Xie GF, Sun X, Wang WC. Interactive Depth-of-Field Rendering with Secondary Rays. JOURNAL OF COMPOUTER SCIENCE AND TECHNOLOGY : 1–

Interactive Depth-of-Field Rendering with Secondary Rays

Guo-Fu Xie[1,2] (谢国富), *Member, ACM*, Xin Sun[3](孙鑫), *Member, ACM*, and Wen-Cheng Wang[1](王文成), *Member, CCF*

[1] *State Key Laboratory of Computer Science, Institute of Software, Chinese Academy of Sciences, Beijing, 100190, China*
[2] *University of Chinese Academy of Sciences, Beijing, 100049, China*
[3] *Microsoft Research Asia, Beijing, 100080, China*

E-mail: guofu@ios.ac.cn, sunxin@microsoft.com, whn@ios.ac.cn

Received May 6, 2012

Abstract This paper presents an efficient method to trace secondary rays in depth-of-field (DOF) rendering, which significantly enhances realism. Till now, the effects by secondary rays have been little addressed in real-time/interactive DOF rendering, because secondary rays have less coherence than primary rays, making them very difficult to handle. We propose novel measures to cluster secondary rays, and take a virtual viewpoint to construct a layered image-based representation for the objects that would be intersected by a cluster of secondary rays respectively. Therefore, we can exploit coherence of secondary rays in the clusters to speed up tracing secondary rays in DOF rendering. Results show that we can interactively achieve DOF rendering effects with reflections or refractions on a commodity graphics card.

Keywords interactive rendering, depth-of-field effects, reflections, refractions, clustering

1 Introduction

Depth-of-field (DOF) dramatically improves photorealism and plays an important role in creating photographic effects. In rendering the DOF effects seen through reflectors or refractors, the rays from the camera are reflected or refracted before arriving at diffuse surfaces, and so their light paths are very complicated, and the visual sharpness of the diffuse objects is determined not only by the focal length of the camera but also by the distance and the curvatures of reflectors or refractors. Therefore, to obtain high quality DOF effects, many more samples per pixel are required. The cost of tracing rays is even more expensive when secondary rays are taken into account.

Tracing a large quantity of rays with complicated light paths is a very difficult task. For this, much effort has been made, and existing DOF methods can efficiently handle primary rays and achieve impressive effects. However, it is difficult to extend their methods straightforwardly to handle secondary rays, since the directions of secondary rays are always distributed irregularly. Image space methods [1–5] using the depth map to determine the blur of each pixel should work well for primary rays, but they cannot use the depth map to represent depth variation of secondary rays and so are unable to generate the DOF effects through reflectors or refractors. As for methods to trace rays in the object space [6, 7], they are always expensive and in general used for offline rendering. Recently, Lee et al. [8, 9] proposed to decompose the scene into multiple depth layers to trace rays for computing the DOF effects. Thus, they can make use of ray coherence and the high computation power of the graphics processing unit (GPU) to achieve real-time rendering of DOF effects. As the depth layers of the scene are constructed by the original viewpoint, the representation may ignore

This work was partly supported by the National Natural Science Foundation of China (Project Nos.60773026, 60833007).

many objects that would be intersected by secondary rays but not visible to the camera and lack occlusion information by reflection or refraction between objects. Thus, it is not suitable to tracing secondary rays. Though distant environmental lighting ignoring visibility can take effect in rendering the DOF effects by a simple reflection [8], the complex illumination computation by reflection or refraction between objects in the scene has not been addressed in real-time/interactive DOF rendering.

In this paper, we present a novel method to efficiently handle the one-bounce reflection rays or refraction rays in DOF rendering. For the rays shot from the camera, their related secondary rays generally have their directions distributed irregularly, and so are always very expensive to trace. However, secondary rays may have relatively high coherence in local areas of reflectors or refractors. Thus, we classify reflectors or refractors to make secondary rays from a cluster of reflectors or refractors share much coherence in finding their intersections with the scene. For each cluster, we set a virtual viewpoint to construct a representation of depth layers for the objects that would be intersected by secondary rays from the cluster of reflectors or refractors. To efficiently utilize GPUs, the representation is optimized to be compact, which reduces the storage requirement and computation on the intersection test. Results show that our method can interactively render the DOF effects with secondary rays on a commodity graphics card, though a large quantity of rays are traced, e.g. 100 sample rays for each pixel from the camera.

In the remainder of the paper, we first review previous work about DOF rendering in Section 2, then present our new method with secondary rays handled in DOF rendering in Section 3. Afterwards, we present and discuss experimental results in Section 4, and give conclusions in Section 5.

2 Related Work

Simulating the DOF effects with full light paths is too computationally expensive. With regard to this, Cook et al. [6] proposed a distributed ray tracing method by densely sampling on the lens. Then, the method was improved by the accumulation buffer technique [7] to use graphics hardware, where a collection of pinhole-based images are rendered, and averaged to generate the final result. Furthermore, such a solution is taken to simulate very complicated camera effects such as a combination of DOF, motion blur and reflections [10]. Hou et al. [11] constructed spatial hierarchies on GPUs to accelerate tracing rays, which can achieve high quality results as the same as CPU-based methods. However, these methods usually cost much time and in general are used for offline rendering.

To reduce the time cost for DOF rendering, the physical process is simplified and it is always assumed that primary rays do not reflect or refract at all. Based on these, the circle of confusion (COC) is regarded as a standard disk on the image whose radius is totally determined by its distance to the lens. So rendering an image with DOF becomes a problem of accumulating the disks of all pixels of the image. The COC can be efficiently accumulated with scatter or gather methods. Scatter methods [12] distribute source pixels in a pinhole image onto COC sprites. Then, the sprites are blended from far to near. However, this method requires heavy processing for depth sorting, even on modern GPUs. The technique is usually used in offline applications. Gather methods [1–5] simulating blurring of pixels by spatially filtering the image can utilize the texture lookup capability of recent GPUs to achieve high performance, where the amount of blur is determined by the COC size. But these spatial blur methods often cause intensity leakage and depth discontinuity because it is difficult to determine the occlusion of a COC. As for the methods to execute rasterization on GPUs [13] or employing light field rendering techniques [14], they also suffer from occlusion problems.

To cope with intensity leakage and the lack of partial occlusion information, multilayer approaches decompose a pinhole image into layers according to the depth of pixels, and then have the layers blurred separately and composited with alpha blending, where a layer is blurred with various techniques such as a Fourier Transform [15],

pyramidal image processing [16], anisotropic diffusion [17, 18], splatting [19], or rectangle spreading [20]. However, such approaches are approximate and cannot recover true scene information, which may lead to overly-blurred and incorrect results, and the discretization artifacts can degrade image quality when objects span more than one layer. To address such a problem, a few methods have been proposed such as special image processing [21], information duplication [16], and depth variation [19]. But they are only suitable for primarily diffuse surfaces, and cannot handle perfectly specular materials, owing to the lack of a ray tracing framework. Recently, Lee at al. [8] proposed to decompose the scene into depth layers and use image-space ray tracing techniques to rapidly compute DOF, and later improved this method [9] by a more compact image-based layered representation and a more efficient ray tracing algorithm. They are able to achieve impressive results and generate specular reflections of distant environmental lighting. However, they cannot handle reflections or refractions between objects of the scene, because their representation by layers is only constructed according to the original viewpoint of the camera.

Recently, some methods proposed to get high quality results with a low sampling rate, which is an important approach to save computation. However, they cannot efficiently handle secondary rays in DOF rendering. Chen et al. [22] used the adaptive sample density, which is determined by the amount of blur and pixel variance, and a multiscale reconstruction filter to reduce the noise in the defocused areas. However, the multiscale filter relies on the depth map, which may fail to handle specular materials. Lehtinen et al. [23] proposed to improve the image quality by exploiting the anisotropy in the temporal light field and permitting efficient reuse of samples between pixels. Though it can generate glossy effects under distant environment lighting by utilizing additional bandwidth information, it assumes the radiance does not vary along a sample's trajectory, which cannot hold for specular materials. Laine et al. [24] tried to avoid processing the samples that fall outside the computed bounds, to increase the efficiency of stochastic rasterization of defocus blur with the low sampling densities. However, this method cannot be extended straightforwardly to treat secondary rays, owing to the rasterization framework. Ragan-Kelley et al. [25] introduced decoupled sampling to reduce the expensive shading cost, which decouples the shading rate from visibility sampling for defocus blur in graphics pipelines. It is based on the assumption that a scene point's color is roughly constant from all views on the lens. However, for reflection or refraction, different views on the lens may have different directions, so that this method is not suitable for secondary rays.

Some methods have been proposed to speed up tracing secondary rays. Roger et al. [26] used a cone-sphere to build the hierarchical structure of secondary rays to accelerate tracing. However, owing to computational complexities, it is generally very difficult for the method to handle a large quantity of irregular secondary rays. Rosen et al. [27] approximated scene geometry with more efficient representations using non-pinhole cameras to accelerate computing reflections and refractions. However, it does not take self-reflection of reflectors into account. Our method is to exploit the coherence of secondary rays by a clustering scheme to achieve high performance. Popescu et al. [28] constructed sample-based cameras from the reflected rays to accelerate tracing multi-bounce reflection rays. It can only handle convex reflectors and be not easy to deal with concave reflectors with high complexity.

3 Our DOF Rendering with Secondary Rays

We develop our method partly motivated by the work of [9]. It represents the scene by layered images which are constructed from the viewpoint. Then, it introduces an efficient image-space ray tracing method to sample many rays on the lens for each pixel to perform DOF rendering. However, its representation is corresponding to the original viewpoint of the camera, and so may ignore many objects that would be intersected by secondary rays but are not visible to the camera and lack occlusion information by reflections or re-

fractions between objects. And also this representation is not efficient for tracing secondary rays, because these rays generally have their directions distributed irregularly. Thus, it is not suitable for tracing secondary rays by reflecting or refracting primary rays from the camera.

As we know, image based ray tracing is very efficient when the rays are almost perpendicular to the layered images. But this is not true for irregular secondary rays. However, for secondary rays with similar normal directions and close positions, they have much coherence to share. Thus, we classify reflectors or refractors of the scene for speeding up tracing secondary rays, where we construct a compact layered image representation for each cluster of reflectors or refractors with a virtual viewpoint. Therefore, in DOF rendering, for each cluster, we first get the cluster of reflectors and refractors for primary rays, and then take its corresponding layer representation to trace secondary rays in the corresponding virtual viewport respectively. The construction of a layered image representation of the specular rays is quite different from the representation of primary rays, because many pixels may not be intersected by the specular rays of the given cluster. Our method is different from the method of [9], as we can further cull these pixels to greatly improve the efficiency of storage and intersection. An overview of our method is illustrated in Fig. 1 by reflectors.

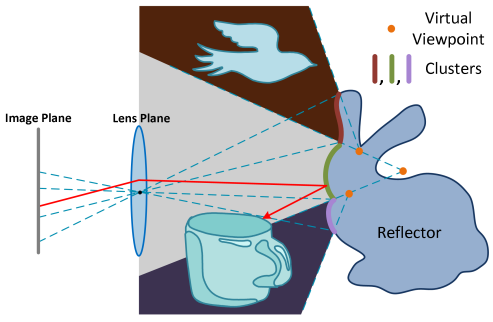


Fig. 1. Overview of our method.

As the measures for constructing the layer representation and tracing rays via the representation are the same for reflectors and refractors, in the following description we will introduce the measures only by reflectors for simplicity.

3.1 Review of Multiview Synthesis

Our ray tracer is similar to the framework of Lee et al. [9]. This method consists of two parts, constructing a representation of the scene with layered images and tracing rays by the representation. In constructing the representation, to get a compact one, it uses an extended-umbra peeling method to early cull some pixels, which no ray can intersect with through the lens. In finding the scene intersections of a ray, it steps iteratively over the footprint, which contains all pixels in the layer's image plane underneath the 2D projection of the ray, until the intersection point is obtained. To reduce the search region, they compute the minimum and maximum depths of the layer using mipmapping techniques and N-buffers [29]. Meanwhile, they handle all lens rays in parallel to reduce the cost of computing the search region. As a whole, it works in the following steps. At first, they intersect all rays for a given depth interval to generate an approximate footprint. Given this footprint, they compute the min/max depth values as a narrow depth interval by using N-buffers. Then, all rays are clamped in the new depth interval and a new footprint is generated. In general, a footprint of moderate size can be obtained after three iterations. After that, they pack four depth values into a single RGBA texture directly to process four layers in parallel.

Because lens rays are almost perpendicular to the image plane, they have very high coherence. Thus it can efficiently utilize the GPU to perform DOF rendering in real time.

3.2 Efficient Classification

Our method is for speeding up tracing secondary rays in DOF rendering. At first, we generate the image-based layered representation of reflectors from the original viewpoint by extended-umbra peeling [9]. Then, the pixels of reflectors in the original layered images are classified into clusters by the positions and reflection directions of these pixels.

Generally, the k-means algorithm can be used for classification. However, if it is used to treat all reflectors at a time, it needs much time. The main

bottleneck is finding the nearest cluster center for each pixel. Meanwhile, the resulting clusters cannot guarantee that the specular rays from a cluster have very similar directions. Considering this, we take the following steps to efficiently classify reflectors and bound the directions of the specular rays, which make the directions of the specular rays of a cluster not deviate much from the viewing direction of the virtual viewpoint for the cluster.

1) The pixels of the reflectors can be quickly sorted into each facet of the unit regular polyhedron by the directions of their related specular rays, such as the 12 facets of the unit regular dodecahedron. With this, it is helpful to limit the range of the directions for the specular rays in a cluster.

2) The standard k-means algorithm is used to treat the pixels of the facets of the unit regular polyhedron respectively. Here, for the pixels of the same facet, they are classified by the Euclidean distances of the positions between the pixels. In our experiments, we separate the pixels of each facet into 1 to 3 clusters to make the final clusters have as similar numbers of pixels as possible.

Fig. 2. Comparison between the standard k-means algorithm to the reflector sphere and our modified clustering method to generate 24 clusters. (a) The result by using the k-means algorithm to handle the reflector by the positions and the directions of the reflections. (b) 12 clusters of reflection directions on a unit regular dodecahedron. (c) The clustering result by our modified method. The images are of 512×512 pixels. Using the standard k-means to the reflector takes 0.618 seconds, while our method takes 0.013 seconds.

In our implementation, for simplicity, we first compute an approximate direction range of each facet of the unit regular polyhedron to include all directions of each facet, then determine the corresponding facet which each pixel belongs to. If the direction of a specular ray of a pixel is in the direction range of the facet, the pixel is assigned to the facet. If not, we detect the direction with next facet iteratively. Though each facet is detected respectively, these specular rays can be quickly handled in parallel. As illustrated in Fig. 2, our modified clustering method can run faster and get a better clustering result. It limits the angle range of the viewing direction of the virtual viewport and the directions of the specular rays for each cluster, especially in the boundary of the sphere in Fig. 2, which makes image based ray tracing very efficient for handling this kind of specular rays. However, the resulting clusters using the k-means algorithm make the specular rays deviate a lot from the viewing direction of the virtual viewpoint, lowering the efficiency on ray tracing.

3.3 Constructing Compact Representation

To efficiently trace the specular rays from a cluster of reflectors, we should have the virtual viewpoint set well and compactly construct the layered image representation for the objects that would be intersected by the specular rays. Because many pixels can be determined that they cannot be intersected by the specular rays of the given cluster, we can further cull these pixels to greatly improve the efficiency of storage and intersection.

In setting the virtual viewpoint for a cluster of reflectors, we first average the specular ray directions of the pixels of the reflectors in this cluster and take it as the viewing direction of the virtual viewpoint. Then, from the point with the averaged position of the pixels in this cluster, we move it along opposite viewing direction of the virtual viewpoint until it can have all the pixels in the cluster in its viewing frustum. Thus, we get the position of the virtual viewpoint for this cluster.

After the virtual viewpoint is determined, we can construct the image layered representation for this cluster. We use adaptive bucket depth peeling [30] instead of Lee's method [9] for two reasons. First, it reduces the peeling from multiple passes to only two passes. Second, because the virtual viewpoints are much closer to the objects than the

original viewpoint, it is not very efficient to cull pixels in the layered images representation with Lee's method [9]. As pointed out by Lee et al. [9], it is more crucial to reduce the number of layers and pixels for accelerating the ray tracing step. Therefore, we try to cull the layer pixels that cannot be reached by any specular rays of a cluster to have a compact representation. For this, we adopt the following three measures.

Firstly, we set the near clipping plane at the closest position of the cluster to the virtual viewpoint. This is to exclude the objects between the virtual viewpoint and the corresponding reflectors. This kind of object will never be intersected with the specular rays from this cluster because these specular rays are shot from reflectors, not virtual viewpoints. It would require much more computation for the intersection test if we include them in our layered representation.

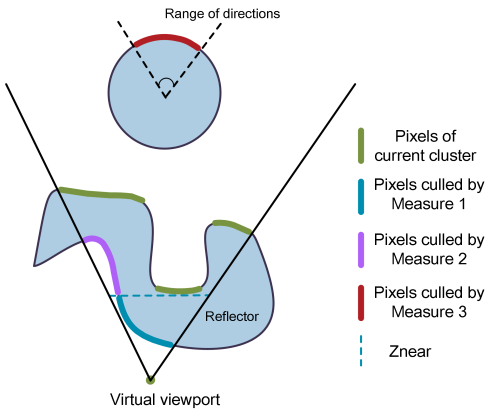


Fig. 3. Using three measures to cull the pixels in a virtual viewport.

Secondly, we can further cull the objects between the cluster and the virtual viewpoint for each pixel. This is finished by postprocessing the layered representation. After the adaptive bucket depth peeling is used to produce the initial layered representation, we abandon all pixels closer to the virtual viewpoint than the pixel of the cluster in the same position of image space.

Thirdly, we cull the pixels whose normal directions are in the range of the directions of the specular rays of the cluster. As we mentioned in Section 3.2, each cluster holds a limited range of directions of the specular rays, so the specular rays cannot be intersected with the objects whose normal directions are in the range of directions of these specular rays. This is similar to the usual back face culling but the difference is that the culling is determined by a range of the directions instead of a single direction. Although it is not an exact measure, the range of the directions is small, which does not cause artifacts in our experiments and can reduce more pixels to accelerate tracing.

The three measures are shown in Fig. 3. The storage requirements for the representations of the clusters can be reduced about 30%∼60%, when the above measures are used in our tests. In practice, we can use 8 depth layers to get very good results for most scenes.

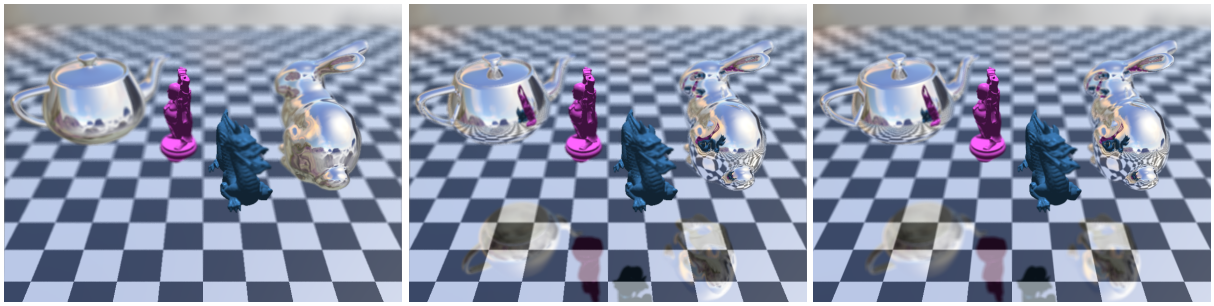
3.4 Rendering

We render primary rays by the method in [9], by which we trace 100 rays from each pixel of the result image to get high quality results for the DOF effects. For such a large number of rays, it is very difficult to trace the specular rays after all primary rays are finished tracing on the GPU, because the limited GPU memory is not enough to store the intersection information of primary rays for tracing all the specular rays. To use the computation power of the GPU and avoid its limitation on storage, we take the following steps to quickly trace the specular rays.

```
// Input the viewport information. Get reflection result.
void TracingSpecularRays (in viewInfo, out refResult)
    Construct layered representation from the viewpoint;
    Classify the reflectors into clusters;
    foreach cluster
        Construct layered image by current virtual viewpoint;
        Generate the specular rays by tracing primary rays
            from the pixels lying in the cluster;
        Use mipmapping and N-buffers to reduce the size
            of the footprint of these specular rays;
        Again the generate specular rays from primary rays;
        Trace these specular rays for the cluster;
    end foreach
```

Fig. 4. Pseudocode for tracing the specular rays.

We should determine which cluster each specular ray of each pixel belongs to respectively before tracing the specular rays in Step 1. As the original layered image representation is generated by



(a) Lee et al. Method (b) Our method (c) GPU-based ray tracing

Fig. 5. Comparison with Lee et al. Method and GPU-based ray tracing.

the primary rays through the center of the lens, the other primary rays in the same pixel of the result image with different sampling positions on the lens may intersect different pixels in the original layered images, which may be in another cluster. We take another pass only for all primary rays traced at the beginning, based on which we can assign the pixels of the result image to corresponding clusters. Most of the pixels with reflection belong to only one cluster, but the pixels close to the boundary of the clusters always belong to multiple clusters. We record this kind of information in a texture.

Clearly, in the above steps, primary rays would be traced multiple times, leading to a little more time consumption. However, owing to the limitation of GPU memory and texture bandwidth, the rendering efficiency will be reduced more if we store the intersection information of primary rays instead of tracing primary rays repeatedly.

In tracing the specular rays from a cluster of reflectors, we also use the tracing techniques in [9]. When a specular ray intersects with diffuse surface, it shades from the color of the surface. If it does not intersect with any object, it shades from the environment lighting. When it intersects with a reflector, the reflection direction of the specular ray is computed according to the normal direction of the intersection point, and then is shaded from the environment lighting.

When the lens radius is large, the specular rays of each pixel may diverge greatly, leading to a large depth interval. For this, we divide the lens disk to 4 or 16 parts, and compute their depth intervals respectively. This can efficiently reduce depth intervals and improve performance.

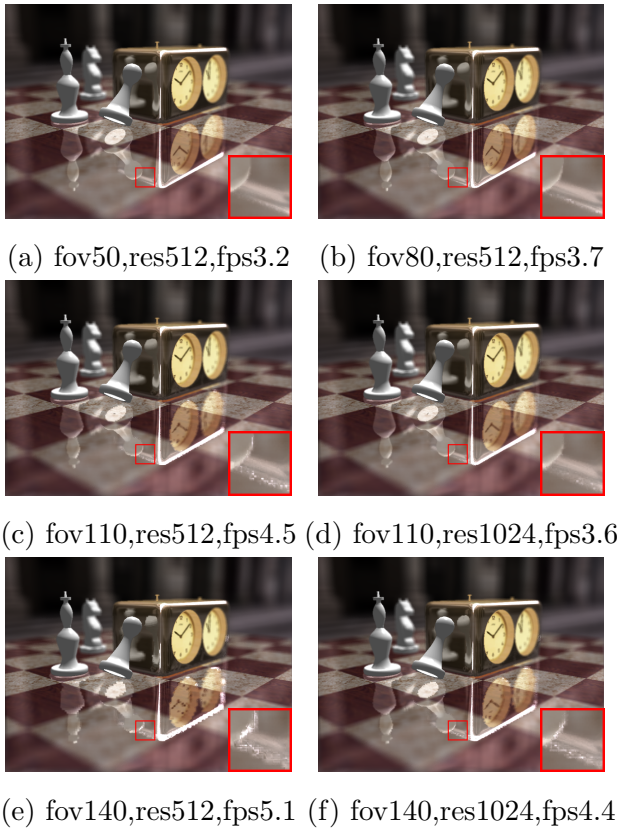
4 Experiments and Results

The clustering algorithm is implemented in NVIDIA CUDA, while the image-based ray tracing is based on OpenGL and the CG 3.0 shading language. All of our results are generated on a commodity PC with Intel Duo Xeon 2.9G Hz and 4GB memory, and an NVIDIA GeForce GTX 285. In this paper, the resolution of the images is 800×600. For each pixel, we shoot 100 rays using jittered sampling patterns on the lens. In clustering, we first distribute the directions over the facets of the unit regular polyhedron, and then classify the pixels whose directions are in a same facet into 1 to 3 clusters, until the final clusters have similar numbers of pixels. So there are usually 20 to 30 clusters in our experiments. Following Lee's method [9], we use 4 layered images with the same resolution as the rendered image for primary ray tracing. Since the virtual viewpoints are much closer to the objects than the original camera, we need more layers to represent the scenes for secondary ray tracing to avoid intensity leakage. By tests, we find 8 layers with a resolution of 512×512 are enough for most scenes. Therefore, we produce an image with 8 depth layers as the representation for a cluster by a virtual viewpoint. Consequently, the memory consumption in our experiments is about 80 MBytes.

We compared our method with Lee et al. [9] method and GPU-based ray tracing [11]. Although Lee et al. [9] has real-time DOF rendering, it only generates reflection effects through distant envi-

ronmental lighting ignoring visibility. Our method can interactively achieve reflection effects between objects, for example the interreflection between teapot and ground shown in Fig. 5. GPU-based ray tracing [11] can produce high quality results by tracing rays individually. As shown in Fig. 5, our rendering quality is comparable, even around the boundary of occlusions. As for the performance, our method does not depend much on the complexity of geometry, so that it can outperform the GPU-based ray tracer, especially in dealing with large scenes. As listed in Table 2, for the scene in Fig. 5, the performance of our method is 1.8 fps, while [11] is only 0.11 fps. Our method is much faster and is interactive.

The performance of our method is dependent on the image resolutions and sampling rate. The field-of-view (FOV) of the virtual viewport is an important factor for our performance and quality. When the FOV is larger, the intersection tests will be faster but with a lower quality, due to a lower sampling rate. On the contrary, when the FOV is smaller, the quality would be better but intersection tests will be slower. Generally, setting the FOV between 80 and 100 degrees is a good trade-off for performance and quality by our tests. For the large FOV, we can increase the sampling rate by using a higher resolution image representation to improve image quality. But it will increase memory consumption and decrease the performance. Comparison data is displayed in Fig. 6. The chessboard in Fig. 6 is assigned a perfect mirror-like specular reflection and a diffuse texture. We can see that the checkerboard texture and the reflected objects usually show different degrees of blurring. With the FOV gradually increasing, the quality gradually declines, especially in the boundary of the reflected objects on the chessboard in Fig. 6 (e). By using a higher resolution such as 1024×1024, the quality increases but the performance decreases. As a result, we prefer to select the FOV as 80 degrees and the resolution of 512×512 pixels in a virtual viewport as a good trade-off between performance and quality. Unless specifically stated, other scenes are also rendered with the same parameters.



(a) fov50,res512,fps3.2 (b) fov80,res512,fps3.7

(c) fov110,res512,fps4.5 (d) fov110,res1024,fps3.6

(e) fov140,res512,fps5.1 (f) fov140,res1024,fps4.4

Fig. 6. Performance and quality with different FOVs and different resolutions. The FOV (fov) is the field-of-view of a virtual viewport, res is the resolution of the rendered image in a virtual viewport, and fps (frames per second) is the rendering performance.

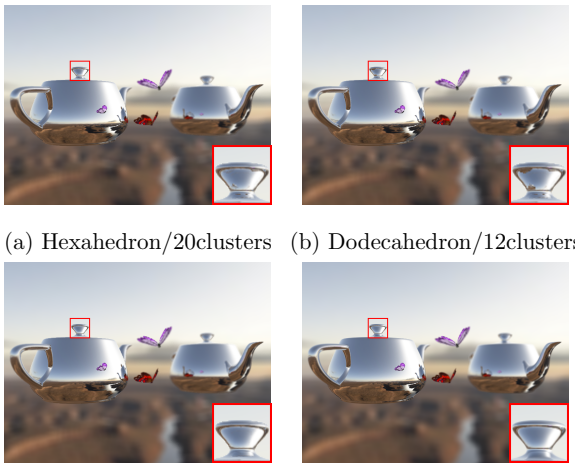
Table 1. Performance comparison with different regular polyhedron and cluster numbers. Polyhedron/clusters represent the number of the facets of regular polyhedron/the number of clusters. The first number 6 represents a unit regular hexahedron, while 12 represents a unit regular dodecahedron. fps (frames per second) is the rendering performance.

Polyhedron/clusters	6/6	6/12	6/20	12/12	12/20	12/31
Performance(fps)	2.9	3.1	2.8	3.4	3.3	2.9

The number of clusters and the generation of clusters affect both of performance and quality. The final number of clusters is a trade-off between performance and quality of tracing secondary rays. When the clusters are fewer, the cost for peeling and bounding the footprint would decrease, but the directions of secondary rays of the pixels on the boundary of a cluster deviate much from the viewing direction of the virtual viewpoint for the cluster, which will increase the cost on intersection

tests and so lower the whole performance. On the contrary, with more clusters, the cost on intersection tests would decrease, but the cost for peeling and bounding the footprint will increase. Some results for comparisons are shown in Fig. 7, where there are two reflective teapots, and the butterflies around them are reflected on both of them. It has some artifacts in the lid of the left teapot in Fig. 7 (b) from using fewer clusters to generate the DOF effects, but it can achieve a better result by using more clusters in Fig. 7 (c), which is comparable with the result of [11] in Fig. 7 (d). However, Fig. 7 (c) increases the cost of peeling and bounding the footprint and lowers the whole performance compared with Fig. 7 (b). A comparison of their performance is shown in Table 1. Besides the final number of clusters, the number of the facets of the regular polyhedron is also related to our performance and quality. Fig. 7 (a) uses a unit regular hexahedron and k-means to classify teapots into clusters, which has some artifacts in the lid of the left teapot even with 20 clusters. Our method can achieve a better result by using a unit regular dodecahedron and k-means to classify into clusters in Fig. 7 (c). Meanwhile, the regular dodecahedron has a smaller range of directions than the regular hexahedron and has better performance. A performance comparison is also shown in Table 1 between Fig. 7 (a) and Fig. 7 (c). A performance comparison of the final number of clusters and the generation of clusters is listed in Table 1. The performance of the regular dodecahedron is better than the regular hexahedron. The optimal setting is using a unit regular dodecahedron and k-means to classify into 1 to 3 clusters in each facet in our method, such as Fig. 7 (c). However, we find the setting is not very sensitive to the scenes, so we use these parameters in all scenes in this paper.

Our method can generate the DOF effects through complex detailed models. Fig. 8 shows a reflective detailed dragon. It has not only the DOF effect of the dragon, but also the DOF effect of reflected color on the dragon. Self-reflections of the dragon are also taken into account in our method. The mouth of the dragon is reflected on the body under the mouth. As shown in Fig. 8, our rendering quality in Fig. 8 (a) is comparable with the result of [11] in Fig. 8 (b).



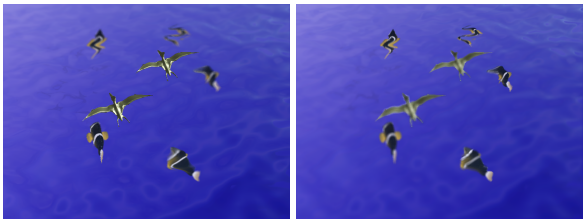
(a) Hexahedron/20clusters (b) Dodecahedron/12clusters

(c) Dodecahedron/20clusters (d) GPU-based ray tracing

Fig. 7. Performance comparison with different regular polyhedron and cluster numbers. (a) The result by using a unit regular hexahedron and k-means to classify into 20 clusters. (b) The result by only using a unit regular dodecahedron to classify into 12 clusters. (c) The result by using a unit regular dodecahedron and k-means to classify into 20 clusters. (d) The result of GPU-based ray tracing.

(a) Our method (b) GPU-based ray tracing

Fig. 8. Comparison with GPU-based ray tracing for a reflective detailed dragon.



(a) Near focal plane (b) Far focal plane

Fig. 9. The DOF effects with refractive water in different focal planes. (a) The focal plane is on the birds in the sky; (b) The focal plane is on the far fishes in the water.

Our method can also generate DOF effects through refractions. Fig. 9 shows the DOF through refractions in different focal planes. The fishes in the water are seen through the refraction on the surface of the water. The effect of the DOF shows us the depth of the fishes and the shape of the water intuitively.

Table 2. Performance of our method with 100 primary rays per pixel, 100 secondary rays of reflectors/refractors, and an image of 800 × 600 pixels. In this table title, Tri. is the number of triangles, Prim. the time of computing primary rays for direct lighting, Clust. the time of clustering, Repres. the time of compact representation of virtual points, Sec. the time of computing secondary rays for reflection, Perf. the performance of algorithm. The time is milliseconds/frame.

Scene	Tri.	Prim.	Clust.	Repres.	Sec.	Perf.(fps)
5 (a)	2.1M	N/A				15.2
5 (b)		22.2	77.8	133.3	322.2	1.8
5 (c)		N/A				0.11
6 (b)	132.8K	13.5	37.8	64.9	154.1	3.7
7 (c)	527.3K	11.8	41.2	70.6	170.6	3.4
8 (a)	124.7K	12.5	43.8	75	181.3	3.2
9 (a)	122.3K	9.8	27.5	47.1	111.7	5.1

From the statistics in Table 2, it is known that our method can efficiently support DOF rendering with secondary rays. These images are rendered interactively, though a very large number of rays are traced. The cost of primary rays/clustering/image presentation/secondary rays is about 4%/14%/24%/58% respectively. The bottleneck of our method is tracing secondary rays. It takes considerable computation time to trace primary rays multiple times due to the limitation of GPU memory. This should be further studied in the future to achieve real-time performance.

5 Conclusions and Future Work

We have presented a GPU-based interactive depth-of-field rendering algorithm with one bounce of reflections or refractions. Secondary rays are divided into multiple clusters based on the positions and directions of reflections or refractions, and so coherence among secondary rays of a cluster can be exploited for fast tracing using GPUs. Here, a layered image representation is constructed for a cluster by a virtual viewpoint and image-space ray tracing techniques [9] can be used efficiently. Experimental results show that we can produce high quality DOF effects with secondary rays, comparable with the GPU-based ray tracer [11], and perform better to render realistic results of moderate complexity scenes interactively. Our method can be used for high quality movie production and quick preview for animation.

Our method has some limitations to be solved in the future. First, the rays can be reflected or refracted only once, and we would like to extend this work to multiple bounces. Second, we hope that we can find a more efficient representation of scene geometry such as [27] to save memory and accelerate tracing. Finally, some effects from traditional ray tracing techniques such as soft shadows, color bleeding and motion blur are hard to combine with image-space ray tracing. It will be very interesting to have a hybrid framework to exploit the benefits of both ray tracers in the image space and the object space.

References

[1] Rokita P. Generating depth-of-field effects in virtual reality applications. *IEEE Computer Graphics and its Applications*, 1996, 16(2): 18–21.

[2] Scheuermann T. Advanced depth of field. *Game Developers Conference*, 2004.

[3] Hammon J E. Practical post-process depth of field. *GPU Gems 3*, 2007: 583–606.

[4] Zhou T, Chen J, Pullen M. Accurate depth of field simulation in real time. *Computer Graphics Forum*, 2007, 26(1): 15–23.

[5] Lee S, Kim G J, Choi S. Real-time depth-offield rendering using anisotropically filtered mipmap interpolation. *IEEE Transactions on Visualization and Computer Graphics*, 2009, 15(3): 453–464.

[6] Cook R L, Porter T, Carpenter L. Distributed ray tracing. *ACM SIGGRAPH Computer Graphics*, 1984, 18(3): 137–145.

[7] Haeberli P, Akeley K. The accumulation buffer: hardware support for high-quality rendering. *ACM SIGGRAPH Computer Graphics*, 1990, 24(4): 309–318.

[8] Lee S, Eisemann E, Seidel H P. Depth-of-field rendering with multiview synthesis. *ACM Transactions on Graphics*, 2009, 28(5): 134:1–134:6.

[9] Lee S, Eisemann E, Seidel H P. Real-time lens blur effects and focus control. *ACM Transactions on Graphics*, 2010, 29(4): 65:1–65:7.

[10] Hou Q., Qin H, Li W., Guo B, Zhou K. Micropolygon ray tracing with defocus and motion blur. *ACM Transactions on Graphics*, 2010, 29(4): 64:1–64:10.

[11] Hou Q M, Sun X, Zhou K, Lauterbach C, Manocha D. Memory-scalable gpu spatial hierarchy construction. *IEEE Transactions on Visualization and Computer Graphics*, 2011, 17(4): 466–474.

[12] Potmesil M, Chakravarty I. A lens and aperture camera model for synthetic image generation. *ACM SIGGRAPH Computer Graphics*, 1981, 15(3): 297–305.

[13] Fatahalian K, Luong E, Boulos S, Akeley K, Mark W R, Hanrahan P. Data-parallel rasterization of micropolygons with defocus and motion blur. In *Proc. of the Conference on High Performance Graphics*, New Orleans, USA, Aug., 2009, pp.59–68.

[14] Yu X, Wang R, Yu J. Real-time depth of field rendering via dynamic light field generation and filtering. *Computer Graphics Forum*, 2010, 29(7): 2099–2107.

[15] Barsky B, Bargteil A, Garcia D, Klein S. Introducing vision-realistic rendering. In *Proc. of Eurographics Rendering Workshop*, Pisa, Italy, Jun., 2002, pp.26–28.

[16] Kraus M, Strengert M. Depth-of-field rendering by pyramidal image processing. *Computer Graphics Forum*, 2007, 26(3): 645–654.

[17] Kass M, Lefohn A, Owens J. Interactive depth of field using simulated diffusion on a GPU. *Technical report*, Pixar Animation Studios, 2006.

[18] Kosloff T, Barsky B. An algorithm for rendering generalized depth of field effects based on simulated heat diffusion. In *Proc. of the 2007 international conference on Computational science and its applications*, Kuala Lumpur, Malaysia, Aug., 2007, pp.1124–1140.

[19] Lee S, Kim, G J, Choi S. Real-time depth-of-field rendering using splatting on per-pixel layers. *Computer Graphics Forum*, 2008, 27(7): 1955–1962.

[20] Kosloff T J, Tao M W, Barsky B A. Depth of field postprocessing for layered scenes using constant-time rectangle spreading. In *Proc. of Graphics Interface*, Kelowna, Canada, May, 2009, pp.39–46.

[21] Barsky B, Chu M T D, Horn D. Elimination of artifacts due to occlusion and discretization problems in image space blurring techniques. *Graphical Models*, 2005, 67(6): 584–599.

[22] Chen J, Wang B, Wang Y, Overbeck R, Yong J H, Wang W. Efficient depth-of-field rendering with adaptive sampling and multiscale reconstruction. *Computer Graphics Forum*, 2011, 30(6): 1667–1680.

[23] Lehtinen J, Aila T, Chen J, Laine S, Durand F. Temporal light field reconstruction for rendering distribution effects. *ACM Transactions on Graphics*, 2011, 30(4): 55:1–55:12.

[24] Laine S, Aila T, Karras T, Lehtinen J. Clipless dual-space bounds for faster stochastic rasterization. *ACM Transactions on Graphics*, 2011, 30(4): 106:1–106:6.

[25] Ragan-Kelley J, Lehtinen J, Chen J, Doggett M, Durand F. Decoupled sampling for graphics pipelines. *ACM Transactions on Graphics*, 2011, 30(3): 17:1–17:17.

[26] Roger D, Assarsson U, Holzschuch N. Whitted ray-tracing for dynamic scenes using a ray-space hierarchy on the gpu. In *Proc. of the Eurographics Symposium on Rendering*, Grenoble, France, Jun., 2007, pp.99–110.

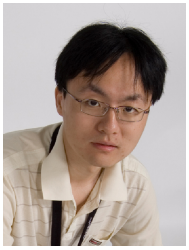
[27] Rosen P, Popescu V, Hayward K, Wyman C. Non-pinhole approximations for interactive rendering. *IEEE Computer Graphics and its Applications*, 2011, 31(6): 68–83.

[28] Popescu V, Sacks E, Mei C. Sample-based cameras for feed forward reflection rendering. *IEEE Transactions on Visualization and Computer Graphics*, 2006, 12(6): 1590–1600.

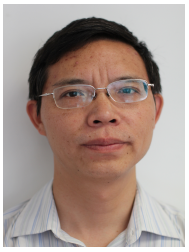
[29] Decoret X. N-buffers for efficient depth map query. *Computer Graphics Forum*, 2005, 24(3): 393–400.

[30] Liu F, Huang M C, Liu X H, Wu E H. Efficient depth peeling via bucket sort. In *Proc. of the Conference on High Performance Graphics*, New Orleans, USA, Aug., 2009, pp.51–57.

Guo-Fu Xie received BS degree in software engineering from Xiamen University in 2007. He is now a PhD student on State Key Laboratory of Computer Science, Institute of Software, Chinese Academy of Sciences. His major research interests lie in real-time photorealistic rendering and vector rendering.



Xin Sun graduated from Zhejiang University, Hangzhou, China, where he received the Bachelor's degree and Ph.D degree in computer science in 2002 and 2008 respectively. He is now working in the Internet Graphics Group of Microsoft Research Asia as a researcher. His research interests lie in real-time global illumination rendering and GPU-based photorealistic rendering.



Wen-Cheng Wang received his PhD degree from the Institute of Software, Chinese Academy of Sciences in 1998, where he is currently a professor of the State Key Laboratory of Computer Science. His research interests include computer graphics, visualization, virtual reality and expressive rendering and editing.