in real time. This kind of method is proposed to balance the computational load between run-time and pre-processing costs. However, the PRT method needs a long pre-computation time, and thus is not suitable for the interactive re-lighting of dynamic scenes. To address the limitations, Crassin *et al.* [3] proposed a voxel cone tracing algorithm (VCT) to compute GI effect in real time, which avoids costly pre-processing steps. The VCT algorithm is based on a hierarchical voxel octree representation. It accelerates the integral calculation of the rendering equation [4] by means of pre-filtering, and is a relatively high-quality method for real-time dynamic illumination. However, although this method results in plausible effects, maintaining such hierarchical voxel representation can limit the size of the scene.

In this paper, we propose a hybrid cone tracing algorithm with which to approximate the GI effect of the open world scene in real time. It not only maintains the advantage of VCT method but also can adapt to more 3D scenes. First, we introduce the theoretical foundation for the cone tracing method and our lightmap computation method based on the classification of meshes. The lightmap computation forms the basis for subsequent calculations. Second, we design a hybrid cone tracing method for use in combination with our hybrid lightmap representation, which can significantly reduce the number of intersection tests performed. This method requires very little pre-processing and hence works well in dynamic 3D scenes. We evaluate the effect of our method with the VCT method and the ray tracing method. Moreover, we observe a significant improvement compared with the VCT method.

## II. RELATED WORK

In this section, we present a brief review of previous works on GI calculation.

Usually, light is scattered multiple times before reaching the eye. The GI algorithm calculates this natural phenomenon of the virtual scene. In the direction of realistic rendering, researchers have investigated a large number of realistic GI algorithms, including path tracing, photon mapping, many-light, radiosity, metropolis light transport (MLT) [5]–[10]. These algorithms can build physically plausible GI effects, some of which can be used together to build more accurate results, but these algorithms require a long calculation time and are generally not directly applicable to real-time rendering.

The introduction of new level programmable commodity GPU hardware in 2002 leads to an exploration in the number of real-time rendering algorithms. This paper mainly focuses on the study of the real-time rendering algorithms. We will analyze the real-time GI algorithms in detail. The core of the real-time GI algorithm is how to collect surrounding optical signals to approximate the rendering equation and take full advantage of the high parallel computing power of GPU. At present, the representative GI algorithms in real-time rendering direction mainly include virtual point light method (VPL) [11], reflective shadow map method (RSM) [12], screen space method [13]–[15], PRT method [16], light propagation volume method (LPV) [17], VCT method [3], Real-time ray tracing method [18], [19]. Among them, PRT method, VCT method, screen space method and real-time ray tracing method are hot research issues. Since screen space ambient occlusion (SSAO) method was first proposed by Vladimir Kajalin in 2007, a large number of improved algorithms, such as screen space volume obscurance (SSVO), screen space directional occlusion (SSDO), horizon based ambient occlusion (HBAO) and high definition ambient occlusion (HDAO) [20], [21] have been investigated. Real-time raytracing method is proposed in recent years and has good development potential. The specific analysis of the representative real-time GI algorithms is shown in Table 1.

In addition, in order to improve the efficiency of the GI algorithm, researchers have carried out explorations on the algorithm implementation. These efforts mainly focus on the integration of rendering task and distributed computing, aiming to make use of the powerful computing power of computer clusters. DeMarle *et al.* [22] used the scene segmentation method and a shared memory-based data communication to achieve interactive ray tracing. Based on the Manta Interactive Ray Tracer rendering engine, Ize *et al.* [23] used the bounding volume hierarchy (BVH) acceleration structure and a multi-thread memory sharing method to boost the speed of distributed ray tracing.

At present, researchers have done a lot of research on real-time GI rendering. However, the real-time GI algorithm of large-scale virtual geographic scenes mainly focuses on static scenes. Most algorithms need pre-computation, and thus these algorithms are difficult to meet the needs of various applications. There are few studies on the real-time GI calculation for large-scale dynamic virtual geographic scenes.

## III. THEORETICAL FOUNDATION

First, we give the theoretical foundation of our algorithm. Our algorithm mainly focuses on the diffuse effect. The calculation of GI is essentially the solution process of the rendering equation [4]. Kajiya's rendering equation states that the outgoing radiance $L_o(x, \omega)$ at a point $x$ in direction $\omega$ is the sum of an emitted radiance term and a reflected radiance term, where the reflected radiance $L_r(x, \omega_o)$ is calculated according to (1).

$$L_r(x, \omega_o) = \int_{\Omega^+} L_i(x, \omega_i) f_r(x, \omega_i \rightarrow \omega_o) < N(x), \omega_i >^+ d\omega_i. \tag{1}$$

**TABLE 1. Representative real-time GI algorithms.**

| Algorithm | VPL | RSM | SSAO/SSVO/SSDO/HBAO/HDAO | PRT | LPV | VCT | Real-time ray tracing |
|---|---|---|---|---|---|---|---|
| Dynamic scene | Support | Support | Support | Partial support | Support | Support | Support |
| Dynamic lighting | Support | Support | Support | Support | Support | Support | Support |
| Advantage | The calculation cost is small and the GI effect is credible. | Only one reflection is used to do the approximation calculation. The calculation cost is small. | It is suitable for the modern deferred shading framework. | It has a good diffuse reflection effect and can be applied to large scale scenes. | The diffuse reflection effect is supported with no preprocessing of scene geometry or light transfer. | Both diffuse and glossy effects are supported without any preprocessing. It has a wide range of applications and high practicality. | The calculation effect of this method is more accurate and more realistic, which is one development direction in the future. |
| Limitation | It mainly focuses on the point light source or the flashlight source. There is jitter in the GI effect, and the research on the distribution method of VPLs is a difficult problem. | In order to achieve fast calculation, occlusion information is usually ignored, resulting in unrealistic lighting effects. | Only the information projected onto the screen space is used in the GI calculation, ignoring the entire scene information. | Pre-computation of scene information is required. When used in large-scale dynamic scenes, pre-calculation of the algorithm is the bottleneck of real-time rendering. | There is jitter in the GI effect, and the mesh partition strategy will cause light leaks. | The storage consumption for the algorithm is large, and it is difficult to handle large-scale dynamic scenes. | Currently, it requires high-performance hardware support; typically needs to be used in combination with other rasterization methods. Moreover, a lot of training work is needed. |

where $\Omega^+$ is the upper hemisphere integral field oriented around the surface normal $N(x)$ at point $x$, $f_r(x, \omega_i \rightarrow \omega_o)$ is the bi-directional reflectance function (BRDF) of the surface and $< N(x), \omega_i >^+$ represents the dot product clamped to zero.

Normally, the BRDF can be treated as a constant term $\rho$ for the diffuse color, and thus in the GI computation the reflected radiance $L_r\ (x, \omega_o)$ can be simplified as follows:

$$L_r(x, \omega_o) = \frac{\rho}{\pi} \int_{\Omega^+} L_i(x, \omega_i) < N(x), \quad \omega_i >^+ d\omega_i. \tag{2}$$

Many techniques exist which try to simplify the integral $\int_{\Omega_k^+} L_i(x, \omega_i) < N(x), \omega_i >^+ d\omega_i$ by making various trade-offs. We introduce a pre-filtering scheme-based cone tracing simplification method. We can send out a few cones over the hemisphere to collect illumination and approximate the radiance of each cone using a constant term based on a filter (for example, VCT method, which adopts the quadrilinear interpolation and a mip-map pyramid). The reflected radiance $L_r\ (x, \omega_o)$ can be simplified as follows:

$$L_r(x, \omega_o) = \frac{\rho}{\pi} \sum_{k=1}^{n} L_k(x, \omega_k) \int_{\Omega_k^+} < N(x), \omega_i >^+ d\omega_i. \tag{3}$$

where $n$ is the number of cones and $L_k(x, \omega_k)$ is the filtered radiance of each cone.

If we treat the term $\int_{\Omega_k^+} < N(x), \omega_i >^+ d\omega_i$ as a weighted term for a further simplification, then the equation can be written as:

$$L_r(x, \omega_o) = \frac{\rho}{\pi} \sum_{k=1}^{n} L_k(x, \omega_k) W_k. \tag{4}$$

## IV. ALGORITHM REVIEW

In this section, we describe in detail the issue we deal with and present the framework of our algorithm. The goal of our algorithm is to allow real-time rendering of dynamic GI effects, which has better scalability than the VCT method.

### A. PROBLEM DEFINITION

GI effect is computationally expensive. It requires not only complex visibility computations between arbitrary points but also the integration computation over a large number of directions. These two computations become extremely difficult tasks for the real-time or interactive algorithms of the dynamic scene. We work on the integration of cone tracing method and pre-filtering scheme in order to allow GI computation on open world scenes. The integration of cone tracing method and pre-filtering scheme can dramatically reduce

the computation cost of the visibility and integration shown in (3).

In this paper, we suppose that the 3D scene of interest is represented by triangle meshes and the scene can be divided into pieces. Based on this assumption, we divide the scene into different mesh types. For different mesh types, we use different methods for lightmap calculations and then use a unified model calculation for synthesis. Based on the theoretical foundation described in Section III, the cone tracing approximation of the GI computation is reasonable. For different lightmap representations, we adopt different cone tracing strategies and design efficient data structures to achieve the cone tracing computation. Enough cones will make the result of our algorithm closer to the accurate solution.

### B. ALGORITHM FRAMEWORK

As shown in Fig. 1, our approach is mainly divided into two steps: lightmap computation and hybrid cone tracing. We first divide the given 3D scene into two types: complex meshes and height-field meshes. Two type lightmaps are used to receive incoming radiance from dynamic light sources. We use 2D texture to represent the lightmap of the height-field mesh, while use 3D texture to represent the lightmap of the complex mesh. This will provide us with a GPU-friendly data structure, whose computation is highly parallelizable. The radiance storage computation is done by rasterizing or voxelizing the corresponding mesh from all light sources. Then, the lightmap values are mipmaped (filtered) into higher levels. We rely on the GPU hardware acceleration technology to do the calculations. Finally, we design a hybrid cone tracing method for use in combination with our hybrid lightmap representation, which can significantly reduce the number of intersection tests performed. The hybrid cone tracing method is performed to do the final gathering. This method requires very little pre-processing and hence works well in dynamic 3D scenes.

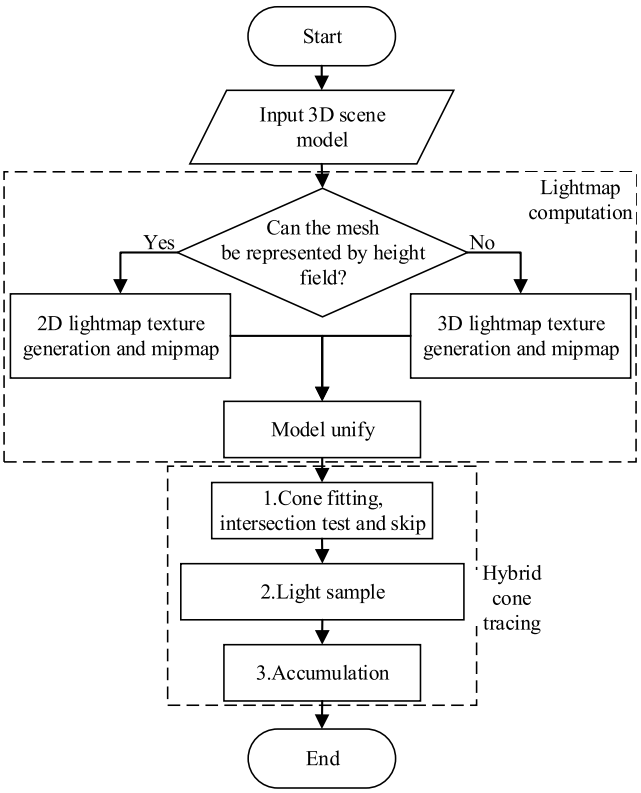



**FIGURE 1.** The flow chart of hybrid cone tracing method.

## V. LIGHTMAP COMPUTATION

To address the storage consumption problem with the previously proposed VCT method, we propose a hybrid lightmap representation method that can be applied to more 3D scenes. For a given 3D scene/mesh, the corresponding lightmap is a discrete sample of the incoming radiance and mesh geometry information. In our algorithm, lightmap will serve as our means of rapidly computing the visibility and integration of the rendering equation for any point in the scene we wish to light. In Subsection A, the judgment calculation of whether a mesh can be represented by height field is introduced. In Subsection B, we introduce the multi-resolution sampling strategy for the hybrid lightmap representation. In Subsection C, a unification strategy for the hybrid representation is presented.

### A. JUDGMENT OF HEIGHT FIELD REPRESENTATION

When the 3D scene model is loaded, we firstly do the judgment calculation of whether a mesh can be represented by height field. For each mesh, the calculations are as follows:

Step1: calculate the bounding box of the mesh, and then use the bottom surface of the bounding box with a certain mesh resolution to sample this mesh to get a height field simplification of the mesh. For each sample point (x, z), the sample value y(x, z) is the maximum value along the Y axis.

Step2: compute the geometric difference between the mesh and the height field simplification mesh. This calculation is a quite common task in mesh processing [24], [25]. However, many accurate comparison algorithms are computationally inefficient. And the situation we discussed is relatively simple and does not involve the problem of model pose adjustment. Thus, we use parallelization Hausdorff distance method to do the judgment, which is enough for this problem. Hausdorff distance is calculated according to (5) in this paper.

$$d(X, Y) = \underset{x \in X}{avg}(\inf_{y \in Y} \|x - y\|)/D. \tag{5}$$

where $X$ represents the vertex set of the original mesh, $Y$ represents the vertex set of the height field simplification mesh, $avg(\cdot)$ denotes the average of the Euclidean distance, $\inf_{y \in Y} \|x - y\|$ searches for each $x$ the closest point $y$ and calculates the Euclidean distance, $D$ depicts the body diagonal of the bounding box. $D$ is used to eliminate the influence of model units. This calculation is realized based on compute shader and octree acceleration structure. Each vertex $x$ is

assigned a compute thread. If $d(X, Y)$ is less than a certain value, then the mesh can be represented by height field. The threshold is usually a small value. In our experiment, the threshold is set to 0.001. In order to improve the calculation efficiency, center of gravity of each triangle can be used to replace the vertex point in the distance calculation.

### B. MULTI-RESOLUTION SAMPLING

As described above, the 3D scene is divided into two types: complex meshes and height-field meshes, based on the judgment of whether the mesh can be represented by height field. For different mesh types, we use different methods for lightmap calculations. In our multi-resolution sampling method, given a 3D mesh, if it can be represented by the height-field mesh, then the lightmap of this mesh is a 2D texture.

For a complex mesh, we use 3D lightmap texture to receive the incoming radiance and cache the spatial attribute of the mesh. The 3D texture is calculated by the GPU hardware rasterizer (voxelization) [26]. Usually, the resolution $256 \times 256 \times 256$ for a piece of the open world scene can meet the visual requirements. Then, the lightmap is mipmaped. GPU hardware acceleration technology is used to do the calculation efficiently. Each voxel of the lightmap contains the diffuse albedo and normal information of the geometry and is used later to do the GI calculation [3]. Since the voxel grid is recreated each frame, the multi-resolution sampling is fully dynamic and does not rely on any pre-computations.

For a height-field mesh whose surface can be described as a single valued function of two coordinates $(x, z)$, the 2D lightmap texture is used to cache the incoming radiance and geometry information. Firstly, the mesh is rasterized according to the resolution of the 2D lightmap texture. Each pixel stores the incoming radiance of the corresponding grid point, and the alpha channel is used to store the height value of the grid point. This will not only reserve the height information for the subsequent calculations but also significantly reduce the storage space. Then, the lightmap is mipmaped. In the mipmap process, the lightmap is used to implement filtering of the scene geometry information. The calculation is repeated each frame as the voxelization process. Also, GPU hardware acceleration technology is used to do the calculation.

At last, for each mesh, a corresponding lightmap is generated and filtered. This kind of direct GPU representation will be very helpful for the GI computation. Changes of light source and geometry do not affect the average frame rate of the algorithm.

### C. UNIFICATION CALCULATION

During the cone tracing, the cone may pass from one lightmap representation to another. Given the hybrid lightmap representation, to ensure that the GI computation has a unified cone tracing result, it is necessary to do the unification calculation between different lightmap representations. In the final calculation, the use of the currently described hybrid lightmap representation can be mathematically described as (6) and (7).

$$voxelvalue += accVoxel(distance, mesh1, 3Dlightmap). \quad (6)$$

$$texelvalue += accTexel(distance, mesh2, 2Dlightmap). \quad (7)$$

where *distance* represents the forward distance of cone tracing, *mesh*1 represents the spatial attribute of the complex mesh, *mesh*2 denotes the morphological characteristic of the height-field mesh, *accVoxel* and *accTexel* depict the accumulation functions of the lightmap. The mesh term represents different ways of dealing with the mesh and lightmap. The actual unification of the cone tracing calculation should be the unification of distance. That is, we should unify the forward distance in each lightmap space. Therefore, we calculate the distance in each lightmap (3D lightmap texture / 2D lightmap texture) layer based on the world coordinate system and unify the distances according to the mipmap level and the forward distance of cone tracing in the world coordinate system. Of course, if the resolution of 2D lightmap is an integer multiple of the resolution of 3D lightmap, the unification calculation will be simpler.

The detailed steps of this process are described as follows:

Step1: use unified metric space to calculate the lightmaps of the complex mesh and the height-field mesh, respectively. That is, the complex mesh and the height-field mesh are scaled to a uniform scale [Dx, Dy, Dz] for lightmap computation (voxelization and rasterization). We calculate the bounding boxes of the two types of mesh and use the bounding box of the union of the calculated bounding boxes as the reference. Then, we scale the two types of mesh to the reference size.

Step2: adopt (8) to calculate the Level Of Detail (LOD) level of the current lightmap, during the cone tracing.

$$Lv = log2(diameter/xelworldsize). \quad (8)$$

where *diameter* represents the forward distance of cone tracing in the world coordinate system, *xelworldsize* represents $max(Dx, Dy, Dz)/voxel\_resolution$ for the 3D lightmap and represents $max(Dx, Dy, Dz)/texel\_resolution$ for the 2D lightmap. The derivation of this equation is based on the power-of-two mip-map scheme.

Step3: perform the coordinate correction calculation according to (9).

$$pos = pos\_pre * max(Dx, Dy, Dz)/max(boundingbox(x), boundingbox(y), boundingbox(z)). \quad (9)$$

where *pos_pre* represents the original coordinate, $boundingbox(x)$ represents the x-coordinate of the bounding box of the mesh, $boundingbox(y)$ represents the y-coordinate of the bounding box of the mesh and $boundingbox(z)$ represents the z-coordinate of the bounding box of the mesh.

## VI. HYBRID CONE TRACING

To use the hybrid lightmap representation, we propose a hybrid cone tracing method that can efficiently complete

the GI computation. Based on the theoretical foundation described in Section III, the core of the cone tracing approximation method is to calculate the filtered radiance of each cone. We use a step-by-step fitting cube method to achieve this calculation. In the step-by-step fitting cube method, cone tracing can essentially be seen as the calculations of intersection of the fitting cube and the scene and light sampling. Therefore, we divide the cone tracing process into three steps: step-by-step intersection test and skip, light sampling and step-by-step accumulation model. In Subsection A, we introduce the intersection test and skip method, including intersection calculation of the fitting cube and the 3D lightmap, intersection calculation of the fitting cube and the 2D lightmap and the skip strategy. In Subsection B, we present the light sampling method, including light sampling calculation of 3D lightmap and light sampling of 2D lightmap. At last, the step-by-step accumulation model is introduced in Subsection C.

### A. INTERSECTION TEST AND SKIP METHOD

In our hybrid cone tracing method, each cone is generated based on the tangent space of the surface point. The essence of the step-by-step fitting cube method is to fit the sample space of the cone along the central axis of the cone step by step. In each step, an axis-aligned fitting cube is constructed. The schematic diagram is shown in the left part of Fig. 2. The step-by-step intersection test and skip method is mainly used to solve the intersection calculation between the fitting cube and the filtered scene represented by the lightmap and perform skip judgment based on the result of intersection test. We use the step-by-step refinement and early rejection strategy to increase computational efficiency. All calculations are expressed by vector calculations and symbolic judgments, which have a good computational efficiency.

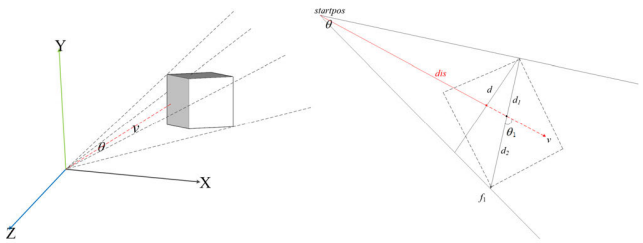



**FIGURE 2. The schematic diagram for the step-by-step fitting cube method. Left: 3D schematic diagram for the step-by-step fitting cube. Right: The schematic diagram for the fitting cube calculation.**

For the intersection calculation of the fitting cube and the 3D lightmap, the calculations are conditionally limited and optimized to combine the calculation of the height-field cone tracing. This intersection calculation not only judges the intersection situation, but also finds the intersection points for the subsequent calculations. We adopt axis-aligned fitting cube to fit the sample space step by step. This kind of calculation method can quickly perform the intersection test, which saves a large amount of 3D triangle intersection calculations compared to the cone axis-aligned fitting cube. We derive the calculation formula for the fitting cube. Our method performs the simple scale $S$ and translation $T$ transformations on the unit cube at the origin to calculate the fitting cube. This is more efficient than the complex rotation and translation calculations of the cone axis-aligned fitting cube. The schematic diagram is shown in the right part of Fig. 2. Detailed calculations are as follows.

If the vector $v$ is parallel to the X-axis (or Y-axis, or Z-axis), then the operators are calculated according to (10).

$$\begin{cases} S = d/\sqrt{2}, \\ T = sp + v * (dis + S/2). \end{cases} \tag{10}$$

If the vector $v$ is not parallel to the X-axis, Y-axis, and Z-axis, then the operators are calculated according to (11).

$$\begin{cases} S = (d_1 + d_2)/\sqrt{3}, \\ T = sp + v * (dis + cos(\theta_1) * d_1) + (d_2 - d_1)/2 * f_1. \end{cases} \tag{11}$$

The parameters of (9) and (10) are as follows:
$v$−unit vector of the central axis of the cone;
$sp$−start point of the cone;
$d$−radius of the cone bottom in the current sample space;
$dis$−forward distance of cone tracing;
$f_1$−unit vector of the body diagonal of the unit cube;
$d_1 = d/2/sin(\theta_1)$;
$d_2 = (dis + cos(\theta_1) * d_1) * sin(\theta/2)/sin(\theta_1 - \theta/2)$;
$\theta$−cone angle;
$\theta_1 = acos(v * f_1/(|v| * |f_1|))$.

Then, we perform the intersection calculation. There are four kinds of spatial relationship between the 3D lightmap and the fitting cube: 3D lightmap contains fitting cube, fitting cube contains 3D lightmap, 3D lightmap intersects with fitting cube and 3D lightmap does not intersect with fitting cube. Based on the above fitting cube calculation, we adopt the following equation to perform the intersection test calculation and calculate the intersection points:

$$\begin{aligned} [X, Y, Z] = &[X^1_{[L^1,U^1]}, Y^1_{[L^1,U^1]}, Z^1_{[L^1,U^1]}] \\ &\cap [X^2_{[L^2,U^2]}, Y^2_{[L^2,U^2]}, Z^2_{[L^2,U^2]}]. \end{aligned} \tag{12}$$

where, $[X, Y, Z]$ represents the intersection point, $[X^1_{[L^1,U^1]}, Y^1_{[L^1,U^1]}, Z^1_{[L^1,U^1]}]$ represents the bounding box of the 3D lightmap and $[X^2_{[L^2,U^2]}, Y^2_{[L^2,U^2]}, Z^2_{[L^2,U^2]}]$ represents the bounding box of the fitting cube. It can be seen from the equation that this calculation is very efficient, avoiding complex triangle-triangle intersection test and intersection point calculation. Based on the final intersection calculation result, we can quickly make a skip decision. If the fitting cube intersects with 3D lightmap, further calculations can be performed; otherwise, the sample space is just skipped.

For the intersection calculation of the fitting cube and the 2D lightmap, the schematic diagram is shown in Fig. 3. In the calculation process of the intersection judgment of the fitting cube and the 3D texture, the intersection judgment of the fitting cube and the height-field mesh is

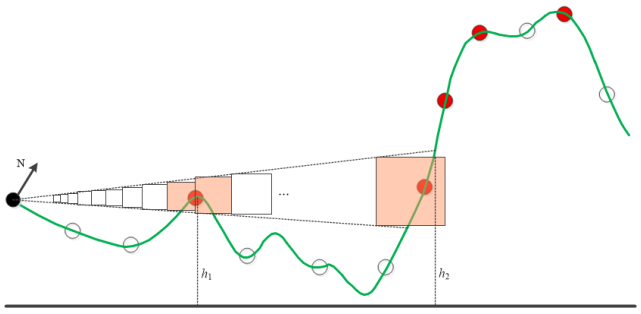



**FIGURE 3.** The schematic diagram for the height-field cone tracing method. The orange square indicates where the sample is to be taken.

simultaneously performed. The sampling rule of the height-field cone tracing is that sampling is started at the minimum height of the cone and abandoned at the maximum height of the cone.

We design a hierarchical min-max summed area table (HSAT) method to realize the judgment calculation efficiently. Since the cone tracing on the height-field mesh is based on the hierarchical 2D lightmap in the hybrid representation-based cone tracing, we perform the min-max SAT calculation [27], [28] for each layer to optimize the intersection test calculation. Based on the min-max SAT of each layer of the 2D lightmap, the fast approximate calculation of minimum and maximum height in the projection quadrilateral (the vertical projection of the fitting cube) of the height-field mesh can be realized.

Generating a box-filtered value using a summed-area table requires sampling the summed-area table at the four corners of a rectangular filter region $[S_{rt}, S_{lb}, S_{lt}, S_{rb}]$. The filtered value is calculated according to (13).

$$s_{filter} = \frac{S_{rt} + S_{lb} - S_{lt} - S_{rb}}{w_x * w_y} \tag{13}$$

where $w_x$ and $w_y$ are the width and height of the filter kernel, $S_{rt}$ is the right-top sample of the filter kernel, $S_{lb}$ is the left-bottom sample of the filter kernel, $S_{lt}$ is the left-top sample of the filter kernel and $S_{rb}$ is the right-bottom sample of the filter kernel.

Maximum value of the filter kernel can be approximated by (14). This method uses some expensive floating point operations, but its run time is constant in the kernel width. The crux of the method is the following approximation calculation, which works well for $0 < x_i < 1, p \gg 1$.

$$max_{i=1}^{N} x_i \approx (\frac{1}{N} \sum_{i=1}^{N} x_i^p)^{1/p}. \tag{14}$$

The minimum filter can be implemented by finding the maximum of the inverse value and then inverting the result.

Based on the filtered value, the intersection judgment of the fitting cube and the height-field mesh can be efficiently implemented. By using the intersection judgment result, we can achieve a fast step-by-step skip test in the step-by-step fitting cube method.

### B. LIGHT SAMPLING

In our algorithm, we use a step-by-step fitting cube method to calculate the filtered radiance of each cone. Based on the intersection calculation of the fitting cube and the lightmap, the light sampling calculation is performed in the fitting cube step by step.

We want to incorporate the GI effect on any point $p$, from all surrounding objects in the scene, not just the effect of nearby concavities. The fact is that objects at a greater distance should have a lesser, blurrier effect than those near the point $p$. So, we use a pre-filtering scheme-based step-by-step multi-resolution sampling method. Each sample (at a distance *dis*) can be treated as capturing the occlusion effects of objects at a distance *dis*. To simulate the increasing blurring effect, our method uses distance-based pre-filtered copies of the lightmap in each step. The fitting cube size for each step is proportional to each *dis*. This maps well to the GPU texture implementation. The pre-filtered lightmap textures are stored in the mip-map chain in the video memory. Based on the calculation method of LOD level described in Subsection B of Section V, the lightmap texture can be efficiently sampled using LOD level biased texture load instructions in the pixel shader.

Light sampling in the fitting cube consists of the light sampling of the irregular projection region in the 2D lightmap and the sampling of the 3D lightmap, as shown in Fig. 4. Therefore, the core of the light sampling method is how to quickly sample the average value of the irregular region of the 2D lightmap and the average value of the intersection part of the 3D lightmap formed by the 3D lightmap and the fitting cube.

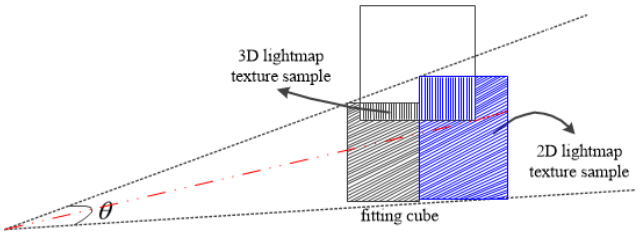



**FIGURE 4.** The schematic diagram for the light sampling of the hybrid lightmap representation.

For the light sampling calculation of the 3D lightmap, we use the geometric center of the intersection part to sample the corresponding pre-filtered copy of the 3D lightmap. Based on the intersection calculation of the fitting cube and the 3D lightmap, geometric center of the intersection points can be calculated and used to sample the 3D lightmap. The geometric center is calculated according to (15).

$$M = (\frac{\sum_{k=1}^{n} x_k}{n}, \frac{\sum_{k=1}^{n} y_k}{n}, \frac{\sum_{k=1}^{n} z_k}{n}). \tag{15}$$

where $n$ is the number of the intersection points.

$M$ is the center of gravity of the intersection part. That is, we use the regional volume distribution as an attribute to

measure the irregularity of the model. Based on the geometric center and the scale information of the sample space, we use quadrilinear interpolation to ensure a smooth effect. We set a threshold for the size of the intersection part. If the size of the intersection part is less than the threshold, then the light sampling calculation can be canceled. We use the bounding ball to do this test. If the radius is less than a certain value, then the calculation can be canceled. During the implementation, there is a self-influence phenomenon that needs to be dealt with. The sample point will be offset in normal direction of the point $p$ to eliminate the bright spot caused by the first sample of the cone tracing.

For the light sampling of the 2D lightmap, we adopt a morphology search method based on the cone direction. The radiance of point $p$ on the height field is calculated based on the weighted average calculation of the normal direction and the cone direction. Based on the intersection calculation of the fitting cube and the 2D lightmap, the morphology search method first divides the projection region, and then performs the morphology digital differential analyzer (DDA) search, including backface culling and dynamic tracing distance. The segmentation of the projection region is shown in Fig. 5.

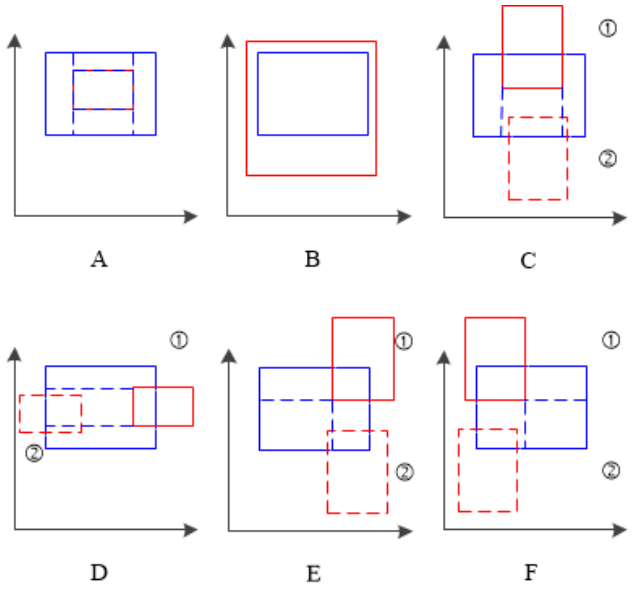



**FIGURE 5.** The schematic diagram for the projection segmentation. The red solid line box represents the projection of the 3D lightmap on the height-field mesh, the red dashed box represents another case of the 3D lightmap projection, the blue solid line box represents the projection of the fitting cube, and the blue dashed line indicates the segmentation of the projection of the fitting cube.

In the second step, the projection segmentation is subdivided, and the height-field morphology search is performed along the projection direction of the cone direction using the 2D DDA method. When there is an occlusion, the previous height is used to determine whether or not the subsequent sampling is retained. So, this is a dynamic distance tracing. The radiance of the center point of the sub-mesh is taken as the radiance of the sub-mesh, and the height of the center point is used as the height of the sub-mesh.

Similar to the light sampling of the 3D lightmap, we also need to handle the self-influence phenomenon.

### C. ACCUMULATION MODEL

Based on the emission-absorption optical model introduced in [29], [30], the accumulation model is used to accumulate the sampling results of each step along the cone. The detailed steps of this calculation are described as follows:

Case 1: the fitting cube only intersects with the 3D lightmap. In each step, the occlusion $a$ can be calculated from the alpha channel of the 3D lightmap. We update the values using the front-to-back accumulation.

Case 2: the fitting cube only intersects with the 2D lightmap. We can adopt (16) to calculate the occlusion $a$ in each step.

$$a = (h_2 * A_2 - h_1 * A_1)/(h_2 * A_2). \tag{16}$$

where $h_1$ and $h_2$ are the average heights of the two sample spaces, $A_1$ and $A_2$ are the occluded areas of the cross sections of the cone. In order to achieve fast approximation calculation, we use (17) to calculate the occlusion $a$ in each step based on the approximate similar triangle principle.

$$a = \frac{step\_diameter}{current\_step\_length}. \tag{17}$$

where $step\_diameter$ represents the size of the sample space and $current\_step\_length$ represents the current forward distance of cone tracing. Based on our experiments, this approximation calculation can have a high efficiency and a good effect.

Mix intersection case: As shown in Fig. 4, the intersection of the fitting cube and the 3D lightmap is consistent between two adjacent levels. Therefore, the sampling value of 3D lightmap is first attenuated by the occlusion $a$ calculated by the method of 'Case 1'. Then, the overall average height in each sample space is calculated based on the sample values of 2D lightmap and 3D lightmap. At last, the method of 'Case 2' is used to calculate the final accumulation value.

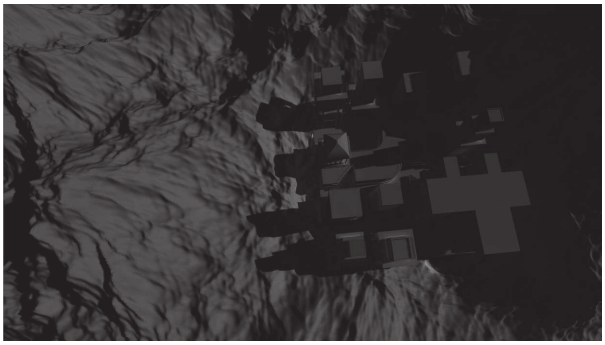
Once the cone falls completely into the 3D lightmap during the tracing process, the 'Case 1' method is used to do the calculation based on the assumption that cone does not penetrate the 3D lightmap after entering the 3D lightmap.

## VII. RESULTS AND DISCUSSION

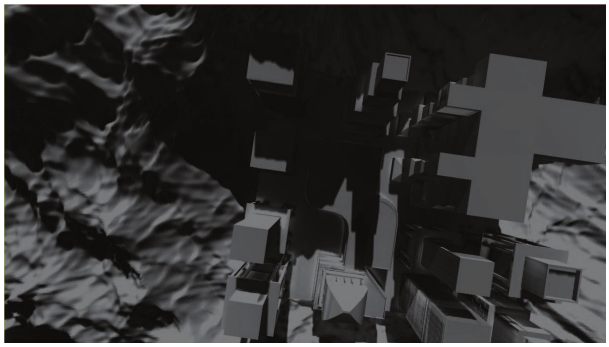
We implemented our hybrid cone tracing method on an NVIDIA GTX 1080 system with an Intel Core i7-6700 CPU. We achieve real-time frame rates on complex scenes. The dynamic incoming radiance (energy and direction) is injected into the hybrid lightmap representation. In our implementation, the complex mesh is voxelized with a compute pass that generates and mipmaps the corresponding 3D lightmap. Usually sparse voxel octree is not fast enough to use per-frame in a real-time application. 3D texture based on the GPU hardware acceleration technology is used to do the

calculation efficiently. The 3D lightmap is generated based on the bounding box of the complex mesh. The height-field mesh is rasterized with a compute pass that generates and mipmaps the corresponding 2D lightmap. We create a HSAT compute pass that is able in a single draw call, to efficiently generate the hierarchical min-max summed area table in parallel. At last, the scene is rendered with a final shading pass that samples the hybrid lightmap represent. In the final shading pass, the hybrid cone tracing method is used to do the final gathering. In this paper, we only deal with the diffuse.

Fig. 7 shows the accurate solution created by the V-Ray (Adv 3.40.01) & 3DS MAX 2017. Fig. 6 shows the GI effect that our method achieves for the application case (1850K triangles) with a 7-level 3D lightmap ($128^3$ virtual resolution) and an 11-level 2D lightmap ($2048^2$ virtual resolution). We use 6 cones to do the final gathering. The screen resolution of all these cases is $1280 \times 720$. On the application scene, the average frame rate of our method is 50FPS. Compared with the effect of VCT method (Fig. 8), our hybrid cone tracing method results in a more detailed description of the GI effect. We use an 8-level 3D lightmap ($256^3$ virtual resolution) for the VCT method. On the application scene, the average frame rate of VCT method is 80FPS with 6 cones, for the $1280 \times 720$ viewport.

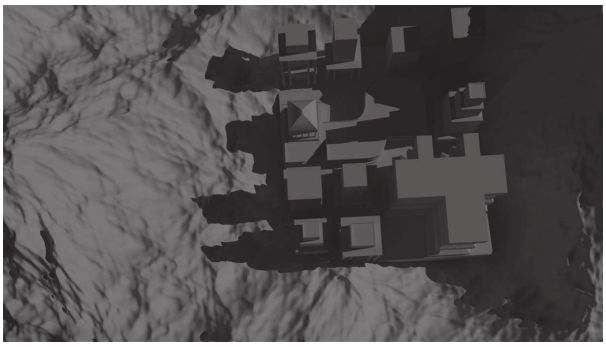


(a)



(b)

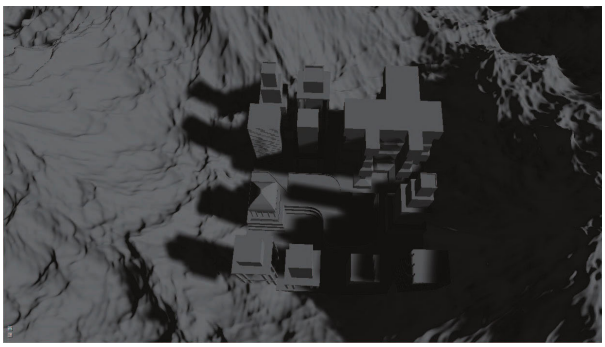
**FIGURE 6.** The effect of our hybrid cone tracing method. Shadowing in this image comes from the shadow map (4096 × 4096 resolution).



(a)

(b)

**FIGURE 7.** The effect of ray tracing. The ground truth is rendered with V-Ray. We use the V-Ray shadow and brute force GI setting.



(a)

(b)

**FIGURE 8.** The effect of 3D texture voxel cone tracing method. Shadowing in this image comes from the shadow map (4096 × 4096 resolution).

It can be seen from the result that the effect of our algorithm looks pleasing and has smaller deviation from the accurate solution compared with the VCT method. Our algorithm can capture much more indirect lighting information. Compared with the ray tracing method, our algorithm yields roughly correct results. The ray tracing method takes 67 seconds to achieve the result in the same hardware environment.

However, our method can produce visual plausible results in real time. Despite being physically incorrect, in the sense of approximately capturing the physics of light bouncing from one diffuse surface to another, our method is this kind of simple, free yet pleasing trick which can go a long way in the real-time application.

We also compared our method with the 3D texture VCT method in the term of GI memory consumption. GI memory consumptions for the two methods are shown in Table 2.

**TABLE 2. Memory consumption comparison.**

| Algorithm | GI Memory for the Case |
|---|---|
| 3D texture VCT | 896MB |
| Our method | 117MB |

Compared with the 3D texture VCT method, our method requires less GPU memory for the GI computation because it is based on a hybrid lightmap representation. However, compared with the VCT method, our method requires more skip tests and samplings and thus is slower. The additional computations for the 2D lightmap significantly degrade the time performance. The VCT method does not consider the mesh characteristics. It simply uses the 3D lightmap to filter and sample the incoming radiance of the scene.

The core of our method is to divide the 3D scene into two types and use different processing methods, based on the judgment of whether the mesh can be represented by height field. In this way, we can not only achieve the expected results, but also reduce memory consumptions. This scheme will allow the scale of the application scenario of pre-filtering cone tracing strategy to be enlarged. Although we use the HSAT method in the height-field cone tracing, the morphology search method of the 2D lightmap slows the performance of the height-field cone tracing method. Therefore, a more efficient sampling strategy is needed. Although our method can not accelerate the frame rate, it can get better results under the premise of guaranteeing the effective use of memory.

As reported in [3], our method also suffers from light leaking due to the discrete representation of the geometry and irradiance. The leaking phenomenon can be reduced by increasing the resolution of the lightmap.

With the development of RTX technology [31], GPU hardware and cloud computing, ray tracing is getting more and more attention. However, current algorithms do not achieve complete real-time ray tracing. These methods require high-performance graphics cards and heavy filtering. These methods are not yet suitable for most application scenarios. And, filtering is based on deep learning methods. The generalization ability of the method needs further study, especially for many real-time generated 3D scenes. Of course, we also believe that real-time ray tracing methods have a bright future. Our method focuses on lightweight real-time GI. The main contributions of our method are hybrid representation and intersection calculation method. This will be useful for RT-based methods, because the core of RT-based methods includes intersection calculation method. And, no matter what, data representation and intersection calculation are of great important for the GI calculation.

## VIII. CONCLUSION

This paper proposes a hybrid cone tracing method for performing GI computation on open world scenes. Unlike the previously proposed VCT method, our method can get better results under the premise of guaranteeing the effective use of memory. The method is scalable for high polygonal scenes and requires very little pre-processing. It can be applied to dynamic scenes.

Our method has some limitations. It uses the morphology search method during the height-field cone tracing process, which performs a lot of searching and sampling. This is an important factor affecting efficiency.

Our method can allow the scale of the application scenario of pre-filtering cone tracing strategy to be enlarged. With enough cones, our algorithm will tend to ray tracing. Further work has to be done in terms of computation optimization. A more efficient sampling strategy is needed.

## REFERENCES

[1] W. A. Stokes, J. A. Ferwerda, B. Walter, and D. P. Greenberg, "Perceptual illumination components: A new approach to efficient, high quality global illumination rendering," *ACM Trans. Graph.*, vol. 23, no. 3, pp. 742–749, Aug. 2004.
[2] A. Evans, "Fast approximations for global illumination on dynamic scenes," in *Proc. ACM SIGGRAPH Courses (SIGGRAPH)*, Boston, MA, USA, 2006, pp. 153–171.
[3] C. Crassin, F. Neyret, M. Sainz, S. Green, and E. Eisemann, "Interactive indirect illumination using voxel cone tracing," *Comput. Graph. Forum*, vol. 30, no. 7, pp. 1921–1930, Sep. 2011.
[4] J. T. Kajiya, "The rendering equation," in *Proc. 13th Annu. Conf. Comput. Graph. Interact. Tech.*, New York, NY, USA, 1986, pp. 143–150.
[5] A. Keller, "Instant radiosity," in *Proc. 24th Annu. Conf. Comput. Graph. Interact. Techn. (SIGGRAPH)*, New York, NY, USA, 1997, pp. 49–56.
[6] E. Veach and L. J. Guibas, "Metropolis light transport," in *Proc. 24th Annu. Conf. Comput. Graph. Interact. Techn. (SIGGRAPH)*, New York, NY, USA, 1997, pp. 65–76.
[7] B. Walter, P. Khungurn, and K. Bala, "Bidirectional lightcuts," *ACM Trans. Graph.*, vol. 31, no. 4, pp. 1–11, Aug. 2012.
[8] Y. Huo, R. Wang, S. Jin, X. Liu, and H. Bao, "A matrix sampling-and-recovery approach for many-lights rendering," *ACM Trans. Graph.*, vol. 34, no. 6, pp. 1–12, Nov. 2015.
[9] Y. Huo, R. Wang, T. Hu, W. Hua, and H. Bao, "Adaptive matrix column sampling and completion for rendering participating media," *ACM Trans. Graph.*, vol. 35, no. 6, pp. 1–11, Nov. 2016.
[10] T. Müller, M. Gross, and J. Novák, "Practical path guiding for efficient light-transport simulation," *Comput. Graph. Forum*, vol. 36, no. 4, pp. 91–100, Jul. 2017.
[11] J. Novák, T. Engelhardt, and C. Dachsbacher, "Screen-space bias compensation for interactive high-quality global illumination with virtual point lights," in *Proc. Symp. Interact. 3D Graph. Games (I3D)*, San Francisco, CA, USA, 2011, pp. 119–124.
[12] C. Dachsbacher and M. Stamminger, "Reflective shadow maps," in *Proc. Symp. Interact. 3D Graph. Games (SI3D)*, Washington, DC, USA, 2005, pp. 203–231.
[13] T. Akenine-Moller, E. Haines, and N. Hoffman, "Global illumination," in *Real-Time Rendering*, 3rd ed. Natick, MA, USA: AK Peters, 2008, pp. 327–430.
[14] T. Ritschel, T. Grosch, and H.-P. Seidel, "Approximating dynamic global illumination in image space," in *Proc. Symp. Interact. 3D Graph. Games (I3D)*, Boston, MA, USA, 2009, pp. 75–82.

[15] B. J. Loos and P.-P. Sloan, "Volumetric obscurance," in *Proc. ACM SIGGRAPH Symp. Interact. 3D Graph. Games*, Washington, DC, USA, 2010, pp. 151–156.
[16] P.-P. Sloan, J. Kautz, and J. Snyder, "Precomputed radiance transfer for real-time rendering in dynamic, low-frequency lighting environments," *ACM Trans. Graph.*, vol. 21, no. 3, pp. 527–536, Jul. 2002.
[17] A. Kaplanyan and C. Dachsbacher, "Cascaded light propagation volumes for real-time indirect illumination," in *Proc. ACM SIGGRAPH Symp. Interact. 3D Graph. Games (I3D)*, Washington, DC, USA, 2010, pp. 99–107.
[18] C. R. A. Chaitanya, A. S. Kaplanyan, C. Schied, M. Salvi, A. Lefohn, D. Nowrouzezahrai, and T. Aila, "Interactive reconstruction of Monte Carlo image sequences using a recurrent denoising autoencoder," *ACM Trans. Graph.*, vol. 36, no. 4, pp. 1–12, Jul. 2017.
[19] P. Moreau, M. Pharr, and P. Clarberg, "Dynamic many-light sampling for real-time ray tracing," in *High-Performance Graphics-Short Papers*, M. Steinberger and T. Foley, Eds. Aire-la-Ville, Switzerland: The Eurographics Association, Jul. 2019.
[20] L. Bavoil and M. Sainz. (2008). *Screen space Ambient Occlusion*. [Online]. Available: https://developer.download.nvidia.com/SDK/10.5/direct3d/Source/ScreenSpaceAO/doc/ScreenSpaceAO.pdf
[21] S. Herholz, T. Schairer, and W. Straßer, "Screen space spherical harmonics occlusion ($S_3HO$) sampling," in *Proc. ACM SIGGRAPH Posters (SIGGRAPH)*, Vancouver, BC, Canada, 2011, Art. no. 76.
[22] D. E. DeMarle, C. P. Gribble, S. Boulos, and S. G. Parker, "Memory sharing for interactive ray tracing on clusters," *Parallel Comput.*, vol. 31, no. 2, pp. 221–242, Feb. 2005.
[23] T. Ize, C. Brownlee, and C. D. Hansen, "Real-time ray tracer for visualizing massive models on a cluster," in *Proc. 11th Eurographics Conf. Parallel Graph. Visual. (EGPGV)*, Llandudno, U.K., 2011, pp. 61–69.
[24] P. Cignoni, C. Rocchini, and R. Scopigno, "Metro: Measuring error on simplified surfaces," *Comput. Graph. Forum*, vol. 17, no. 2, pp. 167–174, Jun. 1998.
[25] T. Liu, J. Gao, and Y. Zhao, "An approach to 3D building model retrieval based on topology structure and view feature," *IEEE Access*, vol. 6, pp. 31685–31694, 2018.
[26] M. Schwarz and H.-P. Seidel, "Fast parallel surface and solid voxelization on GPUs," *ACM Trans. Graph.*, vol. 29, no. 6, pp. 1–10, Dec. 2010.
[27] J. Hensley, T. Scheuermann, G. Coombe, M. Singh, and A. Lastra, "Fast summed-area table generation and its applications," *Comput. Graph. Forum*, vol. 24, no. 3, pp. 547–555, Sep. 2005.
[28] H. Tulleken. (2010). *Simple, Fast Approximate Minimum/Maximum Filters*. [Online]. Available: http://code-spot.co.za/2010/04/16/simple-fast-approximate-minimum-maximum-filters/
[29] N. Max, "Optical models for direct volume rendering," *IEEE Trans. Vis. Comput. Graphics*, vol. 1, no. 2, pp. 99–108, Jun. 1995.
[30] K. Engel, M. Hadwiger, J. M. Kniss, A. E. Lefohn, C. R. Salama, and D. Weiskopf, "Real-time volume graphics," in *Proc. ACM SIGGRAPH Course Notes*, Los Angeles, CA, USA, 2004, Art. no. 29-es.
[31] G. Pascal and L. Martin-Karl. (2018). *Practical Realtime Raytracing With RTX From Concepts to Implementation*. [Online]. Available: https://www.nvidia.com/en-us/events/siggraph/schedule/

**TAO LIU** was born in China, in 1988. He received the M.S. and Ph.D. degrees in traffic information engineering and control from Dalian Maritime University, Dalian, China, in 2011 and 2015, respectively. He is currently an Associate Professor with the College of Transport and Communications, Shanghai Maritime University, China. He is involved in the area of computer graphics. His current research interests include real-time rendering, model feature analysis, and 3D display techniques.

**JIN GAO** was born in China, in 1989. He received the B.S. degree in communication engineering from Hainan University, Haikou, China, in 2011, and the M.S. degree in traffic information engineering and control from Dalian Maritime University, Dalian, China, in 2013. He is currently an Engineer with the China Waterborne Transport Research Institute, China. His current research interests include transport simulation and 3D visualization.

**ZHENGLING LEI** was born in China, in 1988. She received the M.S. and Ph.D. degrees in traffic information engineering and control from Dalian Maritime University, Dalian, China, in 2011 and 2014, respectively. She is currently a Lecturer with the College of Engineering Science and Technology, Shanghai Ocean University, China. Her current research interests include optimization theory, ADRC control, and energy optimization techniques.