Fast Multi-image-based Photon Tracing with Grid-based Gathering

Yun Fei, Bin Wang
School of Software, Tsinghua University, Beijing, China, 100084

Abstract We developed a real-time solution for approximate global illumination, which greatly reduced the temporal cost in complex scenes. Our approach initially traces photon-rays with multiple cube-maps, and then gathers the irradiance of photons using uniform grids filled with low-order spherical harmonics. There are two contributions: 1) a heuristic strategy improving the efficiency of ray-cube-map intersection; 2) a grid-based algorithm that gathers all-frequency light from photons: for smooth indirect lighting (diffuse, etc.), a 3D grid covering the whole scene is used, and for high-frequency lighting (caustics, etc.), a high-resolution 2D grid covering screen is used. Variant global illumination effects can be rendered efficiently in our framework.

Keywords Photon tracing · Global illumination · Real-time rendering

1 Introduction

Ray-tracing and light-gathering are the two key steps in some advanced two-passes rendering algorithms, such as photon mapping[13,14], a technique for global illumination. It uses ray-tracing to calculate the distribution of photons, and uses light-gathering to estimate the intensity contributed by the photons. Recent works focusing on multi-image-based photon mapping [36] have shown their great efficiency in real-time. Before tracing a photon-ray, they initially pick some points in the space and render cube-maps whose centers located at these points. Each cube-map stores the geometric information (position, normal, color, and material) around. Afterwards, they cast photon-rays from light sources and sequentially intersect these rays with the objects recorded in the cube-maps. Each intersection is evaluated and a most suitable one is chosen from these candidates. The ray-tracer is built on a so-called *secant-method*[32], which first calculates an initial testing point very roughly, and then does a refinement around it. After tracing, they splat the photons onto the surfaces of objects to estimate the intensity contributed by the photons[11].

Their solution works well in a simple scene, but has serious limitations in a complex one due to three reasons: a) the coarsely computed initial testing point, which is crucial for an accurate intersection, can be far from the correct one, which may lead the refinement to be nonsense; b) when the number of cube-maps are high, the number of evaluated intersections for each photon-ray can strike the overall performance; c) the method estimating the intensity, called *splatting* [30,11,21,36], is not efficient when the number of photons is high, since a photon will be expanded into a quad that takes hundreds of pixels, and splatting millions of photons will produce billions of pixels.

In this paper, we propose an effective solution focusing on ray-tracing and intensity-estimating in complex scenes (Table 1). Our first contribution is a heuristic ray-tracing scheme that calculates the initial testing point more reasonably, providing a reliable beginning for the later refinement; afterwards only one or two cube-maps are used by each photon-ray, increasing the efficiency of ray-cube-map intersection over several times. Our second contribution is an approximate grid-based light-gathering technique for all-frequency effects. Each photon is transformed into a spherical harmonics (SH) vector, which is later stored into two types of uniform grids. One grid called light-gathering volume, or LGV, is a 3D volume used to gather smooth light such as diffuse lighting, and the other one, called

Address(es) of author(s) should be given

Fig. 1 Without pre-computation, our technique renders dynamic global illumination effects in a complex scene in real-time ($11 \sim 56Hz$ on a GTX480).

screen-space LGV (SSLGV), is a screen-space grid used to gather more detailed light such as caustics. Our solution has the following key features:

Efficiency In the ray-tracing phase, the number of cube-maps that a photon-ray intersects with is independent to the complexity of the scene; and in the gathering phase, we compute all the radiance within a small 3D texture that has tens of texels per dimension. These two improvements greatly reduce the temporal cost while maintaining the quality: the results produced with our algorithm are comparable with the ones rendered off-line, even if the scenes are complicated.

Generality As seen in Table 1, multiple features in a complex dynamical scene can be rendered in real-time, and our framework has the potential to be adaptive to some more applications besides photon mapping, which also requires ray-tracing or spatial integration.

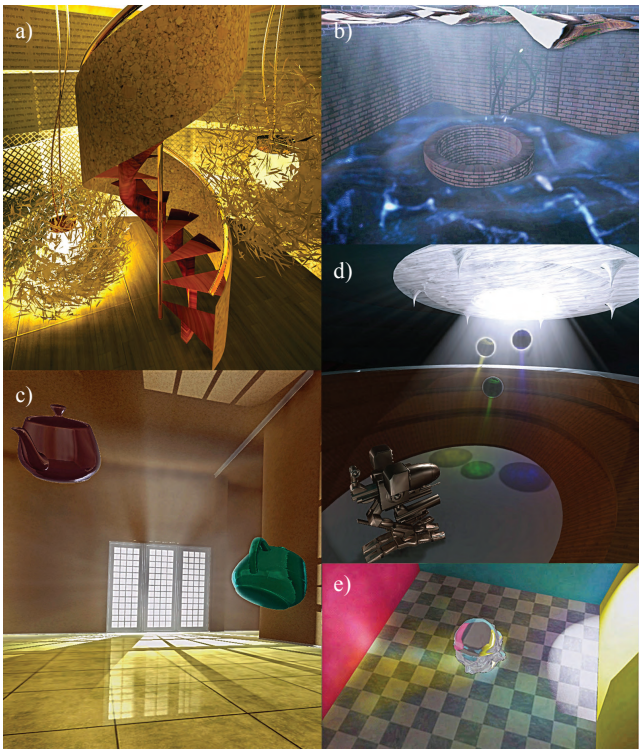

Fig. 2 Our method can efficiently handle variant effects in complex scenes. a) extremely rough hairballs, b) water caustics, c) glossy reflections, d) volume caustics, e) caustics from secondary lighting.

2 Previous Work

2.1 Real-time Ray-tracing

Massive progress has been made in the field of real-time ray-tracing. Several impressive works reached an interactive frame rate using complex acceleration data structures such as *KD-trees* [11,35], *octrees* [31] and *BVH* [7]. Unfortunately, none of them showed the capability of handling complex scenes in real-time. Alternatives were focusing on image-space ray-tracing. They rasterized each photon-ray into a row of *intersection textures* [20], or localized the ray-tracing process by intersecting rays with a single cube-map [32]. A recent work [36] extended the latter one to using multiple cube-maps. We adopt their scheme, and further improve it to be more efficient and accurate when handling scenes that are more complicated.

2.2 Intensity Estimation and Frequency Analysis

To estimate intensity at a specific pixel, some recent works have studied how to gather light efficiently[9,10, 2,3,19]. Although rendering a complex dynamic scene is still challenging, these works are remarkable milestones. Other works based on *splatting* [30,11,21,36] have also achieved great improvement but are still overwhelmed by complex scenes. Besides in the spatial domain, light transport in the frequency domain[6] has also been broadly investigated. Two frequently used basis functions are the *wavelets*[24] and the *spherical har-*

monics (SH)[25,28]. The spherical harmonics are more commonly used in the field of real-time rendering, for its simplicity and efficiency [28]. Some recent lattice-based works injected the low-order SH-approximations of secondary light into a volumetric grid floating and covering the whole scene, which is called an *Irradiance Volume*[8, 23] or a *Light Propagation Volume (LPV)*[16,17]. In the LPV, intensity from direct lighting is coarsely propagated between cells, creating indirect lighting, and the scene is rendered in a subsequent pass. Their method is embarrassingly fast. However, it is observed [18,4] that only near-field indirect lighting (where the reflecting and receiving surfaces are close to each other) can be accurately computed. We try to solve this problem by using photon tracing, and extend their lattice-based methods for gathering the indirect light contributed by photons. Moreover, inspired by the cascaded LPV [17], we introduce a screen-space SH-grid to assist the gathering for high-frequency lighting.

3 Multi-image-based Ray-tracing

In a single-image-based ray-tracer [32], rays are intersected with distance impostors recorded in an environment cube-map. Since in some practical scenes it is found that single cube-map can hardly cover all the surfaces in the scene, a multi-image-based method is proposed [36]. Intersections are computed from multiple cube-maps using the same scheme as its single-image version. To drop the possibility of false intersection, each photon-ray is intersected sequentially with each used cube-map. These intersections are evaluated by their distances to the photon, and the angles to the photon ray. After selection, only one intersection is remained.

Their method has two limitations. **First**, as the complexity of a scene increased, the number of used cube-maps will grow to be overwhelming both temporally and spatially. Besides, the image resolutions of cube-maps are limited. When the center of a cube-map is far from the point of intersection, the intersected surface will be recorded in the cube-map with just a few pixels, which will bring inaccuracy. **Second**, their solution is unstable and can lead the intersection to be wrong under complex situations, even if most cube-maps are covering the correct point.

Let's detail their method in a failure case. Regarding the example in Figure 3a, distances from the objects to V are rendered into an environment cube-map centered at V. We denote the ray originating from $\mathbf{t}$ pointing along direction $\mathbf{r}$ as $\mathbf{t} + d \cdot \mathbf{r}$, where $\mathbf{r}$ is a normalized vector and d is the distance from the starting point to the intersection. To solve for a correct d, an initial testing point is initially selected, by directly reading the distance value in the direction of the photon-ray (showed as the blue point in Figure 3a, or **VA**). Their method *assumed* that *the geometries can be approximated by some plane perpendicular to* **VA**. Under such assumption, a plane AB is made, with an intersection denoted as B. This is achieved by calculating a distance from the photon to B, denoted as dp. Then they find the intersection (denoted as B') in the direction of **VB**. A *secant*-based method[32] is then applied around B' to refine it to the exact position. Obviously, in our tested scene in Figure 3a, the geometries can *never be approximated* by AB, and the correct result (marked by a green point) is far from the fictitious one, which means the distance from photon to this plane, dp, is not correctly estimated. Yao et al. added extra cube-maps to solve this problem, however, this makes no sense since using an extra cube-maps may still fail to find the correct intersection: notice the other blue point in Figure 3a, which denotes the center of an extra cube-map; using this extra one we still get B' as the result.

To solve this failure case as well as the issue of inaccuracy mentioned above, we propose a new solution, based on two observations that: *1) the intersection point is only strongly related to the ambient objects near to the photon-ray, and 2) the existing methods failed since they did not obtain a reliable initial testing point.* To ensure a correct result, we first calculate a reliable initial testing point, and then use the map closest to this point in the refinement. In our experiment, this scheme can efficiently solve the unstable situation mentioned above and increase performance for several times (see Figure 4). We detailed our method into two stages:

1) use a cascaded partition scheme to find an intersection close to be accurate, as the initial testing point. The cube-map *closest to the photon* is chosen for intersection (Figure 3). This stage is detailed in subsection 3.1 and 3.2.

2) refine the initial testing point to an exact solution following the work proposed by Szirmay-Kalos et al.[32], using the cube-map *closest to the initial testing point.*

Our solution is stable since the initial testing point achieved from the first stage is very close to the exact solution. Furthermore, as the initial testing point need not to be numerically accurate, the partition can be accelerated with using down-level mip-maps of the cube-map.

3.1 Find Initial Testing Point using Angle Partition

Our hierarchical partition scheme for the initial testing point (ITP) is similar to a hierarchical ray-casting

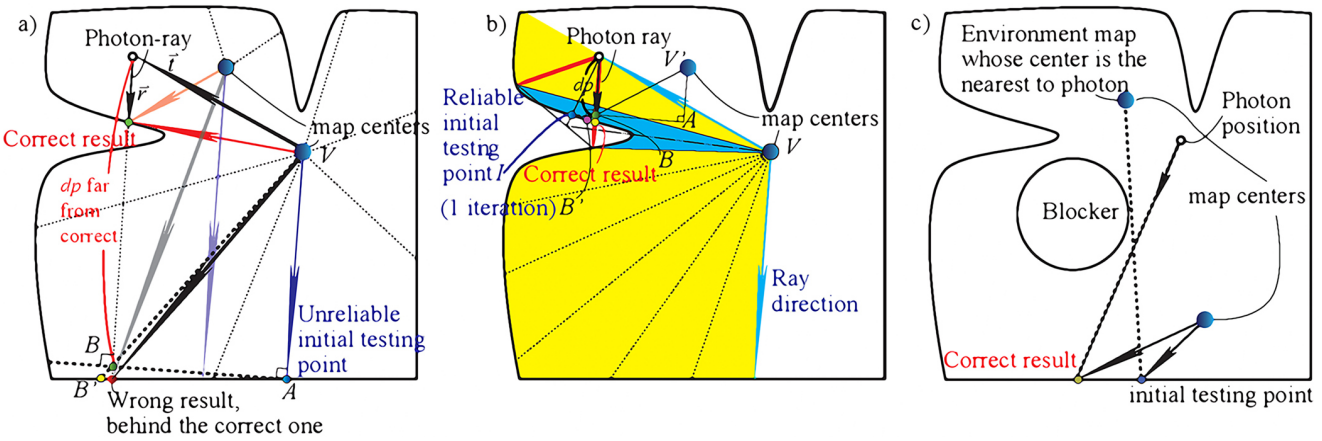

Fig. 3 a) The failure occasion of the existing method. b) The angle-partition scheme to choose an reliable initial testing point. c) The line of sight from the center of the map (nearest to the photon) to the correct result is blocked by some objects, but the correct one can still be found using the map whose center is nearest to the initial testing point in refinement.

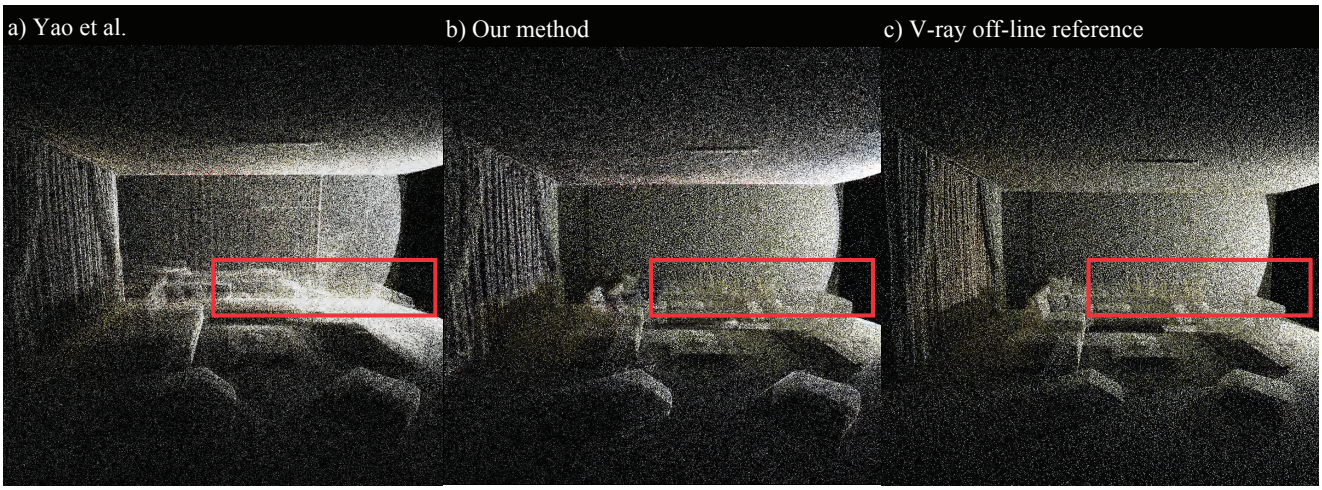

Fig. 4 The comparison of distribution of photons traced with 27 cube-maps. a) *33Hz*; b) *201Hz*; c) geometric method. Please notice the photon distribution in the red box. In (a), lots of photons that should attach elsewhere erroneously pass through the objects (the sofa) and pile up, while our method in (b) can trace photons correctly. This scene is in Figure 1a.

process but operating on cube-maps. Below we denote the photon-ray with a position vector $\mathbf{t}$ pointing from a cube-map centered at $\mathbf{V}$, and a direction vector $\mathbf{r}$. The fan-shaped *3D-planar* region folded by $\mathbf{t}$ and $\mathbf{r}$ is denoted by $\mathcal{A}$.

Our scheme is simple. Firstly we uniformly subdivide the 3D plane $\mathcal{A}$ into Q intervals. Then for each interval $\mathcal{A}_{i\in[0,Q)}$, calculate an energy function $g(\mathcal{A}_i)$ (detailed in subsection 3.2), and choose a section that has the minimal energy, since the minimal energy represents the minimal difference between our estimated ITP and the correct intersection. Then we check whether the changing from the ITP in a previous loop to the current one is sufficiently small. If it is not, we uniformly subdivide the newly computed 3D fan-shape and calculate the energy in this fan-shape; otherwise, we exit the loop.

For all the scenes in this paper, this iteration can be terminated before taking the fourth loop (with $Q = 8$). Notice that after just two passes of calculation (Figure 3b), the estimated ITP comes very close to the correct solution. We denote $\mathbf{I} = d_{\mathbf{q}}\mathbf{q} + \mathbf{V}$ as the *initial testing point*, where $\mathbf{q}$ is the geometric bisector of the fan-shaped area $\mathcal{A}$, and $d_{\mathbf{q}}$ is the distance value fetched from the cube-map in the direction $\mathbf{q}$. Then we calculate the projected length of $\mathbf{IV'}$ in the $\mathbf{r}$ direction for further refinement (denoted by *dp* below, see [32] for detail). $\mathbf{V'}$ is the center of the cube-map *nearest to the initial testing point.*

$$dp = (\mathbf{I} - \mathbf{V'}, \mathbf{r}), \tag{1}$$

where the $(,)$ is the dot product.

3.2 The Energy Function used in Angle Partition

To compute the energy function $g(\mathcal{A}_i)$, we firstly fetch two distance imposters from the cube-map, in the directions of the two boundaries of i, denoted as $\mathbf{p}_i$ and $\mathbf{p}_{i+1}$. Then we individually compute $f(\mathbf{p}_i)$ and $f(\mathbf{p}_{i+1})$ (f is defined below). With adding them up, we get the energy function: $g(\mathcal{A}_i) = f(\mathbf{p}_i) + f(\mathbf{p}_{i+1})$.

The function $f(\mathbf{p})$ is defined as a *difference volume* that represents the difference between a testing point $\mathbf{p}$ and the correct intersection. It is designed based on the two key observations below:

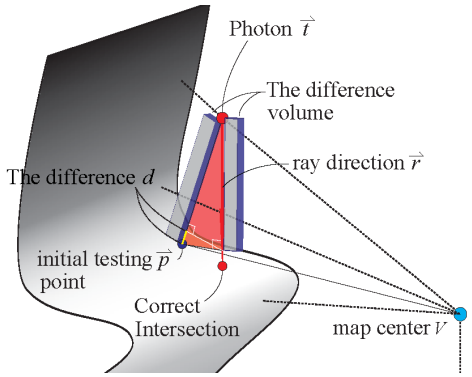

Fig. 5 The difference volume showed with translucent blue boxes.

1) the length of the vector from the position of photon $\mathbf{t}$ to a distance impostor $\mathbf{p}$ should be as close as to the length of its projection on the ray direction, specifically:

$$d(\mathbf{p}) = ||\mathbf{p}-\mathbf{t}| - (\mathbf{p}-\mathbf{t},\mathbf{r})| \approx 0 \tag{2}$$

2) both of the comparers should be small. Therefore, an addition of these two value is computed as $a(\mathbf{p}) = |\mathbf{p}-\mathbf{t}| + (\mathbf{p}-\mathbf{t},\mathbf{r})$. Then $f(\mathbf{p})$ is weighted from these two values $d(\mathbf{p})$ and $a(\mathbf{p})$, as $f = d^2a$. This is exactly the frame of a difference triangle between $\mathbf{p}-\mathbf{t}$ and its projection on $\mathbf{r}$, with a thickness of d (Figure 5).

$$f(\mathbf{p}) = d^2a = \begin{cases} (|\mathbf{p}-\mathbf{t}| - (\mathbf{p}-\mathbf{t},\mathbf{r}))^2 \times \\ (|\mathbf{p}-\mathbf{t}| + (\mathbf{p}-\mathbf{t},\mathbf{r})), & \text{if } (\mathbf{p}-\mathbf{t},\mathbf{r}) > 0, \\ \infty & \text{otherwise.} \end{cases} \tag{3}$$

When the $|\mathbf{p}-\mathbf{t}|$ gets smaller, f is determined on the square of difference between $\mathbf{p}-\mathbf{t}$ and its projection on $\mathbf{r}$, which is also the thickness of the triangle frame. This follows the first observation. When the difference between $\mathbf{p}-\mathbf{t}$ and its projection on $\mathbf{r}$ gets smaller, which means that d gets smaller, f is determined on $|\mathbf{p}-\mathbf{t}|$, the distance between the photon and its distance impostor, following the second observation.

4 The Light-gathering Volume (LGV)

In this section, we introduce our novel strategy for photon gathering. Our approach can be detailed into three phases: 1) photon injection (Figure 6a), 2) light-gathering (Figure 6b, c), and 3) scene illumination. In this section, we initially describe the photon injection method, which is our main contribution (subsection 4.1); afterwards for completeness we shortly review the light gathering and scene illumination method that we are following [17] (subsection 4.2); and then we describe how to extend our solution into screen-space for high-frequency light-gathering (subsection 4.3).

4.1 Photon Injection

The LGV is a 3D texture covering all objects in the scene. After ray-traced, photons are injected into the LGV in two stages: 1) we transform the position that a photon-ray intersects with objects into the coordinate of texel in the 3D texture, and 2) we accumulate the SH-approximation of intensity into the texel. Each texel stores the SH-approximations of the intensity contributed by injected photons. For each photon j, the intensity I_j is calculated with the following formula (the deduction is proposed in Appendix A):

$$I_j = \frac{AC_0^2}{4N}\Phi_p(\Theta_j,\mathbf{n})^+ f_r(x) \tag{4}$$

A: the total area of the scene; N: the amount of photons cast; C_0: a variant bandwidth factor [11]; Θ_j: the normalized incident direction of the photon. Φ_p: the power of the photon injected; $f_r(x)$: the BRDF around point x; $\mathbf{n}$: the normal of surface hit by the photon. The derivation of this equation is proposed in Appendix A. The intensity of photons is approximated by spherical harmonics (SH) vectors (Figure 6a). Multiple SH-vectors each of which corresponds to one photon are accumulated with additive blending. The accumulated SH-vector c_i in the cell i is denoted as:

$$\mathbf{c_i} = \sum_{j=1}^{n} I_j \mathbf{y}(-\Theta_j), \tag{5}$$

n: the number of photons located in this cell; $\mathbf{y}(-\Theta_j)$: the SH-basis vector *rotated* [28] into direction $-\Theta_j$. Since we use SH-approximation only up to two orders, $\mathbf{c_i}$ can be written as a vector of four scalars.

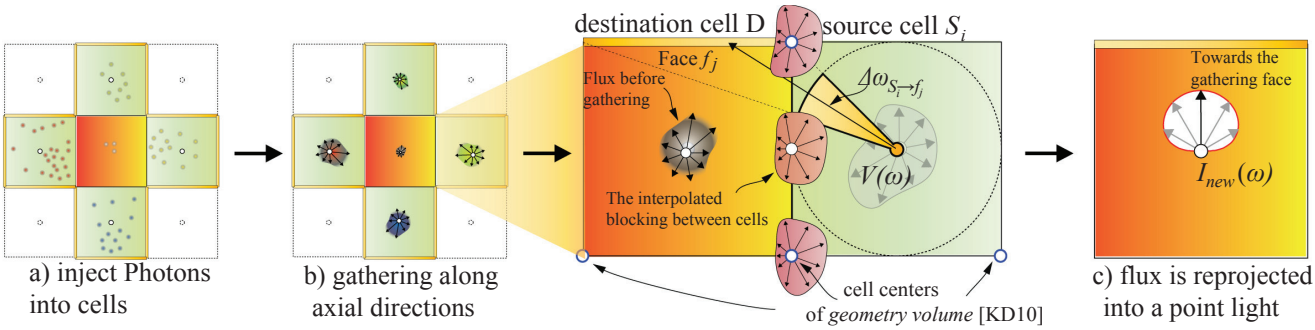

Fig. 6 The light-gathering process. a) The intensity of photons is accumulated into SH-approximations. b) Cells of LGV store the intensity of light. The cell in the center is to gather light from its neighbors. Flux is computed onto faces f_j of the destination cell D from a source cell S_i with fuzzy occlusion of geometries between the cells. c) The accumulated new intensity in the gathered cell. A part of this figure is adapted from the work proposed by Kaplanyan et al.[17]

4.2 Light-gathering and Scene Illumination

After injection, the intensity in each texel represents the local illumination around the position of the cell (in world space). Then for each cell, we gather intensity from its six neighborhoods adjacent to it:

1) compute the flux from the texels adjacent in axial, orthogonal directions. For a destination cell D, and for each adjacent cell S_i around it, we compute the flux I_{S_i} projected onto each faces f_j of cell D by estimating the integral on different orientations that $\Phi_{S_i \to f_j} \approx \Delta\omega_{S_i \to f_j} I_{S_i} G_{S_i \to D}$, where $\Delta\omega_{S_i \to f_j} = \int_{\Omega_{f_j}} V_{S_i}(\omega) d\omega$. The visibility function, is defined as $V_{S_i}(\omega) = 1$, if a ray starting at the center of S_i in direction ω intersects the face f_j directly, otherwise $V_{S_i}(\omega) = 0$. In addition, the $G_{S_i \to D}$ is a transmittance value given the blocking of geometry between S_i and D.

2) accumulate the intensity from adjacent texels, the new intensity of the current texel is computed with:

$$I_{new} = \frac{1}{\pi} \sum_{j=1}^{6} \sum_{i=1}^{6} \Phi_{S_i \to f_j} \tag{6}$$

The gathering scheme is iterated for several times (equivalent to 1/4 of the grid size in one dimension, empirically) to gather in larger radius. Light are propagated from texel to texel. As a result, the light from the rest of the scene is received by each texel and accumulated. This maps the light propagation to final-gathering: since the total light energy in the LGV never changes, and will never be artificially increased after gathering, the energy in the gathering process can be conserved, just as final-gathering does [33,29].

After gathered, the intensity at each pixel p on the screen, denoted by L_p, is computed by:

$$L_p = 4(\mathbf{c_p}, \mathbf{y}(-\mathbf{n_p}))^+ / l^2 \tag{7}$$

where $\mathbf{c_p}$ is interpolated from the values read from the texels in LGV. These texels are located around the position (in world-space) of pixel p. The position for indexing the LGV is read from the G-buffer rendered in the view of camera. $\mathbf{y}(-\mathbf{n_p})$ is SH-basis vector rotated into direction opposite to the surface normal n_p, and l is the length of a cell in LGV.

4.3 The Screen-space Light-gathering Volume (SSLGV)

High frequency details such as caustics need higher order of SH-vectors. When using a lattice filled with SH-approximation in low-order, the resolution of the grid has to be increased. This is unacceptable for both spatial and temporal considerations. Moreover, the hierarchical solution in Kaplanyan's work [17] can only solve detailed light around camera, which makes no sense for caustics since they may appear anywhere in the sight. Fortunately, the high-resolution computation is much less costly to be taken in screen-space. Therefore, we propose a screen-space grid to assist the light-gathering process.

Injection Photons are transformed into eye-space and filtered, according to whether they are caustics photons or not. Then they are injected into a 2D texture in the manner similar to LGV. To gather the photons within the gathering radius, we measure the depth difference between the photon and the pixel position, in eye-space, denoted as $d_{x \leftrightarrow x_p} = ||\mathbf{x}| - |\mathbf{x_p}||$, where $\mathbf{x_p}$ is the position of photons, and $\mathbf{x}$ is the pixel position read from G-buffer. If $d_{x \leftrightarrow x_p} < h$ for some gathering radius h, the photon is accepted and otherwise rejected. Inspired from the work of Yao et al.[36], this radius h is defined as the *variant photon splatting radius*, specifically $h = C_0 \sqrt{A/pN}$ where C_0 is a global parameter, A is the total area of the scene, N is the number of emitted photons and p is the probability density factor clamped to some boundary $[P_{min}, P_{max}]$. Afterwards,

the intensity of each accepted photon is estimated by a SH-vector and blended into the pixel.

Gathering For each pixel located in $\mathbf{p}$ in the eye-space, the SH-approximation is extracted from its 4-connected neighbor pixels located at $\mathbf{u_i}$ and is projected to the directions of $\mathbf{u_i p}$. When $|\mathbf{u_i p}| < h$ for some gathering radius h calculated same as in the injection stage, the neighboring SH-approximation is accumulated to $\mathbf{p}$.

Illumination After several times of iteration (about twelve, empirically, for a SSLGV at a resolution of 512^2), the SSLGV-texture is rendered with projection that estimates the intensity along the surface normal. The coverage area of a pixel p is calculated using a formula from the work of Yao et al.[36]:

$$A(p) = |\frac{4tan^2\frac{\alpha}{2}((\mathbf{x_p} - \mathbf{v}) \cdot \mathbf{r})^3}{N^2(\mathbf{v} - \mathbf{x_p}) \cdot \mathbf{n_p}}|$$

$\mathbf{v}$: the position of the camera; $\mathbf{x_p}$ and $\mathbf{n_p}$: the position and surface normal in p; N: the size of the SSLGV-texture in one dimension; $\mathbf{r}$: the forward direction of the camera; α: the field of view of the camera.

Our approach of SSLGV is illustrated in Figure 7: 1) during the injection of the photons, the light contributed by each photon is accumulated in the Z-axis that is perpendicular to the clipping plane of the camera (the lower left cell in Figure 7), and only the photons in some given radius around the actual position of the surface are accepted; 2) in the process of gathering, SH-approximations in the yellow cells (pixels) are accumulated into the current grey cell (pixel), and the one in the red cell (pixel) is declined for its distance to the current pixel is larger than some given radius for gathering.

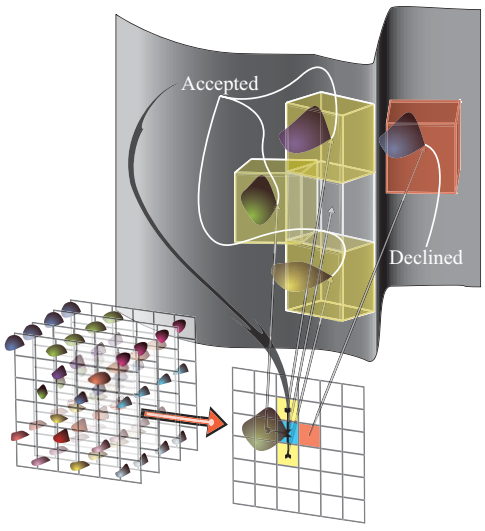

Fig. 7 Illustration of the SSLGV used in our technique.

5 Comparisons

Our experiments are performed on DirectX 11 with an *GTX 480* with 1.5GB video memory. Statistics are presented in Table 1. Compared with off-line render using classical photon mapping, our real-time method presented a more similar look than other real-time approaches (Figure 8) with a much higher efficiency. Our result is comparable to the referenced results rendered in two hours with V-Ray(Figure 8g), as well as the result rendered in 24 minutes with PBRT (Figure 8j). Complex lighting effects such as indirect caustics is well handled (Figure 2e). Glossy reflections with indirect lighting (Figure 2c) are rendered with minor cost (only a texture fetching is necessary). When integrated with some existing algorithms [12], volume light (Figure 1b, Figure 2c), or volumetric caustics (Figure 2b and d) are also plausibly rendered in real-time, which demonstrates the generality of our method.

Image-space Photon Mapping: Compared with image-space photon mapping building KD-Trees on CPU [21], our solution has no pre-computations and can free CPU for other crucial features in games (AI, scene management, etc.). Compared with existing multi-image-based photon mapping [36], our method traces photon-rays much more rapidly and accurately in average of $5 \sim 6\times$ in a complex scene (Figure 4), and provides more accurate results.

Geometry-based Ray-tracing: Compared with approaches using advanced accelerating data structures [35,10], our algorithm operates on much simpler data structures textures. This provides much convenience during practical game development. Some knotty objects with extremely tortuous surfaces such as the *hairballs*(Figure 2a) are also plausibly rendered. Besides, using no complex hierarchy structures makes our implementation simpler and more easily followed.

Discrete Ordinate Methods: Compared with existing lattice-based methods such as light propagation volume (LPV)[8,23,16,17], our solution can handle general global light transport instead of the *near-field* one (which means only the objects close to the light reflecting surfaces are affected by global illumination): in their LPV, the distance of light transport is limited by the number of iterations and grid resolutions [18,4]. In the contrary, our method benefit from photon-tracing is free from this problem.

Virtual Point Lights (VPL): Compared with VPL-based approaches [27,22,26,1], or voxel-based approaches [5,34], our approach can handle detailed lighting such as caustics (Figure 1c, d, Figure 2c), especially the indirect caustics (Figure 2e), naturally without additional techniques. Besides, we produce a higher performance

Fig. 8 Comparisons of the results. The top row is the rendering result from our method, and the bottom row is the rendering results generated by off-line renders using classical photon mapping.).

on average than the existing VPL-based approaches dedicated on large scenes[26].

6 Limitations

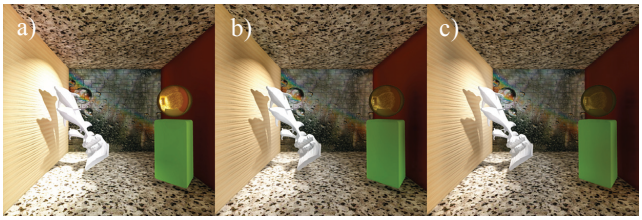

Fig. 9 a) 16^3 @ 41Hz, b) 32^3 @ 40Hz, and c) 48^3 @ 33Hz show the balance between quality and performance with different size of LGV used.

The artifacts pervasive in existing lattice-based methods are temporal discontinuity and light bleeding. One solution is to increase the resolution of LGV. Figure 9 shows the dilemma of performance and quality while changing the resolution of LGV. Indirect shadow ambiguous in the image rendered from most coarse LGV (Figure 9a) gradually becomes detailed (Figure 9c) along with the increasing of LGV size, which brings higher temporal and spatial cost. Using some cascaded LGV would partially decrease the temporal cost but not essentially. Besides, the rendering quality of some flattened "middle"-frequency caustics is still very dependent on the amount of emitted photons. Tracing tens of millions of photons is still overwhelming both spatially and temporally to real-time applications.

7 Conclusions

In this paper, an image-space ray-tracing scheme and a gathering technique are proposed, providing with greater efficiency and results comparable to the ones from off-line renders. In the future, we would like to investigate some more accurate light-gathering techniques, for instance, the ones considering detailed occlusions between objects.

References

1. Airieau, B., Bridault, F., Meneveaux, D., and Blasi, P. Photon Streaming for Interactive Global Illumination in Dynamic Scenes. *Vis. Comput. 27*, 3 (2011), 229–240.

Figure	Features	Triangles	Photons	LGV Size	Ours	Yao et al.
1a	Color bleeding, glossy reflections	477.9k	0.6M	46x37x24	56Hz	23Hz
1b	Complex occlusions, volume light	267.4k		41x42x24	22Hz	-*
1c	Caustics in a large scene	163.9k	0.81M	39x49x25	19Hz	7Hz
1d	Caustics, refractions	33.0k	2.5M	31x29x24	11Hz	5Hz
8a	Complex objects and light	372.6k		40x33x24	62Hz	3Hz
8b	Large scene	77.5k	0.6M	54x24x39	48Hz	14Hz
2a	Very involved objects	671.2k		29x29x24	31Hz	-*
2b	Volume caustics for water	31.2k	2.5M	25x24x31	11Hz	-*
2c	glossy and specular reflections	53.8k	0.6M	43x40x24	38Hz	3Hz
2d	Volume light and volume caustics, glossy reflections	69.0k	0.81M	52x43x24	16Hz	-*

Table 1 Performance Statistics. Resolutions: Results in 1, 8top-right and 2 are the same 800x800; 8top-left is in 1280x800; All SSLGVs are 512^2. All maps are 64^2. *: No volume light or caustics, or no effect of specular reflections. The LGV sizes are different in each dimension since the sizes of the geometry are different in each dimension.

2. Brouillat, J., Gautron, P., and Bouatouch, K. Photon-driven irradiance cache. *Comput. Graph. Forum 27*, 7 (2008), 1971–1978.
3. Chen, H., and Tatarchuk, N. Lighting research at bungie. In *Courses of Advances in Real-Time Rendering in 3D Graphics and Games - SIGGRAPH '09* (2009).
4. Corporation, N. Diffuse global illumination, 2011. from NVIDIA Direct3D 11 SDK.
5. Crassin, C., Neyret, F., Sainz, M., Green, S., and Eisemann, E. Interactive indirect illumination using voxel cone tracing: An insight. In *Talks of the SIGGRAPH '11* (2011).
6. Durand, F., Holzschuch, N., Soler, C., Chan, E., and Sillion, F. X. A frequency analysis of light transport. *ACM Trans. Graph. 24* (2005), 1115–1126.
7. Fabianowski, B., and Dingliana, J. Interactive global photon mapping. *Comput. Graph. Forum 28*, 4 (2009), 1151–1159.
8. Greger, G., Shirley, P., Hubbard, P. M., and Greenberg, D. P. The irradiance volume. *IEEE Comput. Graph. Appl. 18* (1998), 32–43.
9. Hachisuka, T. Final gathering on gpu. In *Proceedings of the GPGPU '04* (2004).
10. Havran, V., Herzog, R., and Seidel, H.-P. Fast final gathering via reverse photon mapping. *Comput. Graph. Forum 24*, 3 (2005), 323–332.
11. Herzog, R., Havran, V., Kinuwaki, S., Myszkowski, K., and Seidel, H.-P. Global illumination using photon ray splatting. *Comput. Graph. Forum 26*, 3 (2007), 503–513.
12. Hu, W., Dong, Z., Ihrke, I., Grosch, T., Yuan, G., and Seidel, H.-P. Interactive volume caustics in single-scattering media. In *Proceedings of the I3D '10* (2010), pp. 109–117.
13. Jensen, H. W. Global illumination using photon maps. In *Proceedings of the EGSR '96* (1996), pp. 21–30.
14. Jensen, H. W. *Realistic image synthesis using photon mapping*. 2001.
15. Kajiya, J. T. The rendering equation. In *Proceedings of the SIGGRAPH '86* (1986), pp. 143–150.
16. Kaplanyan, A. Light propagation volumes in cryengine 3. In *Courses of Advances in Real-Time Rendering in 3D Graphics and Games - SIGGRAPH '09* (2009), pp. 1–45.
17. Kaplanyan, A., and Dachsbacher, C. Cascaded light propagation volumes for real-time indirect illumination. In *Proceedings of the I3D '10* (2010), pp. 99–107.
18. Kirsch, A., and Chajdas, M. G. Light propagation volumes, 2010. Lab course at http://blog.blackhc.net/tag/light-propagation-volume/.
19. Knauer, E., Bärz, J., and Müller, S. A hybrid approach to interactive global illumination and soft shadows. *Vis. Comput. 26* (2010), 565–574.
20. Krüger, J., Bürger, K., and Westermann, R. Interactive screen-space accurate photon tracing on gpus. In *Proceedings of the EGSR '06* (2006), pp. 319–330.
21. McGuire, M., and Luebke, D. Hardware-accelerated global illumination by image space photon mapping. In *Proceedings of the HPG '09* (2009), pp. 77–89.
22. Novák, J., Engelhardt, T., and Dachsbacher, C. Screen-space bias compensation for interactive high-quality global illumination with virtual point lights. In *Proceedings of the I3D '11* (2011), pp. 119–124.
23. Oat, C. Irradiance volumes for real-time rendering. In *ShaderX5: Advanced Rendering Techniques*. 2006.
24. Overbeck, R. S., Donner, C., and Ramamoorthi, R. Adaptive wavelet rendering. *ACM Trans. Graph. 28* (2009), 140:1–140:12.
25. Ramamoorthi, R. *A signal-processing framework for forward and inverse rendering*. PhD thesis, 2002.
26. Ritschel, T., Elmar, E., Han, I., D. K. Kim, J., and Seidel, H.-P. Making imperfect shadow maps view-adaptive: High-quality global illumination in large dynamic scenes. *Comput. Graph. Forum 30*, 4 (2011), 2258–2269.
27. Ritschel, T., Grosch, T., Kim, M. H., Seidel, H.-P., Dachsbacher, C., and Kautz, J. Imperfect shadow maps for efficient computation of indirect illumination. *ACM Trans. Graph. 27* (2008), 129:1–129:8.
28. Sloan, P.-P. Stupid spherical harmonics (sh) tricks. In *Lectures of the GDC '08* (2008), pp. 1–41.
29. Spencer, B., and Jones, M. W. Hierarchical photon mapping. *IEEE Trans. Vis. Comput. Graph. 15* (2009), 49–61.
30. Stürzlinger, W., and Bastos, R. Interactive rendering of globally illuminated glossy scenes. In *Proceedings of the EGSR '97* (1997), pp. 93–102.
31. Sun, X., Zhou, K., Stollnitz, E., Shi, J., and Guo, B. Interactive relighting of dynamic refractive objects. *ACM Trans. Graph. 27* (2008), 35:1–35:9.
32. Szirmay-Kalos, L., Aszódi, B., Lazányi, I., and Premecz, M. Approximate ray-tracing on the gpu with distance impostors. *Comput. Graph. Forum 24*, 3 (2005), 695–704.
33. Tawara, T. *Efficient Global Illumination for Dynamic Scenes*. PhD thesis, 2006.
34. Thiedemann, S., Henrich, N., Grosch, T., and Müller, S. Voxel-based global illumination. In *Proceedings of the I3D '11* (2011), pp. 103–110.

35. Wang, R., Wang, R., Zhou, K., Pan, M., and Bao, H. An efficient gpu-based approach for interactive global illumination. *ACM Trans. Graph. 28* (2009), 91:1–91:8.
36. Yao, C., Wang, B., Chan, B., Yong, J., and Paul, J.-C. Multi-image based photon tracing for interactive global illumination of dynamic scenes. *Comput. Graph. Forum 29*, 4 (2010), 1315–1324.

A The math behind photon injection

Consider the intensity over a hemisphere along some normal at pixel x visible from the camera, denoted as I below[15]:

$$I = \int_{\Omega^+} f_r(x) L(x \leftarrow \Theta) cos\theta d\omega_\theta, \tag{8}$$

Where Ω_x is the surface angle visible from any point of a hemisphere at a given point x; $f_r(x)$ is the BRDF; Θ is the incident direction; $d\omega_\theta$ is the solid angle of incident direction, where attaches the incoming light. Using the terms of SH-approximation [28], this integral can be rewritten into:

$$I = \sum_{i=0}^{n^2} c_i y_i \tag{9}$$

where y_i is the SH-basis. Moreover, we have:

$$c_i = \int_{\Omega} f_r(x) L(x \leftarrow \Theta) cos\theta y_i(-\Theta) d\omega_\theta \tag{10}$$

represents the re-projection of SH-approximation. This integral can be estimated by:

$$c_i \approx \sum_{j=1}^{n^2} f_r(x) L(x \leftarrow \Theta_j) \Delta\omega_j (\Theta_j, \mathbf{n})^+ y_i(-\Theta_j), \tag{11}$$

where Θ_j is the incident direction of the j-th photon, and is just used as the sample angle (so the θ in Equation 10 is substituted). $\mathbf{n}$ is the surface normal, while $\Delta\omega_j$ is the subtended solid angle respectively. The flux emitted from the photon is necessary to be converted into radiance. The emitted intensity of a photon j is:

$$I_{p_j}(\omega) = \frac{\Phi_p}{\pi}(\Theta_j, \mathbf{n})^+ = A(\Theta_j, \mathbf{n})^+ \int_{\Omega} L(p \rightarrow \Theta_j) cos\theta d\omega_{\Theta_j} \tag{12}$$

$$= \Phi_p(\Theta_j, \mathbf{n})^+ \frac{\Delta\omega_{p_j \rightarrow x}}{4\pi^2}. \tag{13}$$

Therefore, the light (or radiance) contributed by a photon $L(x \leftarrow \Theta_j) = dI_p/d\omega \approx \Phi_p/4\pi$. Inject this into Equation 11:

$$c_i \approx \sum_{j=1}^{n^2} \frac{\Phi_p}{4\pi} f_r(x) \Delta\omega_j (\Theta_j, \mathbf{n})^+ y_i(-\Theta_j). \tag{14}$$

Moreover, the subtended solid angle can be evaluated as $\Delta\omega_j = \Delta A_j((\Theta_j, \Upsilon_j)^+)/r^2$ where Υ_j is the vector from the centroid position of the injected cell pointing to the photon position. ΔA_j is the differential area of surface where the photon is located before the last tracing, and r is the transportation distance of photon. A density estimation formula[36] is used to aid the computation of the differential area:

$$h = dC_0 \sqrt{\frac{A}{N} cos\theta}, \tag{15}$$

where h is the radius of influence of a splatted photon computed in the same way as previous works [11], A, the total area of the scene; N, the amount of photons cast; C_0, a variant bandwidth factor and d, the traveled distance from where the photon is located before to current position. With $cos\theta = (\Theta_j, \Upsilon_j)^+$, ΔA_j is computed as: $\Delta A_j = \pi h^2 = d^2 A C_0^2/(N(\Theta_j, \Upsilon_j)^+)$, with $d = r + (\Theta_j, \Upsilon_j)$:

$$\Delta\omega_j = \frac{\pi(r + (\Theta_j, \Upsilon_j))^2 A C_0^2 (\Theta_j, \Upsilon_j)^+}{r^2 N (\Theta_j, \Upsilon_j)^+} \tag{16}$$

With canceling some terms: $\Delta\omega_j = (\pi(r+(\Theta_j, \Upsilon_j))^2 A C_0^2)/r^2 N$, and because $(\Theta_j, \Upsilon_j) << r$ when we use a finer grid, we have the formula Equation 4 and Equation 5:

$$c_i \approx \sum_{j=1}^{n} \frac{A C_0^2}{4N} \Phi_p(\Theta_j, \mathbf{n})^+ f_r(x) y_i(-\Theta_j) \tag{17}$$