공학박사 학위논문

Depth of Field Rendering Using Progressive Lens Sampling in Direct Volume Rendering

직접 볼륨 렌더링에서 점진적 렌즈 샘플링을 사용한 피사계 심도 렌더링

2021 년 2 월

서울대학교 대학원

전기 · 컴퓨터공학부

강 지 선

Depth of Field Rendering Using Progressive Lens Sampling in Direct Volume Rendering

지도교수 신 영 길

이 논문을 공학박사 학위논문으로 제출함
2020 년 10 월

서울대학교 대학원
전기 · 컴퓨터공학부
강 지 선

강지선의 공학박사 학위논문을 인준함
2020 년 12 월

위 원 장 김 명 수 (인)

부위원장 신 영 길 (인)

위 원 서 진 욱 (인)

위 원 김 보 형 (인)

위 원 이 정 진 (인)

Abstract

Depth of Field Rendering Using Progressive Lens Sampling in Direct Volume Rendering

Jiseon Kang

Department of Electrical Engineering and Computer Science

The Graduate School

Seoul National University

Direct volume rendering is a widely used technique for extracting information from 3D scalar fields acquired by measurement or numerical simulation. To visualize the structure inside the volume, the voxel's scalar value is often represented by a translucent color. This translucency of direct volume rendering makes it difficult to perceive the depth between the nested structures. Various volume rendering techniques to improve depth perception are mainly based on illustrative rendering techniques, and physically based rendering techniques such as depth of field effects are difficult to apply due to long computation time. With the development of immersive systems such as virtual and augmented reality and the growing interest in perceptually motivated medical visualization, it is necessary to implement depth of field in direct volume rendering.

This study proposes a novel method for applying depth of field effects to volume ray casting to improve the depth perception. By performing ray casting using multiple rays per pixel, objects at a distance in focus are sharply rendered and objects

at an out-of-focus distance are blurred. To achieve these effects, a thin lens camera model is used to simulate rays passing through different parts of the lens. And an effective lens sampling method is used to generate an aliasing-free image with a minimum number of lens samples that directly affect performance. The proposed method is implemented without preprocessing based on the GPU-based volume ray casting pipeline. Therefore, all acceleration techniques of volume ray casting can be applied without restrictions.

We also propose multi-pass rendering using progressive lens sampling as an acceleration technique. More lens samples are progressively used for ray generation over multiple render passes. Each pixel has a different final render pass depending on the predicted maximum blurring size based on the circle of confusion. This technique makes it possible to apply a different number of lens samples for each pixel, depending on the degree of blurring of the depth of field effects over distance. This acceleration method reduces unnecessary lens sampling and increases the cache hit rate of the GPU, allowing us to generate the depth of field effects at interactive frame rates in direct volume rendering.

In the experiments using various data, the proposed method generated realistic depth of field effects in real time. These results demonstrate that our method produces depth of field effects with similar quality to the offline image synthesis method and is up to 12 times faster than the existing depth of field method in direct volume rendering.

Keywords : Direct volume rendering, ray casting, volume visualization, depth of field effect, computer graphics, image generation

Student Number : 2013-20736

Contents

List of Figures

Notation and Symbols

Circle of Confusion (CoC)

Central Processing Unit (CPU)

Computed Tomography (CT)

Depth of Field (DoF)

Direct Volume Rendering (DVR)

Distributed Ray Tracing (DRT)

Early Ray Termination (ERT)

Elliptical Weighted Average (EWA)

Frames Per Second (fps)

Graphics Processing Unit (GPU)

Magnetic Resonance Imaging (MRI)

Open Graphics Library (OpenGL)

OpenGL Shading Language (GLSL)

Three-Dimensional (3D)

Two-Dimensional (2D)

Volume Bounding Box (VBB)

CHAPTER 1

INTRODUCTION

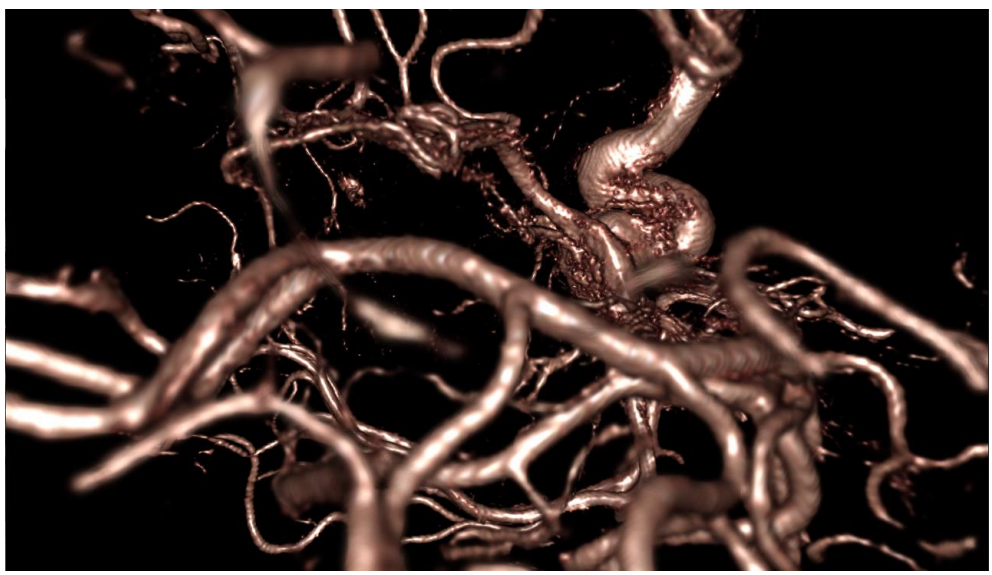
1.1 Motivation

Volume rendering is a technique that provides insight to users by projecting volumetric data in 2D images. Volume rendering techniques have been developed for decades with the advancement of graphics processing units (GPUs) and can be grouped into two categories: indirect volume rendering and direct volume rendering. *Indirect volume rendering* techniques use intermediate representations (e.g., polygon meshes) to approximate surfaces contained in volumetric data, which then can be rendered in the same way as surface rendering using conventional graphics accelerator hardware. The marching cubes algorithm [1] is commonly used to extract iso-surfaces from volumetric data. This method of using geometric primitives of GPU is useful in many applications. However, since only surface representations are used, much of the information contained inside the volumetric data can be lost in the result image. And an excessive number of geometric primitives may be required for complex surface approximation [2]. Unlike indirect volume rendering, *direct volume rendering* (DVR) creates a 2D image directly from associated voxels of the volumetric data. There are two algorithms that formed the main stream of the DVR.

These are slice-based (or texture-based) volume rendering, which slices 3D texture blocks that store volumetric data, and volume ray-casting, which traces rays from the camera to the volume [2][3]. Volume ray-casting is more widely used than slice-based approaches because rays are processed independently of each other, allowing for several optimization strategies [4]. These DVR methods can capture both the interior and the surfaces of objects by mapping the scalar values of volumetric data to semi-transparent colors [5][6].

Direct volume rendering has been applied to various applications that require interactive exploration of volumetric data in medical and industrial fields. Especially in medical field, it helps doctors to navigate anatomical structure of a patient quickly and accurately with minimal effort. Medical imaging data, usually obtained by physical measurement devices such as computed tomography (CT) and magnetic resonance imaging (MRI) scanners, inherently has noise and inhomogeneity. Therefore, the transfer function of DVR that maps voxels to specific colors may not be sufficient to classify and visualize the structure of volumetric data. In addition, the organic shapes and spatial relationships between anatomical structures can be often quite complex to recognize, making medical visualization difficult [7]. In particular, visualizing inter-object relations is challenging when nested structures are rendered with translucent colors. Thus, illustrative rendering techniques of methods that mimic physical phenomena have been studied to enhance the perception of DVR [8]. Depth Darkening [9], Volume Halos [10], Volume Illustration [11], Volumetric Line Drawings [12] and Depth of Field rendering [13] are perception-based

(a)



(b)

Figure 1.1 Aneurysm images rendered by DVR. The depth information between the aneurysms, which is difficult to distinguish in (a) the conventional DVR image, is well revealed in (b) the DVR image with depth of field effects (our method [14]).

techniques widely used in DVR [15]. Depth of Field enhances depth perception in a more realistic way than other techniques based on illustrative rendering techniques. Figure 1.1 shows the effect of applying DoF when rendering medical images with DVR. The depth relationship between aneurysms, which is difficult to distinguish in the conventional DVR image (Figure 1.1(a)), is well revealed in the DoF DVR image (Figure 1.1(b)).

Depth of field (DoF) is the distance range of an object that the eye or camera perceives as in focus. As the object is far away from the distance range, it gradually blurs. This is one of the ways how humans perceive depth, and it has been used in computer graphics to enhance realism or draw attention to certain areas. The DoF effects was first applied to ray tracing. It creates the DoF effects by sampling an area of the camera lens [16], and is considered as the ground truth in surface rendering. This method is suitable for creating high-quality offline images such as movies since it requires many rays to obtain an DoF image without aliasing. Therefore, many studies have been published to quickly approximate the DoF effects in the field of surface rendering where high frame rates must to be guaranteed, such as games. Most researches focus on postprocessing a pinhole image using blur filters based on depth map [17][18]. However, the postprocessing methods quickly increase the computation time as the lens size increases, and produce unnatural DoF effects in areas of the scene that are not visible in the pinhole image. Nevertheless, this approach has been actively studied as it is very fast and accurate effect simulations are not critical. However, in volume rendering fields, the DoF effects have been

rarely researched. One of the well-known studies is by Schott *et al.* [13], which applies the DoF effects to slice-based volume rendering. In Schott's paper, the DoF effects can be easily integrated into the conventional volume rendering algorithm by applying blur filters between slices. However, this method produces an overly blurry image in some cases because the blurred slices are repeatedly composited. In addition, acceleration is difficult due to the structure of the basic rendering algorithm. All voxels in the volume need to be rendered even if they don't affect the final image. Except for this study, little has been explored about DoF rendering techniques in DVR, especially with respect to volume ray casting. Considering that volume ray casting has more acceleration techniques than slice-based volume rendering [4], the volume ray casting with the DoF effects is more useful for visualizing medical volumetric data in real time.

1.2 Dissertation Goals

In this dissertation, we propose a novel DoF volume rendering method that provides realistic volume visualization in real time. The purpose of this study is to enhance depth perception in DVR using DoF effect. Our method is easily integrated into GPU-based volume ray casting by adding a few steps to the conventional ray casting pipeline, which means that existing acceleration techniques can be used without restrictions. To simulate the DoF effects, the camera model is changed from the pinhole to the thin lens model. This approach increases the number of rays that need

to be computed per pixel, but produces an optically accurate DoF image. Our DoF ray casting method use *early ray termination* (ERT), one of the ray casting acceleration techniques, to reduce calculations per ray. In addition, a progressive lens sampling technique that reduces the number of rays per pixel was introduced to address the slowdown caused by the increased cost of ray computation.

To evaluate that it synthesizes the appropriate DoF effects, our method is compared with Cook *et al*. [16], which is considered as the ground truth. Cook's method is a ray tracing technique for surface rendering, but can be applied to volume rendering by borrowing the concept of sampling a lens and generating multiple rays from one pixel. In addition, Our method is compared to the existing DoF volume rendering method [13] based on 3D texture slicing, regarding the image quality and rendering performance. Our goal is to overcome the limitations of the existing DoF volume rendering technique and propose an appropriate acceleration technique that enables interactive rendering. We expect that the DoF images generated by our method to be used in medical applications such as preoperative planning or immersive systems such as virtual or augmented reality systems [19].

1.3 Main Contributions

The main contributions of this dissertation are:

- **A novel object-space approach to generate realistic DoF effects in DVR.**

  Until now, only the image-space approach using a blurring filter kernel has

been studied in DVR, so it is the first DVR method for simulating DoF effects based on physical phenomena. Our method is an object-space approach that samples the lens and casts rays, thus generating realistic DoF effects. In this dissertation, we explain how to convert a camera model from the pinhole camera model to the thin lens model in DVR. And, we describe how to construct rays from samples on the thin lens to compute the volume-rendering integrals.

- **A fast DoF rendering method that extends GPU-based volume ray casting.** The DoF rendering method in this paper is based on the GPU-based ray casting. Our method does not require any pre-processing and there are no restrictions on applying ray casting acceleration techniques. Thus, it can be easily integrated into existing ray casting applications. We extend the GPU-based volume ray casting pipeline and apply the existing acceleration technique to our algorithm as well to quickly obtain DoF images.
- **A new acceleration technique based on multi-pass rendering using progressive lens sampling.** We further propose the acceleration technique that uses a different number of lens samples to determine the result color of each pixel. For this, we designed multi-pass rendering and assigned progressive lens samples to each render pass. Each pixel determines the final render pass using the circle of confusion computed based on the thin lens model. This acceleration technique suppresses lens oversampling, allowing time-consuming DoF effects to be produced quickly and without quality degradation.

1.4 Organization of Dissertation

The remainder of this dissertation is organized as follows. Chapter 2 presents of DoF related works in surface rendering and volume rendering. Chapter 3 provides a brief overview of camera models and volume rendering algorithms for DoF rendering. Then, the DoF volume ray casting method using a finite lens model is described. This chapter also describes an acceleration technique that uses multi-pass rendering with appropriate lens sampling strategies. In Chapter 4, the overall workflow of the proposed method and implementation details are described. Chapter 5 present the experimental results of the proposed method for various volumetric data compared with the existing method. Finally, we summarize the results and discuss future work in Chapter 6.

CHAPTER 2

RELATED WORK

Depth of field is often used in photography and cinematography. Images created with proper defocusing are generally perceived more realistic and aesthetic [20]. Blur affects the perception of the size and depth order of objects. Of course, blur alone does not provide accurate depth information of objects. However, it is known that the combination of blur and other depth cues can greatly increase the viewer's ability to perceive absolute or relative distances between objects [21][22].

In computer graphics, DoF is used to enhance realism and improves depth recognition. And it is also used to guide viewers to salient regions by blurring less relevant regions [13]. The method of simulating the DoF effects has been actively studied for decades since the introduction of the lens and aperture camera models [17]. A typical solution for simulating the DoF effects is Monte-Carlo ray tracing using a finite lens model [16]. DoF ray tracing is accurate, but it takes a long time to calculate. Therefore, post-filtering techniques that synthesize DoF images in real time for interactive applications such as games have been studied the most.

In volume rendering, on the other hand, little research has been done on DoF compared to other techniques that simulate physical phenomena to improve visualization. This is because volume rendering is mainly used for scientific visualization, so research is focused on expressing features more accurately than

creating realistic images. In addition, it is difficult to ensure real-time rendering if the DoF effects is simulated with the same fashion of surface rendering without additional acceleration. Therefore, a new DoF volume rendering technique suitable for volume rendering is required. This chapter describes some of the studies that have made up the mainstream of DoF surface rendering. And we discuss attempts to apply the DoF effects to volume rendering.

2.1 Depth of Field on Surface Rendering

As the DoF effects can be easily found in games and movies, many DoF rendering techniques have been developed in surface rendering. Accordingly, several surveys on DoF rendering algorithms have been compiled. Demers [23] published a short survey of DoF rendering algorithms, most of which are based on the GPU depth buffer (z-buffer). It focuses on the real-time rendering method of the current graphics hardware and covers how to apply a blur filter with the pixel depth value of the pinhole image via forward or reverse mapping. Barsky and Kosloff [24] classified DoF rendering methods into two categories: the object-space approach and the image-space approach. Generally, the object-space approaches [25] produce a more realistic image and have fewer artifacts than the image-space method. However, the image-space approaches [26] are more widely used as they approximate the DoF effects much faster. According to Barsky's category, representative DoF rendering techniques in surface rendering can be listed as follows.

2.1.1 Object-Space Approaches

The object-space approaches work on 3D objects using lens sampling. To generate the DoF effects, a camera model with a lens is used instead of a pinhole camera model. This approach simulates rays starting from an object and passing through different parts of the lens and reaching the image plane. Since not all rays passing through the lens can be considered, sampling techniques are required to uniformly sample the lens. This method produces optically correct results because it simulates how the DoF image is actually formed. However, a large number of lens samples are required to obtain a high-quality image, and the calculation time increases in proportion to the number of samples. Several methods have been proposed for acceleration, but it is still difficult to reach in real time. So, it is mainly used in the field of offline image synthesis such as cinema, or as the ground truth for evaluating DoF rendering algorithms.

Cook *et al*. [16] proposed the *distributed ray tracing* (DRT) that directly simulates geometric optics. Rather than tracing a single ray per pixel as in a pinhole camera, it generates multiple rays per pixel to sample the finite aperture. DRT is illustrated in Figure 2.1 for a single ray. This method constructs a ray from the center of the lens to the film plane (image plane). A point on the lens is selected and a ray from the point to the focal point of the original ray is traced. If the number of lens samples is insufficient, aliasing appears in the result image. The larger the lens size and blur, the more the lens samples are required. As shown in Figure 2.1, each ray

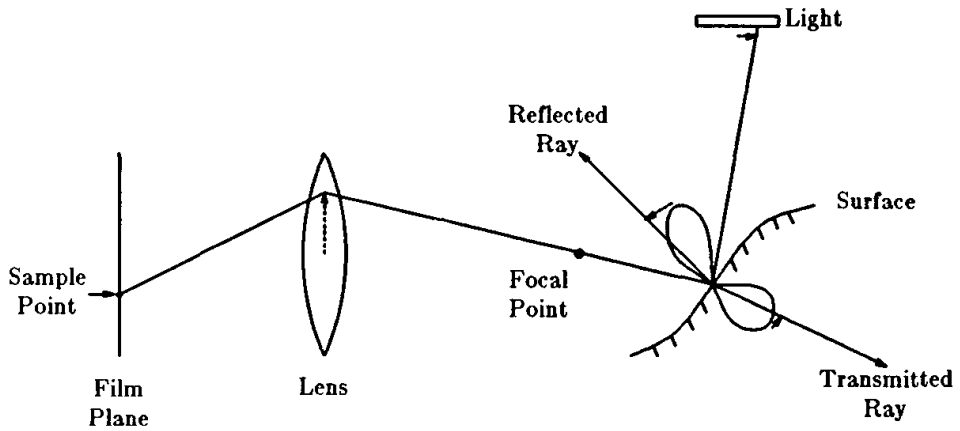


Figure 2.1 Typical ray path of the distributed ray tracing. Construct a ray from the center of the lens to a point on the film plane. Choose a location on the lens, and trace a ray from the location to the focal point of the original ray. Image courtesy of Cook [16].

Figure 2.2 The ecosystem scene rendered by the distributed ray tracing (DRT) with 128 samples per pixel. Image courtesy of Pharr *et al*. [27].

generated from the image plane generates additional rays that reflect or pass through the surface of the object. Therefore, DRT performs more than twice the number of lens samples and is suitable for generating off-line images. Pharr *et al*. [27] gives a fine reference for implement details and examples (see Figure 2.2). In this paper, a visualization program implementing DRT is used to verify the quality of DoF volume rendering methods. For a detailed description of the visualization program, refer to the related document [28].

Haeberli and Akeley [29] proposed a multi-pass rendering method using an accumulation buffer for rapid generation of DoF images. A collection of pinhole images is rendered with the projection matrix modified to view the scene at various discrete points across the lens aperture. The projection matrix is calculated by moving the camera position while the specific plane ($z = z_f$) in the view frustum is fixed as shown in Figure 2.3 [30]. And the accumulation buffer is used to average the pinhole images. Figure 2.4 is an example image created by this method. In principle, this method is very similar to DRT in that it takes many samples per pixel and averages them. Unlike DRT, which can be optimized by applying adaptive control over the number and distribution of sample per pixel, image accumulation technique has no choice but to use the same lens samples for all pixels. In addition, it is difficult to guarantee interactive frame rates because many passes are needed to get good image quality. However, since the image accumulation method is more accurate than the image-space approach, several studies on additional acceleration methods are still published [31][32].

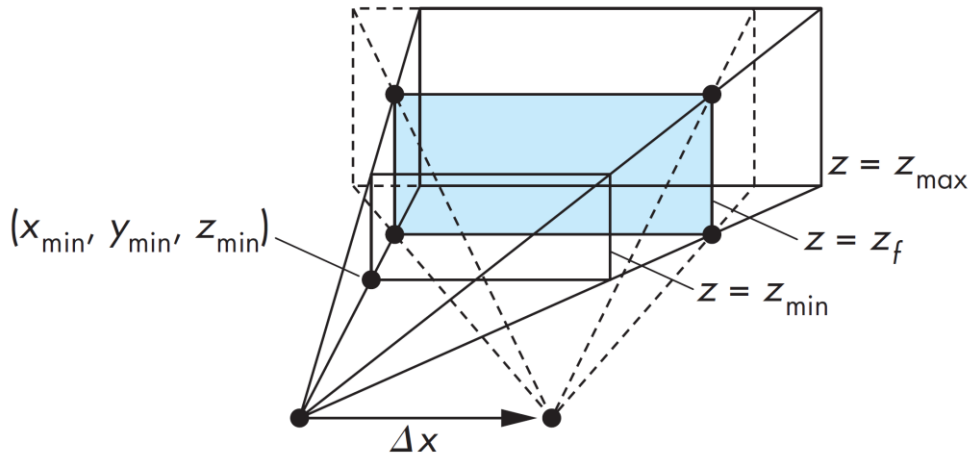


Figure 2.3 Illustration of depth of field jitter. The trick is to move the viewer in a manner that leaves a particular plane ($z = z_f$) fixed. Image courtesy of Angle and Shreinner [30].

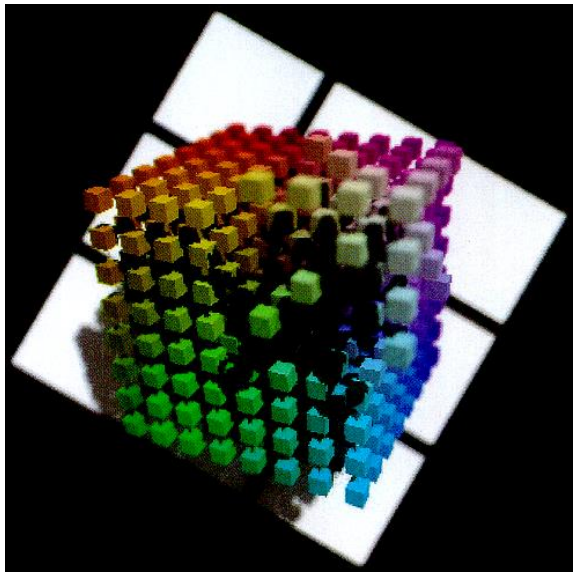


Figure 2.4 Depth of field image generated by accumulating 23 pinhole images. Image courtesy of Haeberli and Akeley [29].

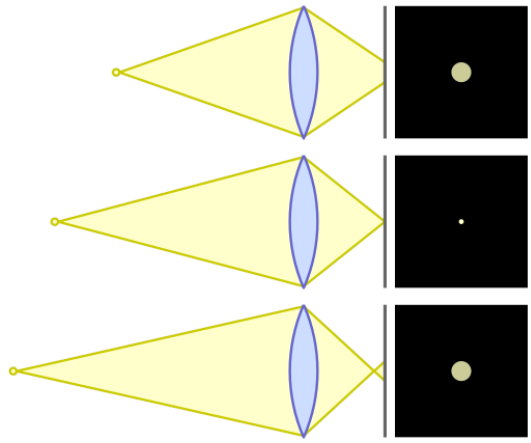


Figure 2.5 Circle of confusion (CoC) when the point source is too close, in focus, and too far. Image courtesy of Wikipedia.

2.1.2 Image-Space Approaches

The image-space approach simulates the DoF effects on the image plane using post-filtering. In this approach, pinhole images generated using the conventional surface rendering method are filtered with variable-sized filter kernels to simulate *circle of confusion* (CoC). The CoC refers to the phenomenon in which a point light source at an out-of-focus distance appears as a blurred spot on the image plane. And the degree of blurring is determined by the lens parameters (e.g., focal length and aperture size) and the distance from the lens to the light source (see Figure 2.5). The image-space approach is faster than the object-space approach so that it is most commonly used for real-time surface rendering such as games. However, there are inevitable problems on the image-space approach. First, the calculation time

increases rapidly as the lens size increases. The size of the lens determines the size of the blur filter, which in the worst case may take several minutes to process the large size blur filter. Second, *Partial occlusion* is an inherent problem of the image-space approach. The pinhole image captures only the region visible from the center of the lens. However, in DoF images, some invisible parts of the pinhole image may appear beyond the soft edges of blurred foreground objects. Handling these issues well is the most important challenge in the image-space approach.

Most image-space methods are based on the early work of Potmesil and Chakravary [17][18], which is the *scattering* approach. The scattering method spreads the pixel intensity of the pinhole image into a circular sprite. All sprites are sorted by depth and blended in the image plane. This method requires heavy sorting of entire depths and is too slow to implement in real time. Křivánek *et al*. [33] presented a point-based DoF rendering similar to the Potmeil's method. The proposed algorithm is an extension of the *elliptical weighted average* (EWA) surface splatting [34][35], which renders each point as an elliptical Gaussian. Individual points are blurred based on its depth before forming EWA surface splatting. Figure 2.6 shows examples of DoF surface splatting when there are transparent surfaces. This algorithm is accelerated with a level-of-detail technique that selects a coarse level when the blur is large (see Figure 2.7). Lee *et al*. [36] was inspired by Krivánek's method. Lee's method uses an additional pinhole image of the partially occluded area (see Figure 2.8). And the partial occlusion information was used to reduce the resolution of the point array to accelerate the algorithm almost in real time.

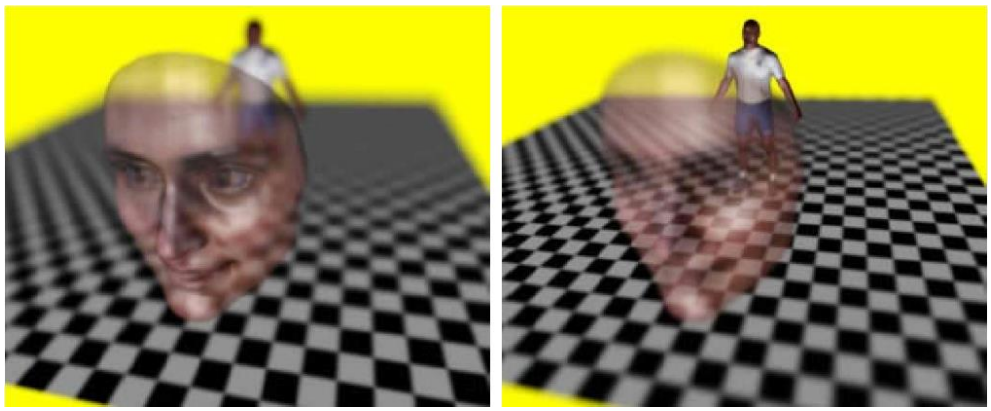


Figure 2.6 Point-based DoF rendering with semitransparent surface. On the left, the transparent mask is focused, and on the right the male body is focused. Image courtesy of Křivánek *et al*. [33].

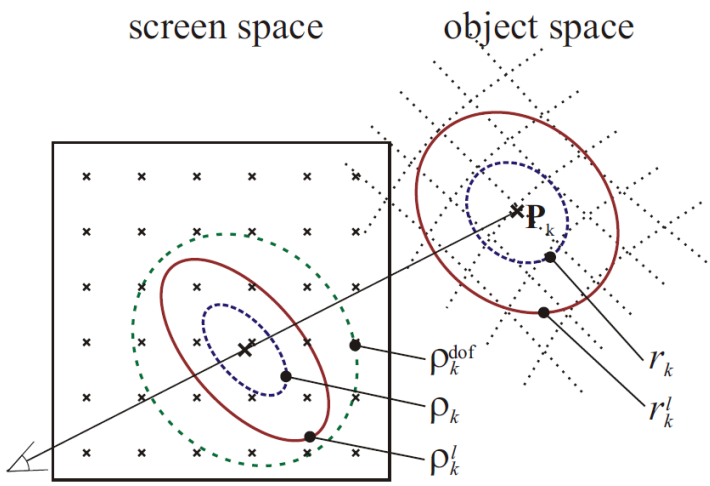


Figure 2.7 DoF surface splatting with different LODs. Image courtesy of Křivánek *et al*. [33].

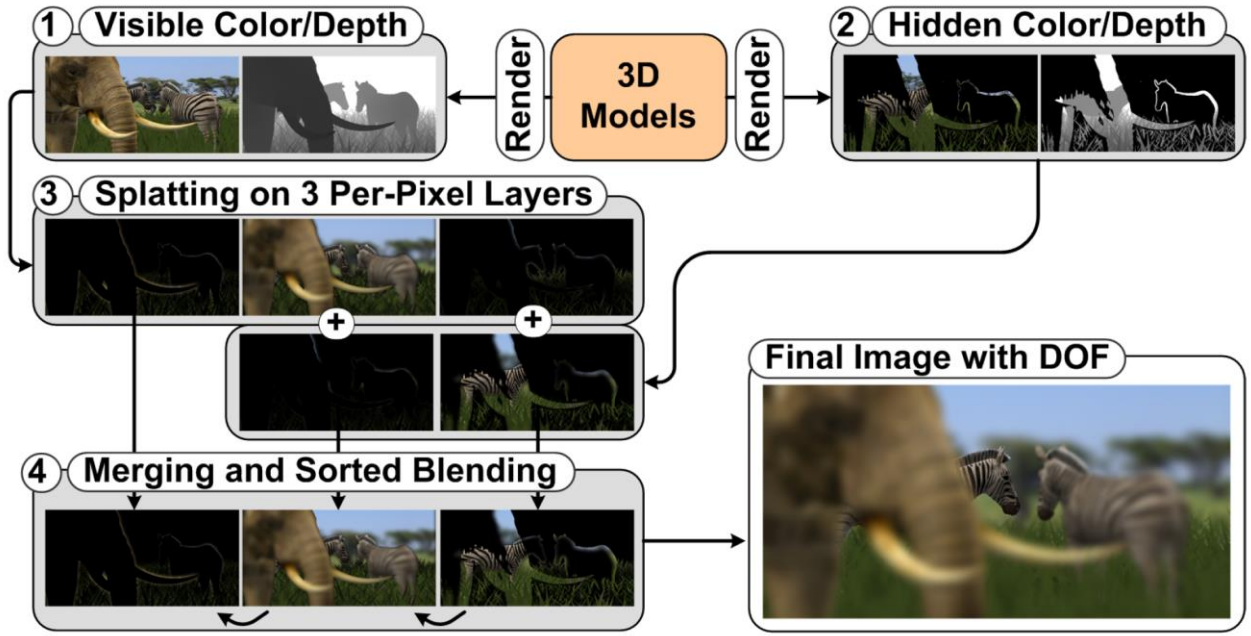


Figure 2.8 Overview of DoF rendering using point splatting on per-pixel layers. First, a typical pinhole image is rendered with depth. Second, another color/depth image is rendered for partially occluded areas. Third, the two pinhole images are separately blurred using point splatting. In the last step, the blurred layers are alpha-blended and normalized in the descending order of layer depths. Image courtesy of Lee *et al*. [36].

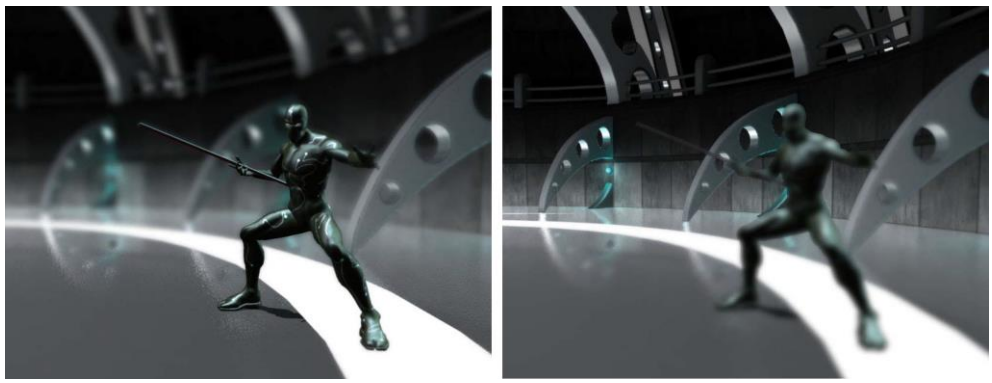
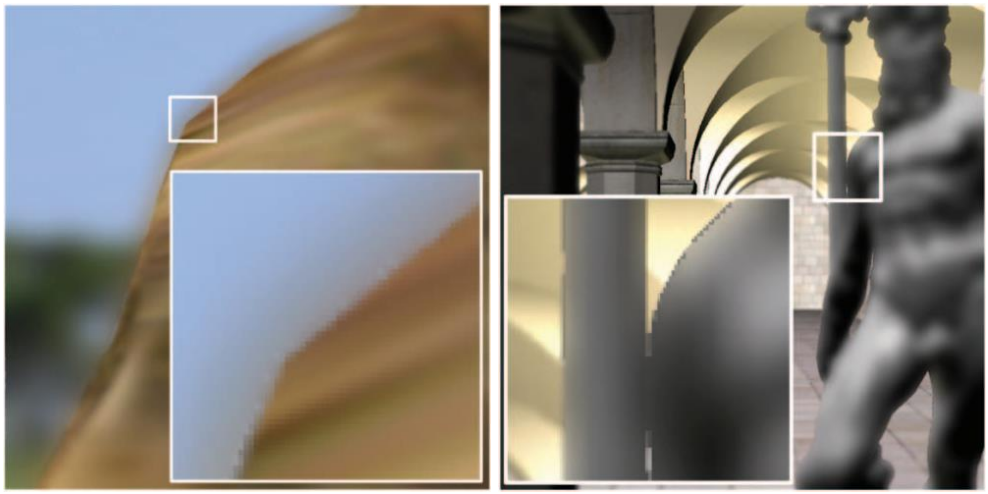


Figure 2.9 Depth of field effects using down-sampling and pre-blurring techniques. A $1/16$ sized image is created from the pinhole image and blurred with 3×3 Gaussian filter. Then, the original pinhole image and the pre-blurred image is blended. Image courtesy of Scheuermann [43].

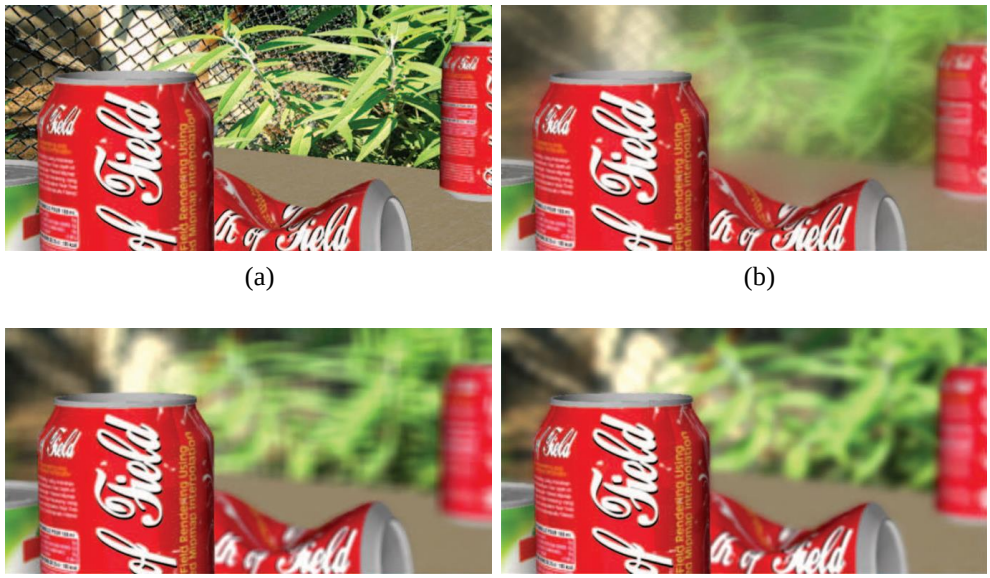
The *gathering* approach collects blur information by filtering adjacent pixels within the CoC region of the current pixel. This approach is easy to implement, regardless of the scene complexity. And, the algorithm is well mapped to the GPU, allowing real-time rendering. Early work using blur-filtering was perfoemed by Rokita [37][38], which simply and inexpensively generated blur circles using the iterative filtering with a 3×3 convolution mask. Based on Rokita's model, Mulder [39] presented a faster DoF rendering for virtual reality systems using two techniques: an accurate DoF filtering for high-resolution focused regions and a fast DoF approximation based on Gaussian pyramids for low-resolution unfocused regions [40][41]. Riguer *et al*. [42], Scheuermann [43] and Hammon [44] presented the most commonly used DoF implementations on GPUs (see Figure 2.9). They generate



(a) (b)

Figure 2.10 Artifacts of image-space DoF rendering based on a linear-filtering. (a) Intensity leakage. (b) Blurring discontinuity. Image courtesy of Lee *et al.* [47].

down-sampled images pre-blurred using linear filters for the DoF effects in real time. However, these methods have quality issues because the surfaces hidden from the center of the lens is processed with surrounding pixels. Figure 2.10 shows two artifacts in image-space DoF rendering when applying a linear filtering technique to a pinhole image. Figure 2.10(a) is an example of *intensity leakage*, which occurs where pixel intensities in a focused area flow into a blurred area. And, Figure 2.10(b) is an example of *blurring discontinuity* in which a blurred object has a sharp silhouette in front of the focal plane where the depth map changes abruptly [24]. Real-time DoF rendering studies published later focused on addressing these issues well. Zhou *et al*. [45] presented an advanced two-pass filter technique similar to Riguer's method. In Zhou's method, vertical and horizontal Gaussian filters are



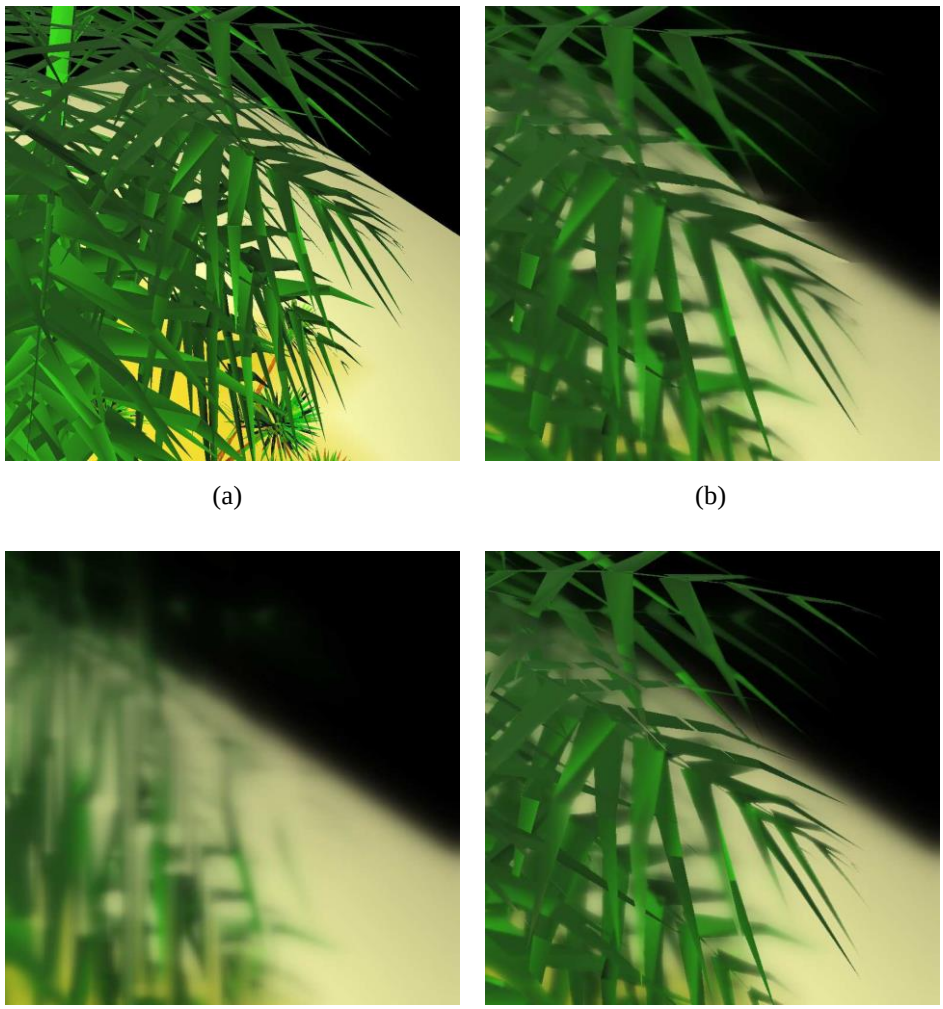
(a) (b)

(c) (d)

Figure 2.11 A comparison of background-blurred images for (a) the pinhole image, rendered by (b) Scheuermann's method [43], (c) Zhou *et al.*'s method [45], and (d) Lee *et al.*'s method [47]. Image courtesy of Lee *et al.* [47].

applied sequentially, and each pixel's weight is dynamically adjusted according to the CoC to eliminate intensity leakage. However, more subtle artifacts can be observed due to the lack of correct depth order. Some methods have attempted to better solve these problems with depth information using anisotropic filtering. Bertalmío *et al*. [46] proposed a method based on heat diffusion to reduce artifacts at the original resolution. And, Lee *et al*. [47] proposed an anisotropic mipmapping scheme that nonlinearly interpolates the DoF effects from discrete-level mipmap images. Figure 2.11 is rendered with the mentaioned gathering methods, and the Lee's anisotropic filtering showed the most natural DoF effects.

Since the single view approach using a pinhole image cannot properly solve the partial occlusion problem, DoF rendering methods using multiple layers have been studied. The *layering* approach was first presented by Scofield [48]. It is a simple and fast approach, but requires additional memory. In Scofield's algorithm, objects are sorted in groups according to their depth and rendered into separate images called layers. Each layer is individually blurred with frequency domain convolution. Then, the blurred layers are composited into the final image with alpha blending. Barsky [49] presented a non-interactive system developed from Schfield's algorithm. In this method, the pinhole image is decomposed into sub-images according to the depth of the pixels, individually blurred and then blended. Kass *et al.* [50] applied multi-layering in DoF rendering based on heat diffusion. This method separates the background layer to prevent blurring of the foreground objects (see Figure 2.12), and is implemented on the GPU to get the interactive frame rate. Kraus and Strengert [51] implemented Barsky's approach [49] on the GPU to achieve real-time DoF rendering (see Figure 2.13). Kraus' method is fast, but has the downside of creating artifacts for large-sized CoC. Recently, Lee *et al*. [52] present a faster alternative solution employing the multi-view accumulation method [29] (see Figure 2.14). Unlike previous image-space studies based on color extrapolation and alpha blending, the proposed method shoots rays into the layers and accumulate their contributions. Lee's method recovers the invisible part of the scene more accurately than the methods using extrapolation. However, a large number of rays are required to achieve smooth results, which directly affects the computation time.



(a) (b)

(c) (d)

Figure 2.12 Multi-layered DoF rendering based on heat diffusion. (a) Original pinhole image. (b) Single-layer diffusion results in artifacts at the horizon adjacent to the leaves. Kass compute a separate background layer (c) and blend it with the single-layer diffusion for the final result (d). Image courtesy of Kass *et al*. [50].

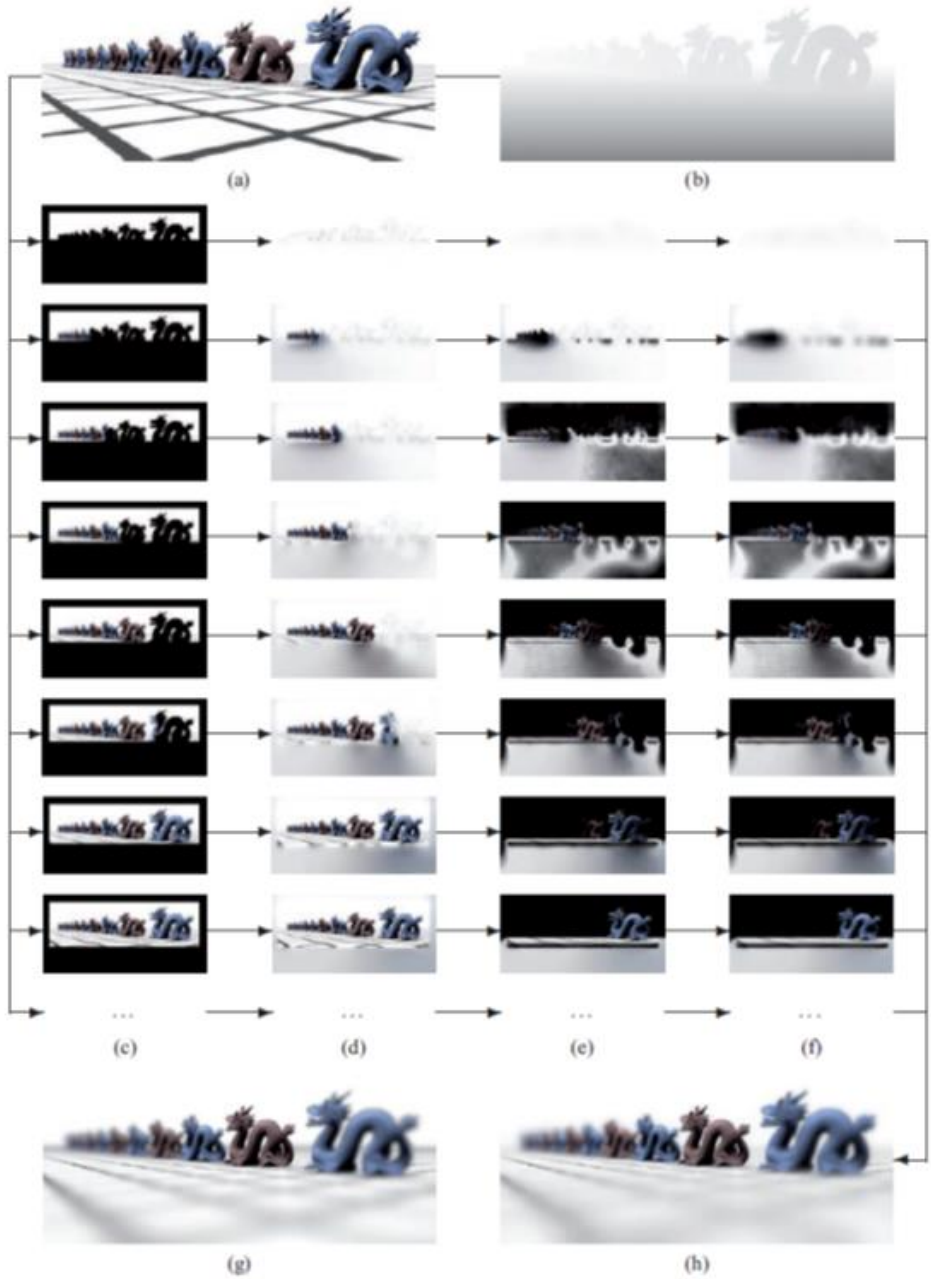


Figure 2.13 Data flow of the pyramidal DoF rendering. (a) input pinhole image, (b) input depth map, (c) sub-images after culling of foreground pixels, (d) sub-images after disocclusion of culled pixels, (e) sub-images after matting, (f) sub-images after blurring, (g) ray-traced reference image, (h) blended result. Image courtesy of Kraus and Strengert [51].

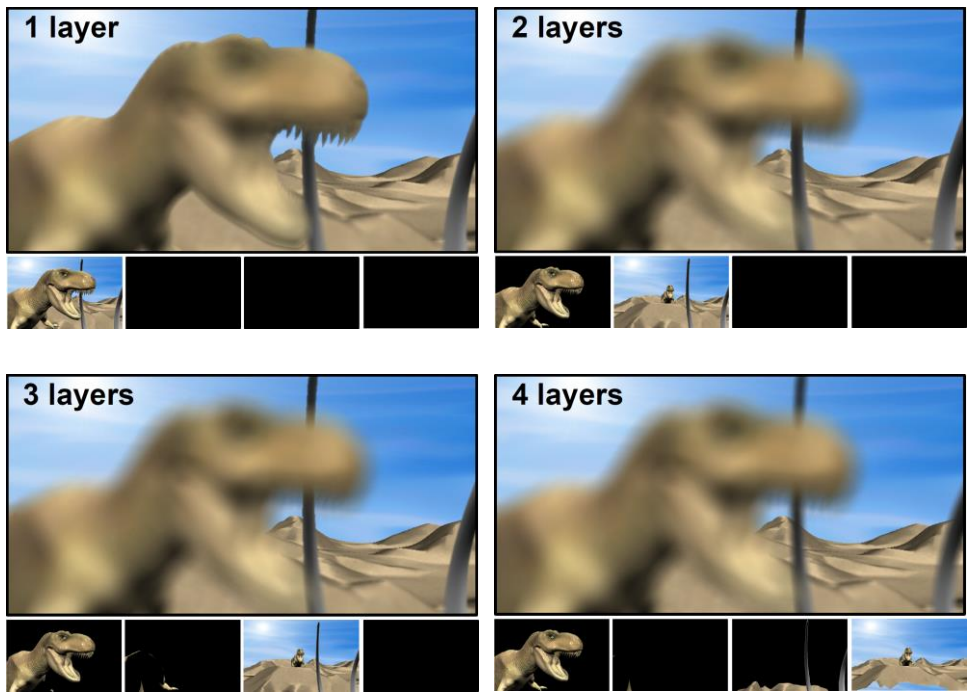
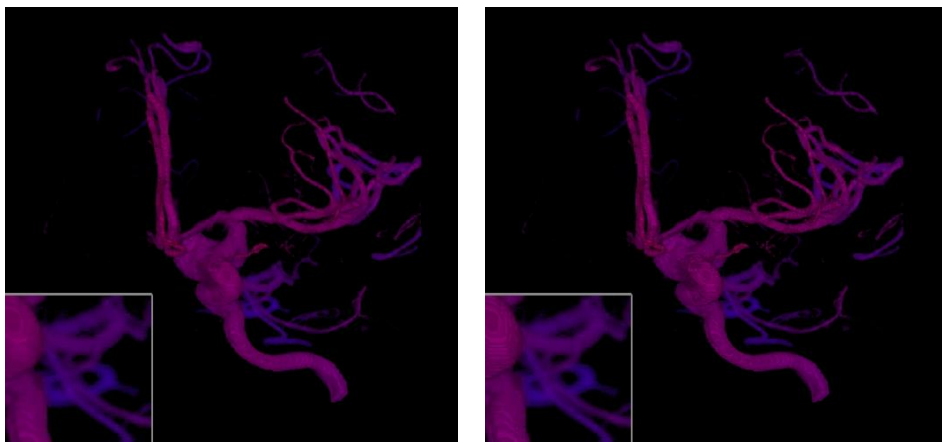


Figure 2.14 Depth of field rendering with multiview synthesis. Images shows the effect of the number of layers on the finale result. Image courtesy of Lee *et al*. [52].

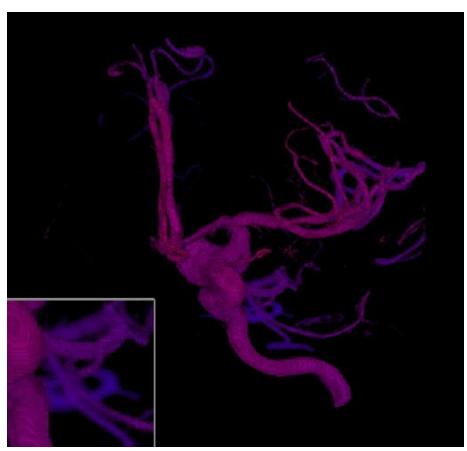
2.2 Depth of Field on Volume Rendering

Many of the effects used in surface rendering have also been used in volume rendering. However, there are few DoF studies in volume rendering compared to the amount of DoF surface rendering studies presented previously. The first reason is that volume rendering is basically more computationally expensive than surface rendering. The object-space approach of generating multiple rays per pixel makes real-time image acquisition impossible when applied to volume rendering. Second, the image-space approach, which is a fast approximation method, is difficult to apply as it is. In surface rendering, each pixel value of a pinhole image is determined by a single point in the 3D scene. But, in volume rendering, each pixel value is determined by sampling a ray passing through 3D space. This means that the pinhole image in volume rendering does not have a single depth map. For these reasons, there has been little interest in DoF for volume rendering, and only a few outstanding studies.

Ropinski [53] tried to achieve a DoF-like effect in angiography visualization. In Ropinski's paper, the authors assume that the viewer is focusing on an object closer to the camera. So, they blur the part where the distance to the camera exceeds a certain threshold. The distance threshold is given as percentage of the depth interval of the volumetric data. Figure 2.15 shows examples of DoF-like effects applied with different distance thresholds, where (a), (b) and (c) are 10%, 30%, and 60%, respectively. As shown in the enlarged image in Figure 2.15, the relative distance can be inferred from the change in blurring of blood vessels according to



(a) (b)



(c)

Figure 2.15 Application of modified depth of field effect with varying distance thresholds: 10% (a), 30% (b), and 60% (c). Image courtesy of Ropinski *et al.* [53].

the distance threshold. This method only borrows the concept of the DoF effects to improve depth perception, it does not simulate the actual DoF effects. Therefore, it is impossible to estimate the absolute distance from the viewer. However, this is meaningful as an example showing that the DoF effects increases depth perception in volume rendering.

DoF rendering technique for volumetric data is influenced by the underlying volume rendering algorithm. Until recently, there are only two known ways to adequately simulate the DoF effects in direct volume rendering. One is a method of performing blur filtering on slice compositing in slice-based volume rendering, and the other is a method of referring to surrounding voxels using a filter for each ray sample point in volume ray casting. The following describes the two methods in detail.

2.2.1 Blur Filtering on Slice-Based Volume Rendering

Schott *et al*. [13] is the most the representative way to implement the DoF effects for volume visualization. Schott's method is based on 3D texture slicing and incorporates blur filtering into the slice compositing process, like the image-space approach. As shown in Figure 2.16, volume slice sets are divided into front and rear subsets of the focal plane and composited separately. As each subset is synthesized sequentially from the slice furthest from the focal plane, incremental filtering is performed that samples pixels and accumulate previous synthesis results. So, the

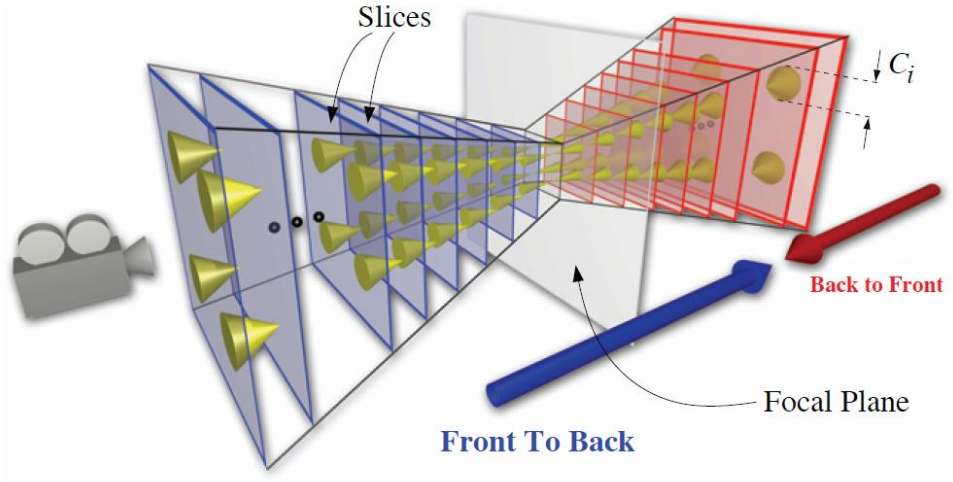
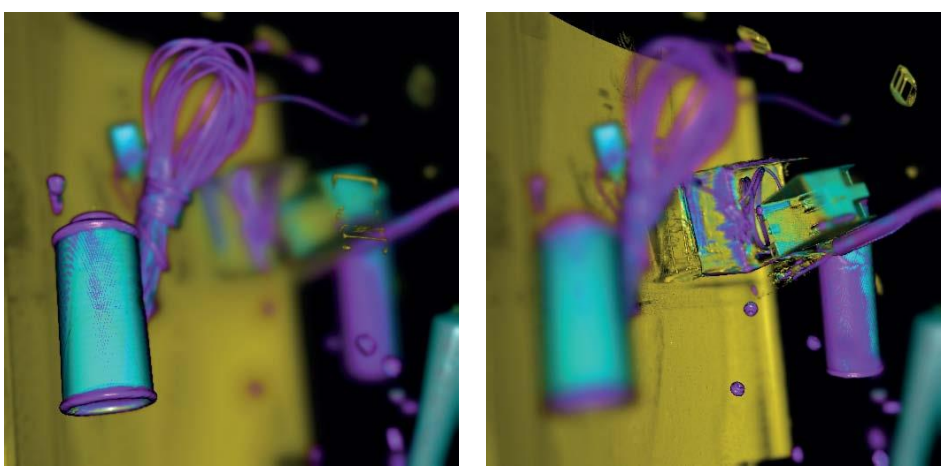


Figure 2.16 Geometric setup of DoF slice-based direct volume rendering system in two separate passes; those in front of the focal plane in front-to-back order and those behind the focal plane in back-to-front order. Image courtesy of Schott *et al.* [13].



(a) (b)

Figure 2.17 The backpack data set with $512 \times 512 \times 373$ voxels rendered by DoF slice-based volume rendering (1469 slices). In each image, the focus is (a) on the spray can, (b) and the boxes, respectively. Image courtesy of Schott *et al.* [13].

farther the slice is from the focal plane, the larger the blur is due to kernel convolution. This is similar to the layering technique of the image-space approach in surface rendering, which uses a kernel of CoC size based on the depth. The advantage of this method is that it requires only minor modifications to the conventional slice-based volume rendering algorithm. However, this method requires rendering all voxels in the slice because the previous blur information is transmitted when the slices are composited. Therefore, it is difficult to increase the rendering speed by applying general acceleration techniques that skip voxels that do not affect the final image according to the transfer function. In Figure 2.17, the 512×512 images were generated at 3.8 fps (frames per second), which is half the speed when rendered without the DoF effects. Moreover, as the compositing stage progresses, blur is repeatedly applied to already-synthesized slices, which could result in exaggerated blurring. This is an inherent disadvantage of the algorithm, and the user cannot finely control the degree of blurring. Excluding these two drawbacks, it is the best DoF volume rendering technique in existence, and is the most widely used.

2.2.2 Stochastic Sampling on Volume Ray Casting

Recently, a study on volume ray casting has been published that approximates the DoF effects with a fairly small number of samples. Sharifi and Boulanger [54] drew attention to the study of Křivánek *et al*. [33] that a high sampling rate of low-

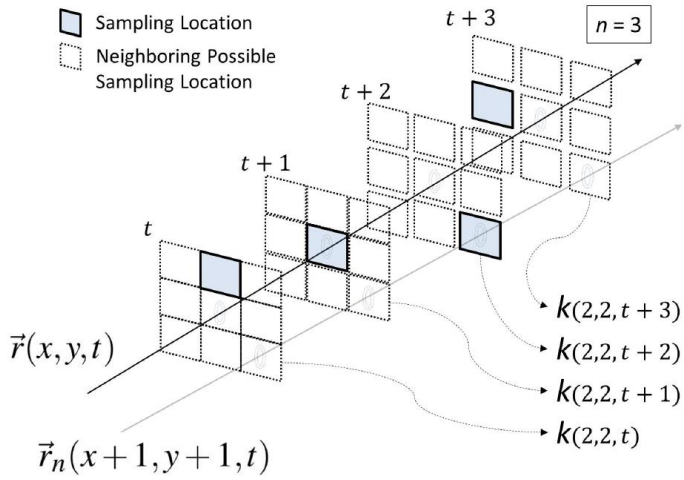
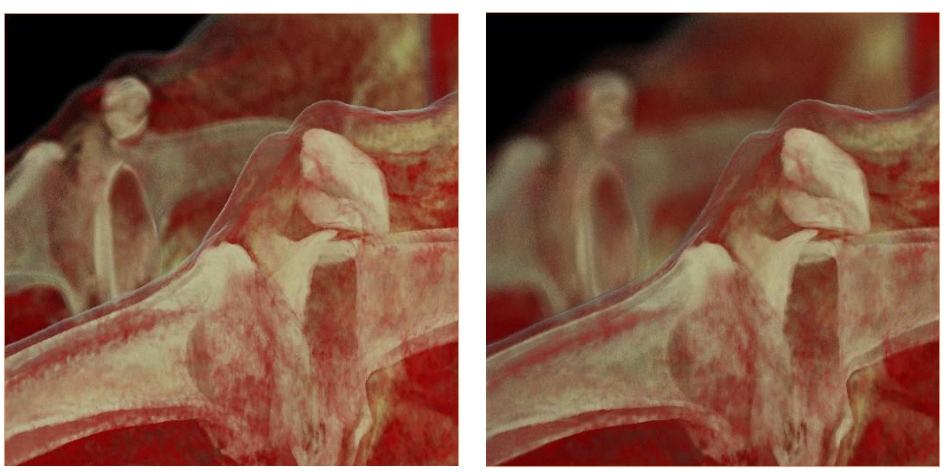


Figure 2.18 Permutation-based stochastic sampling method for ray casting to render images with DoF effects. A different sampling location is chosen within a kernel on each time-step. Image courtesy of Sharifi and Boulanger [54].



(a) (b)

Figure 2.19 The knees data set with $512 \times 512 \times 912$ voxels rendered by DoF stochastic sampling. (a) shows the moderated DoF effects and (b) shows the shallower DoF effects. Image courtesy of Sharifi and Boulanger [54].

resolution part of the input signal is not required as long as the Nyquist rate is preserved. They used this idea to reduce the number of samples required for blurring. As shown in Figure 2.18, the algorithm places a 2D kernel parallel to the viewing plane at each sampling location, and average intensity of all samples taken from the kernel plane. Each kernel size is determined according to the CoC, so a blur similar to the DoF effect is simulated. Aliasing due to insufficient sampling is reduced by sparsely selecting samples using permutations in the kernel. However, using sparse kernels may result in the omission of some details of the blurry area (see Figure 2.19). Therefore, it is hard to say that it provides better quality, although superior to Schott's method [13] in terms of controlling the intensity of the blurring.

In summary, DoF rendering technique, which provides natural depth information to the user, has little progress in the field of volume rendering. Since volume rendering applications deal with medical or industrial images, the accuracy of the DoF simulation is very important. At the same time, the applications need to interact with the user, so real-time rendering must also be guaranteed. For these reasons, DoF rendering based on volume ray casting can be a good solution. Volume ray casting can simulate the DoF effects using lens sampling and has various acceleration methods. To the best of our knowledge, a DoF study suitable for volume ray casting has not yet been proposed. This dissertation deals with how to apply the object-space approach of DoF rendering to volume ray casting, which could not be applied due to rendering speed issues. It also details GPU-based acceleration techniques and implementations for real-time DoF rendering.

CHAPTER 3

DEPTH OF FIELD VOLUME RAY CASTING

In this chapter, we propose an interactive DoF volume rendering technique that produce realistic DoF effects in volume ray casting. The idea is to change the pinhole camera model of the conventional ray casting to a camera model that uses a finite-sized lens to generate multiple rays per pixel. To do this, we modify the volume ray casting algorithm to use the thin lens model. We also describe an appropriate sampling strategy to uniformly sample the lens to reduce aliasing. The proposed method works in the GPU-based ray casting pipeline without any pre-processing, and can use the acceleration techniques of conventional volume ray casting. However, conventional acceleration techniques are insufficient to handle the amount of computation that increases in proportion to the number of lens samples. Therefore, a new acceleration technique based on the thin lens model is proposed to compensate for the frame rate drop due to the increased ray computation.

3.1 Fundamentals

This session covers the basic concepts required before explaining the DoF volume ray casting proposed in the next session. First, the definition of DoF and the

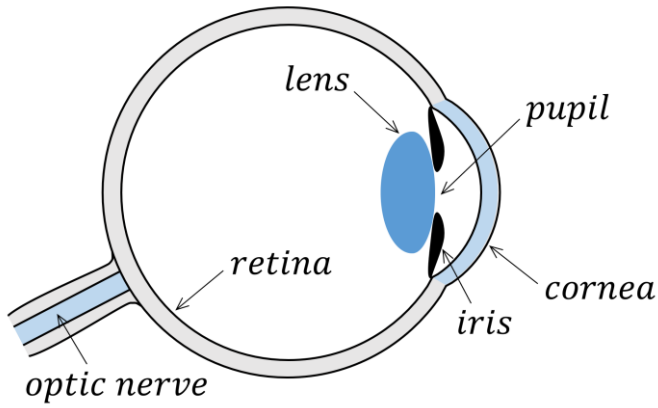


Figure 3.1 The human eye. Important features include the fovea, where vision is sharpest; the pupil, around aperture through which light enters the eye; and the two principal optical elements, the lens and the cornea [55]. Image courtesy of Angel and Shreiner [30].

camera model for the DoF effects in computer graphics are described. The settings of camera model are important not only to control the DoF effects, but are also to accelerate the DoF rendering algorithm. Then, the conventional DVR algorithms and acceleration techniques are mentioned. The characteristics of each algorithm and their suitability as an underlying DoF rendering framework are discussed.

3.1.1 Depth of Field

In our eyes, the shape of the lens is distorted to focus light on the retina (see Figure 3.1). This process is called accommodation. The human visual system obtains

information from the amount of distortion of the lens and use it as a depth cue especially for objects at close distance [56][57]. Real cameras also have lenses with a finite aperture and a lens control that adjusts the lens position with respect to the film plane. Due to the aperture is finite, lenses have a specific distance at which objects are “in focus”. That is, both the human eye and the real camera can focus on only one plane, called the focal plane [27]. Points that are not on the focal plane appear as blur spots in the image. The blur spot gradually increases in size and decreases in intensity as it moves away from the focal plane. This phenomenon is mentioned in Session 2.1.2, and the blur spot, called CoC, is illustrated in Figure 2.5. The human eye and real camera recognize spots less than a certain size as points according to the optic nerve and film resolution, respectively. Depth of field is a zone in which these blur spots appear very small, indistinguishable from points. And the blur effect observed in out-of-focus areas is called the DoF effect.

Figure 3.2 shows an example of the DoF effects. The image was rendered by ray tracing using 25 samples per pixel and a large disk lens. The focus is on the glass and bottle on the table. Other objects that are out-of-focus behind the focus appear blurred. Note that the picture frame on the wall is blurrier than the chair because it is located further from the focal plane. The glass and bottle naturally catch the user’s attention. Also, due to the difference in blur, it feels like the walls are located considerably farther than the table and chair. In this way, the DoF effects can give an image a sense of realism and provide information about the relative distance of objects.

Figure 3.2 An example of depth of field. Image courtesy of Marschner and Shirley [58].

3.1.2 Camera Models

In computer graphics, a camera model is an essential tool for capturing 3D scenes into images. In this session, we describe the *pinhole camera model* commonly used in computer graphics and the *thin lens model* for generating the DoF effects. It helps to understand the differences in the implementation of each of the two camera models and convert the conventional DVR algorithm using the pinhole camera into a DoF rendering algorithm using the thin lens camera. In addition, the calculation the DoF and CoC size depending on parameters of the thin lens model is covered.

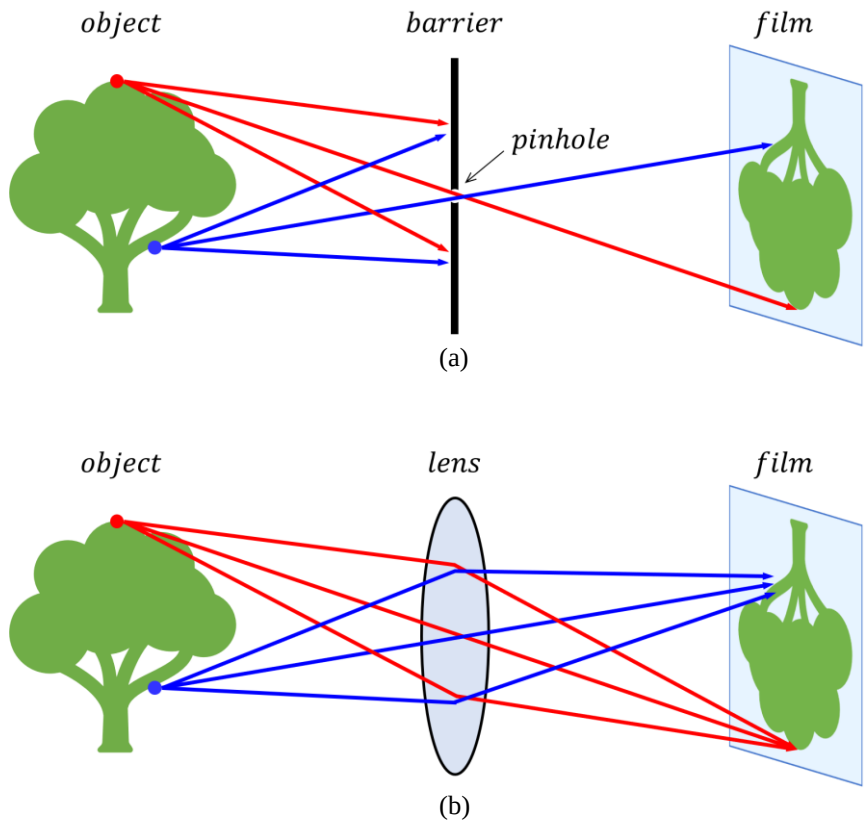


Figure 3.3 Camera models. (a) Pinhole camera model. (b) Thin lens model. Image courtesy of Hata and Savarese [59].

Pinhole Camera Model

Most computer graphics are based on the pinhole camera model with sharp focus on all objects. Figure 3.3(a) shows how the pinhole camera works. There is a barrier with a small aperture between an object and a film plane. The aperture is referred to as the pinhole or center of the camera and is considered a single point. Thus, even if each point of the object emits multiply rays, only a single ray can reach the film plane. The film plane is commonly called the image plane in computer graphics. Each pixel of the image plane and the point of an object are mapped to one-to-one [29]. In computer graphics, the camera is placed at a pinhole and the image plane is placed in the direction the camera is facing so that the result image dose not flip.

Thin Lens Model

Human eyes and real cameras collect light that passes through a finite-size lens. Figure 3.3(b) illustrates a simple lens model. The rays emitted from the top point on the tree (red arrows) are refracted by the lens and converged to a single point on the film plane. On the other hand, the rays from the middle point of tree (blue arrows) are not converged perfectly on the film plane. The way to simulate these rays in computer graphics is to replace the pinhole with a lens in the camera model and trace the rays passing through different locations on the lens [29]. Real lenses have various shapes and combination depending on the purpose of use. In computer graphics, the

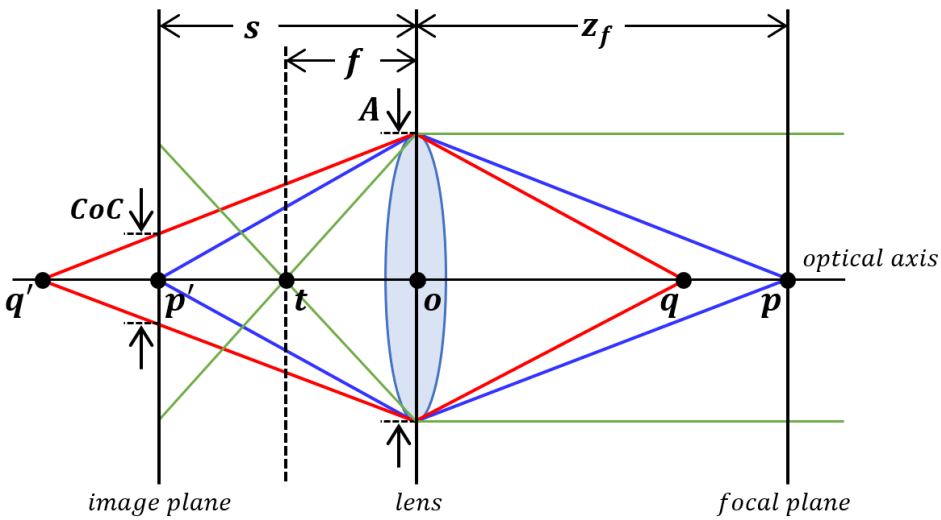


Figure 3.4 Illustration of the thin lens model. Point p is on the focal plane at distance z_f and blue rays from point p gather at point p' on the image plane. Red rays starting at point q out of the focal plane form a blur circle on the image plane (circle of confusion, COC) at distance s [13]. Note that all rays parallel to the optical axis intersect at the focal point t. The distance from the center of the lens o to the focal point t is called the focal length f.

thin lens model, a hypothetical lens model, is used to generate the DoF effects.

Figure 3.4 shows the thin lens model illustrated in 2D. This lens model assumes that the lens is infinitely thin, so refraction inside the lens is ignored. The line that passes through the center of the lens and is perpendicular to the principal plane of the lens is calls the optical axis. Note that all rays parallel to the optical axis intersect at a single point on the optical axis, the focal point. The distance from the center of the lens to the focal point is called the focal length. In Figure 3.4, rays starting at the point p on the focal plane pass through the lens and form the point p' on the image plane. To focus an object on the image plane, the object must be at a certain distance from the lens, called the focal distance. This relationship can be formulated as the thin lens equation (3.1), where f is the focal length, s is the distance from the lens to the image plane, and z_f is the distance from the lens to the focused object. Note that the focal length f and the focal distance z_f are different.

$$\frac{1}{f} = \frac{1}{s} + \frac{1}{z_f} \tag{3.1}$$

In Figure 3.4, rays starting at the point q, not on the focal plane, converge at the point q' at a location other than the image plane [60]. And in the image plane, it appears as a blur circle called as CoC. The CoC size is affected by the radius of the aperture and the object's distance. The diameter of CoC is calculated using Equation (3.2), where z is the object's distance from the lens and A is the diameter of the lens aperture [13].

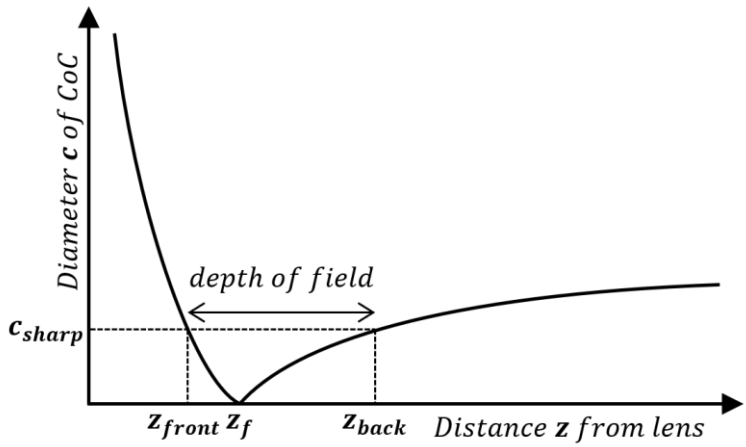


Figure 3.5 The diameter of the circle of confusion (CoC) in image space as a function of the distance of an object from the lens. Image courtesy of Schott *et al*. [13].

$$c(z) = \left| A \frac{f(z_f - z)}{z(z_f - f)} \right| \tag{3.2}$$

As shown in Figure 3.5, changes in the CoC size do not have the same slope in front and behind the focal plane. Objects between the lens and the focal plane have a much bigger CoC compared to those behind the focal plane. As the object's distance z goes to infinity, the CoC size converges to constant value [13]. In Figure 3.5, $[z_{front},\ z_{back}]$ is the DoF where objects appear in focus because the CoC size is smaller than the pixel size c_{sharp}. In the thin lens model, the DoF is computed as follows:

$$z_{front} = \frac{Afz_f}{Af + c_{sharp}(z_f - f)}, \tag{3.3}$$

$$z_{back} = \frac{Afz_f}{Af - c_{sharp}(z_f - f)}, \tag{3.4}$$

where

$$c_{sharp} = \frac{2}{h} \times s \times \tan\left(\frac{FOV}{2}\right), \tag{3.5}$$

h is the height of viewport, s is the distance from lens to the image plane, and FOV is the camera's field of view.

3.1.3 Direct Volume Rendering

Direct volume rendering visualizes the volumetric data by evaluating the emission-absorption optical model (see Figure 3.6). The voxel's scalar value is virtually mapped to a physical quantity that describes the interaction of light. This mapping is called classification and is usually performed with a transfer function. In DVR, the light propagation along viewing rays is computed by integrating light interaction effects based on the optical model. The light energy of the viewing ray is described by its radiance I, which is computed by the *volume-rendering integral*:

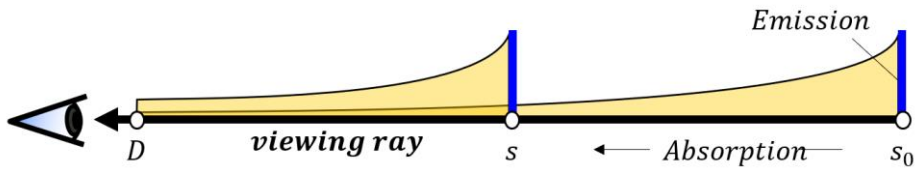


Figure 3.6 Illustration of the emission-absorption optical model. The light is absorbed along the viewing ray until it reaches the eye. The volume-rendering integral is a formula for this optical model. Image courtesy of Engel *et al*. [4].

$$I(D) = I_0 e^{-\int_{s_0}^{D} \kappa(t)dt} + \int_{s_0}^{D} q(s) e^{-\int_{s_0}^{D} \kappa(t)dt} ds, \qquad (3.6)$$

with optical properties κ (absorption) and q (emission) and integration from entry point into the volume, $s = s_0$, to the exit point toward the camera, $s = D$ [4]. And, the volume-rendering integral is solved numerically by a discrete approximation via back-to-front or front-to-back compositing [61].

Direct volume rendering algorithms are grouped in object-order or image-order approach. The object-order approach uses a forward scheme in which the 3D volume is projected onto the image plane. The image-order approach, on the other hand, uses a backward scheme in which viewing rays are cast from each pixel in the image plane into the 3D volume. The most popular algorithms for each approach are texture slicing and ray casting respectively, which are described in detail below.

Slice-Based Volume Rendering

Slice-based volume rendering or *texture slicing* is an object-order approach. There are several texture slicing methods, and view-aligned slicing in combination with 3D textures is the most common algorithm. In the slice-based volume rendering, volumetric data is stored in a 3D texture in GPU memory. Then, the 3D texture is sliced into geometric primitives, which are proxy slices parallel to the image plane (see Figure 3.7). In this way, the volumetric data can be represented as a stack of slices with slices spacing that meet the Nyquist-Shannon sampling theorem [62]. During rasterization, 3D texture coordinates are interpolated over the interior of the proxy slices, and used to sample the 3D texture values. Finally, all slices are colored by a transfer function and projected onto the image plane. Either front-to-back or back-to-front order can be used to composite slices. Typically, the back-to-front compositing using GPU's alpha blending is selected.

One drawback of 3D texture slicing is that the sampling interval is not consistent for all viewing rays in the perspective projection. As shown in Figure 3.7, the distance of sampling points varies with the angle between the slice and the viewing ray. This problem can be solved by using spherical shells [63] instead of planar slices, but is usually ignored because it is noticeable in large fields of view. Another disadvantage is that the entire data needs to be rendered. This method is inefficient, especially if the voxels near the camera are mapped to opaque colors by the transfer function. A huge amount of work such as shading and blending

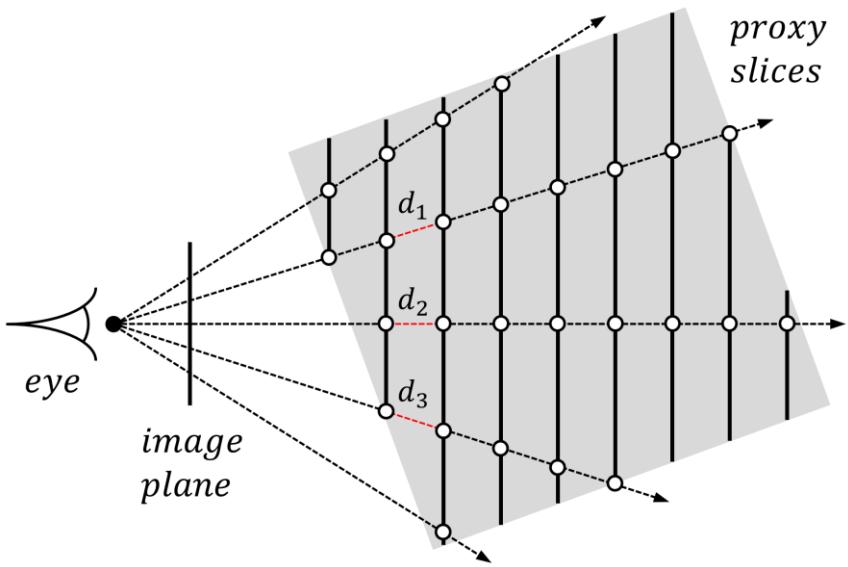


Figure 3.7 The slice-based volume rendering using view-aligned slices as proxy geometry. The distance between adjacent sampling points indicates by d_1, d_2 and d_3 are slightly different. Image courtesy of Engel *et al*. [4].

on the back-side slices doesn't contribute to the final image. Since the existing DoF rendering method in DVR [13] is based on slice-based volume rendering, it has all the aforementioned problems.

Volume Ray Casting

Volume ray casting is an image-order approach and is the dominant method in GPU-based volume rendering [3]. The basic idea is to emit a ray from each pixel towards

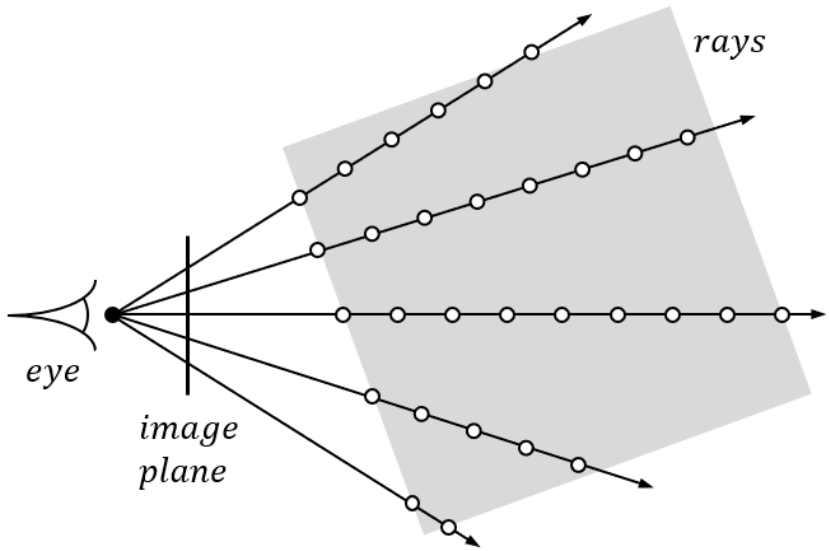


Figure 3.8 Volume ray casting principle. For each pixel, one viewing ray is traced and sampled at discrete positions to evaluate the volume-rendering integral. Image courtesy of Engel *et al*. [4]

the volume and compute the volume-rendering integral along the ray. Figure 3.8 illustrates volume ray casting. For each pixel in the image plane, a single ray is cast into the volume. Then the volume is resampled at discrete positions along the ray. The scalar values at each sampling position are mapped to the optical properties defined by the transfer function. Front-to-back compositing is applied in the same order as the ray traversal.

The outstanding feature of volume ray casting is that each viewing ray is independent, which makes it easy to overcome the problems of slice-based volume

rendering. Since ray sampling is performed at regular intervals along the viewing ray, there is no problem which is inconstant sampling distance as in slice-based volume rendering. In addition, volume ray casting is easy to accelerate than slice-based volume rendering. For example, ERT is a simple acceleration technique that terminates the ray casting process when the ray's opacity is full, and considers only the voxels that affect the final image. It helps to take into account only the voxels that affect the final image. Due to these advantages, volume ray casting is much more widely used than slice-based volume rendering.

Volume ray casting is more suitable for DoF rendering than slice-based volume rendering because it properly produces perspective views. Direct volume rendering typically uses the orthogonal projection by placing the camera outside the volume. However, endoscopic views require the perspective projection with large fields of view from inside the volume, and accurate computation of the volume-rendering integral is important [64][65]. Since the existing DoF rendering method for DVR proposed by Schott [13] is based on slice-based volume rendering, incorrect DoF images may be generated when rendering endoscopic images with a wide field of view. In addition, volume ray casting can apply the realistic DoF effects with the object-space approach using lens sampling. Of course, there is a problem that the performance is degraded by the number of rays increased by lens sampling. There is a limit to reducing the increased computational load just by using ERT. Therefore, a new acceleration technique that utilizes ray independence and performs DoF rendering in real time must be accompanied.

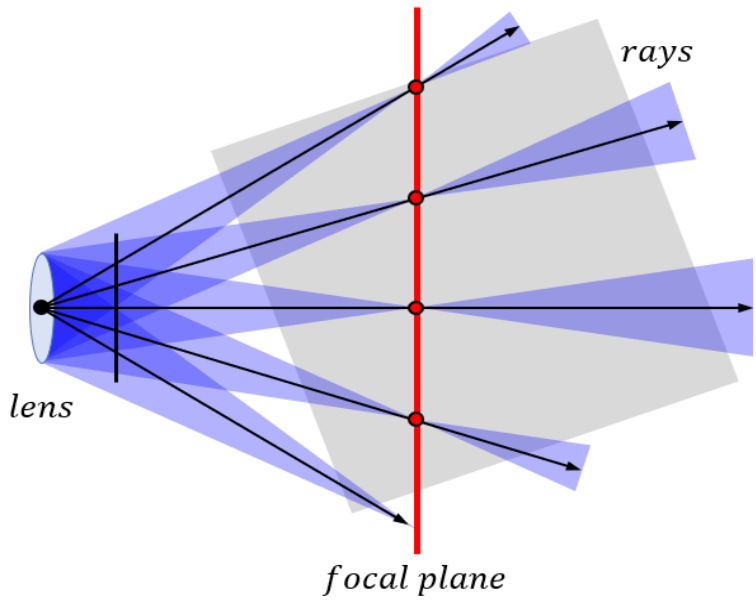


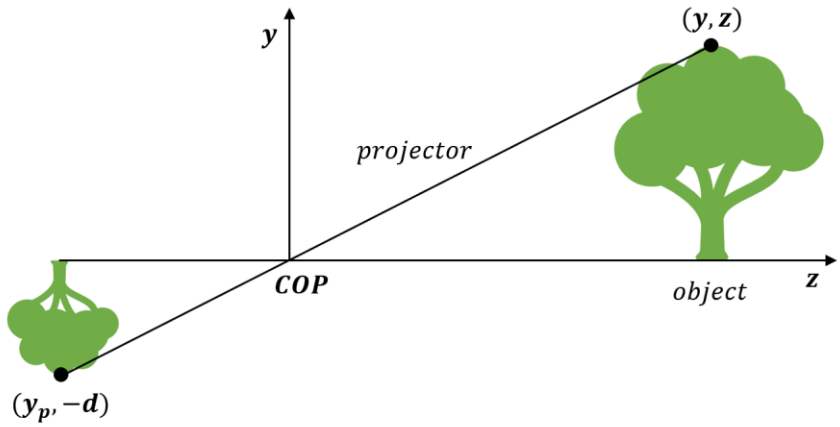
Figure 3.9 Depth of field volume ray casting. At each pixel, multiple rays passing through the blue region is computed. The color of the pixel is determined by averaging the result of the volume-rendering integral of all rays.

3.2 Geometry Setup

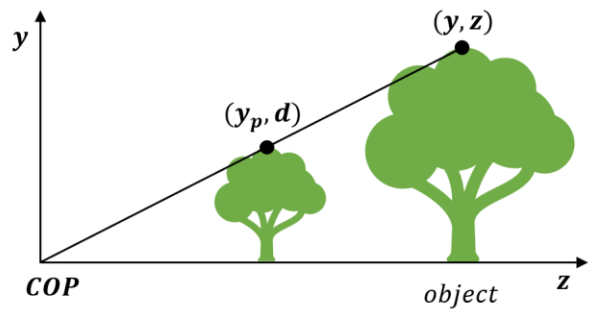
Integrating the DoF effects into the volume ray casting algorithm begins with placing the thin lens at the camera position and adding a focal plane. Figure 3.9 illustrates DoF volume ray casting using the thin lens model in 2D. The rays originating from the center of the lens are cast toward the volume, similar to the conventional volume ray casting. These rays are generated one by one at each pixel and intersect a single

point on the focal plane (the red line), called the focal point (red dots). Then, at each pixel, a thin lens is used to create additional rays which is passing through the focal point. The blue areas in Figure 3.9 represent all additional rays passing through the focal point and the lens. Since it is not possible to generate rays infinitely to capture all the blue areas, a limited number of rays is generated with sample points evenly distributed on the lens. In other words, an appropriate number of lens samples for quality and performance are generated and used for additional ray construction. At each pixel, multiple rays starting at different locations on the lens compute the volume-rendering integral, and these results are used to determine the final pixel color.

One thing that complicates the application of the thin lens is the position of the image plane. In the actual camera, the image plane and the object are located opposite each other with the lens centered as shown in Figure 3.10(a). By drawing a line called the projector from the center of projection (COP) or the center of the lens, you can find a point $(y_p, -d)$ on the image plane located at $-d$ that coincides with a point on the object (y, z). Figure 3.10(b) shows the case where the position of the image plane is changed. When the image plane moves to d, the point (y, z) of the object is formed at the point (y_p, d) where the projector meets the image plane. At this position, the result image is not flipped because the object and the image plane are in the same orientation with respect to the camera. Based on this fact, in computer graphics, the image plane is placed between the camera and objects to prevent the result image from flipping. Like real cameras, the thin lens model also has the image



(a)



(b)

Figure 3.10 Equivalent views of image formation. (a) Image formed on the back of the camera. (b) Image plane moved in front of the camera. Image courtesy of Angle and Shreinner [30].

plane and objects in opposite directions based on the lens. The application of the lens equation (3.1) can be confusing if the image plane is on the same side as the objects. Therefore, it is better to use the image plane only conceptually, and think of the focal plane acting as the image plane to generate rays per pixel [58]. In this way, we can use the DoF and CoC equations of the thin lens model as-is (Equation (3.2)-(3.5)) and obtain an upright image at the same time.

In the thin lens model, we generate samples on the lens to construct rays. Figure 3.11 shows how the thin lens model generates rays for one pixel in 2D. The focal plane is perpendicular to the optical axis and is located at the focal distance z_f, the distance that the viewer focuses on. First, we compute a ray that passes through the center of the lens and a point where the ray intersects the focal plane (a green line in Figure 3.11). This ray is named the *chief ray* and can be considered the same as the ray generated per pixel in conventional volume ray casting. The point where the chief ray intersects the focal plane is called the focal point, which is marked with p in Figure 3.11. Lens sample points are evenly distributed over the lens to represent all rays passing through the lens. The ray corresponding to each lens sample, named the *sample ray*, starts at the lens sample point and passing through the intersection p of the focal plane (blue lines in Figure 3.11). The sample ray configuration is the same as how to generate multiple rays from pixels for DoF rendering in ray tracing (ref. [27][58]). All sample rays compute the volume-rendering integral individually. Then, the results of the volume-rendering integral of all sample rays are averaged to determine the corresponding pixel color.

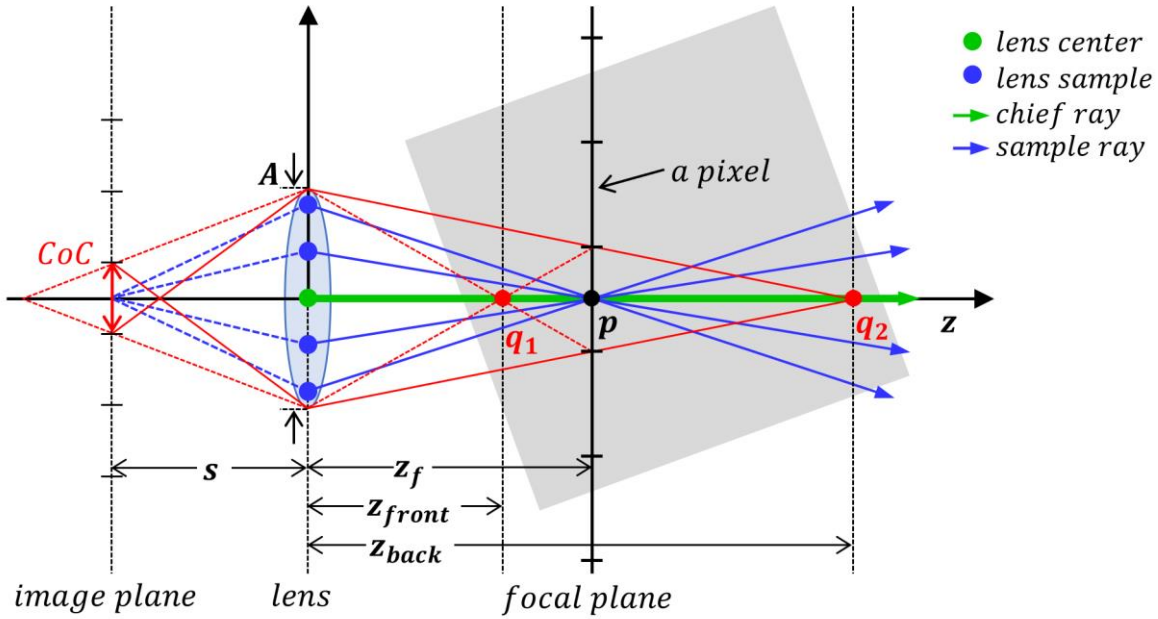


Figure 3.11 Volume ray casting based on the thin lens model for single pixel value. First, we find the intersection of the focal plane and the chief ray (point p). Then trace sample rays from lens samples to the intersection point p. The final pixel color is the average of the volume-rendering integrals of all sample rays. The distance interval $[z_{front}, z_{back}]$ where CoC is less than or equal to the pixel size is for three-pass progressive DoF rendering.

3.3 Lens Sampling Strategy

The processing time of DoF rendering using lens sampling increases in proportion to the number of lens samples. To get a high-quality DoF image without aliasing, we need to use a sufficient number of lens samples. Therefore, it is necessary to select an appropriate lens sampling technique that produces good quality images even with a small number of lens samples. To faithfully simulate the DoF effects of real cameras, we need to make the lens a disk. Of course, a very similar effect can be achieved by randomly selecting the origin of the rays in a square area [58]. In fact, many rendering techniques, including ray tracing that simulates the DoF effects, perform sampling on a square to generate multiple rays for a pixel. However, to increase the frame rate by using the minimum number of samples, it is necessary to distribute the samples evenly across a disk-shaped lens. Sampling on disk can be obtained by mapping the samples on the unit square to a unit circle. Therefore, we first focus on the unit square sampling techniques, then discuss disk mapping.

3.3.1 Sampling Techniques

To sample the lens, we need a sample pattern with low bias and good stratification. Sample patterns are divided into two categories: *sample sets* and *sample sequences*. Sample sets consist of finite unordered sample points and is used to render with a

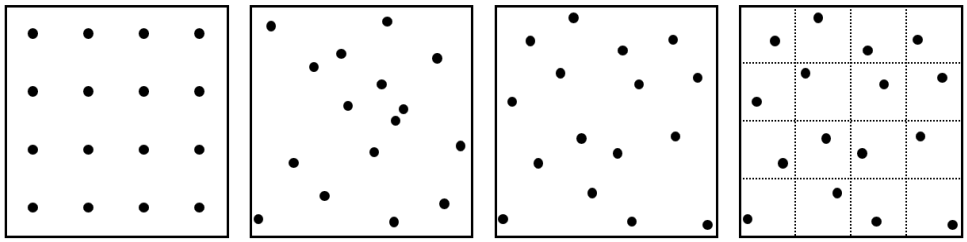


Figure 3.12 Sample pattern comparison between random and jittered samples (16 samples for a single pixel). (a) Regular samples. (b) Random samples. (c) Jittered samples. (d) Jittered samples with grid. Image courtesy of Marschner and Shirley [58].

fixed number of samples. Sample sequences consist of infinite ordered sample points and is used to render with a variable number of samples. The following describes sample patterns suitable for sampling the lens, by category.

The most straightforward way to create a sample set is *regular sampling*, which generates sample points at regular intervals as shown in Figure 3.12(a). One potential problem of regular sampling is that regular artifacts such as *moiré*① patterns can occur. Normally in computer graphics that use probabilistic sampling, *random sampling* is used to suppress artifacts. However, even with random sampling, the samples may clump together or have gaps as shown in Figure 3.12(b). Therefore, a technique is needed to distribute samples evenly while maintaining random

① A word meaning wave pattern, which is an interference phenomenon that occurs when patterns with regular intervals are repeatedly overlapped.

properties. One solution is to use a hybrid strategy that randomly perturbs a regular grid [58]. *Jittered (or stratified) sampling* is a sampling technique that divides a space evenly and generates samples randomly within each subspace (see Figure 3.12(c) and (d)). Samples produced by jittered sampling are still rather clustered. But, comparing (b) and (c) in Figure 3.12, jittered sampling shows a better distribution than random sampling [60]. Jitter sampling has been widely used for anti-aliasing and light sampling in surface rendering and is also known to produce good results when simulating the DoF effects. The number of samples as well as the distribution of samples is an important factor in rendering. In sampling for anti-aliasing, the number of samples per pixel can be determined by considering only the complexity of the scene and the resolution of the final image. However, in DoF rendering, the number of samples is highly dependent on the lens size as well as the aforementioned factors. This makes it difficult to determine the number of samples considering both performance and quality. For example, Figure 2.2 rendered by ray tracing used 128 samples per pixel and Figure 2.4 rendered by image accumulation used 23 lens samples. Therefore, to perform DoF volume rendering using a sample set, the challenge is to find the optimal number of samples that produce images of acceptable quality in real time.

The number of samples required for DoF rendering varies as the view changes. It is difficult to change the sample pattern every time the camera moves, depending on the complexity of the scene and the scale of the effect. This problem can be solved by using a *progressive (hierarchical or extensible) sample sequence*, in which any

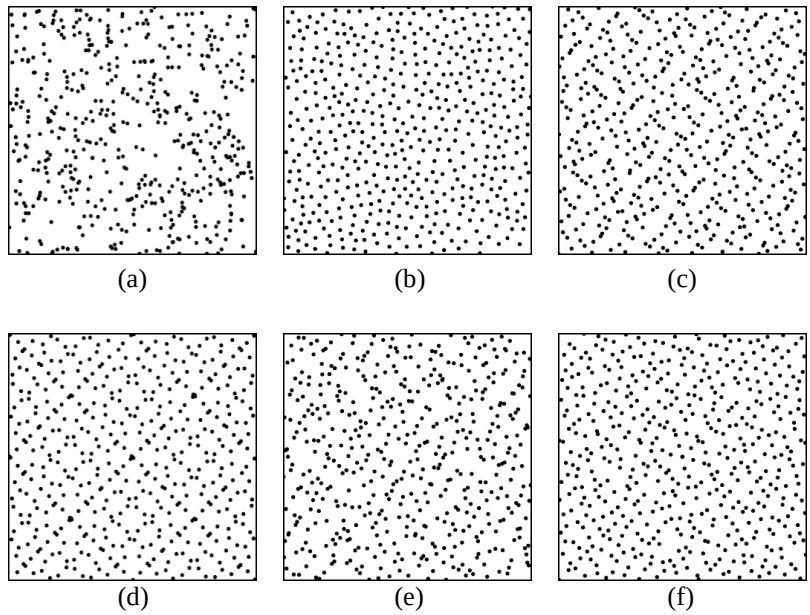


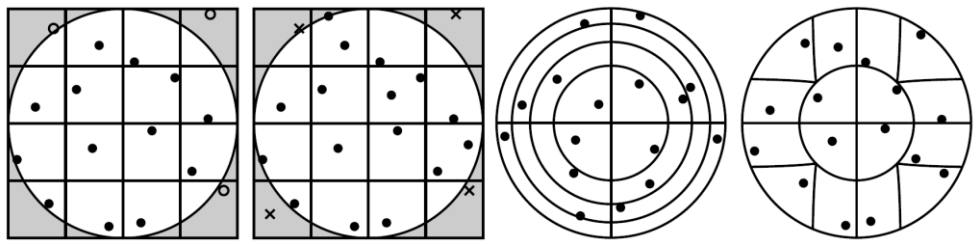
Figure 3.13 First 500 samples from six progressive sequences: (a) uniform random; (b) best candidates; (c) Halton with base 2 and 4; (d) Sobol'(0,2); (e) scrambled Halton; (f) Owen-scrambled Sobol'(0,2). Image courtesy of Christensen *et al*. [66].

prefix of full sequence is well-distributed [66]. So, we try to reduce oversampling by using progressive lens sampling. Figure 3.13 shows examples of the first 500 samples of representative sample sequences. The most basic sampling method is *uniform random* of which samples are generated by pairing two statistically independent pseudo-random numbers. As shown in Figure 3.13(a), this sample sequence shows a poor distribution with clumping. *Best candidates* [67] generates a random set of points and selects the candidate point with the largest distance from the closest existing point. This sequence gives good spacing, but the overall

distribution is uneven (see Figure 3.13(b)). *Halton sequence* [68] and *Sobol' sequence* [69] is quasi-random sequence generated by combing numbers with different bases. *Sobol'(0,2) sequence* is stratified in all elementary intervals in base 2. As shown in Figure 3.13(c) and (d), both the Halton samples and the Sobol'(0,2) samples have clumped points and systematic patterns. To avoid these problems, randomization is often used for quasi-random sequences. In Figure 3.13, (e) shows Halton sequence with random digit scrambling, and (f) shows Sobol'(0,2) sequence with *Owen scrambling* [70][71]. We can see that the systematic pattern is less noticeable and evenly distributed than before applying randomization. In previous studies [66][72], *Owen-scrambled Sobol'(0,2) sequence* is known to show the best performance in progressive rendering. In this paper, Owen-scrambled Sobol'(0,2) sequence is chosen for progressive lens sampling. Then, the optimal number of samples is determined through experiments on various volumetric data.

3.3.2 Disk Mapping

Figure 3.14 shows examples of different disk mapping strategies. The simplest way of disk mapping is to assign zeros to samples outside the unit circle (*Pad-zero*) or skip that samples (*Rejection*). These methods are simple and widely used, but notorious for degrading the continuity of random and evenly distributed sampling. *Concentric mapping*, which squeezes the corners of a square into a disk shape, is also a popular disk mapping, but it is not suitable for evenly sampling a disk. *Polar*

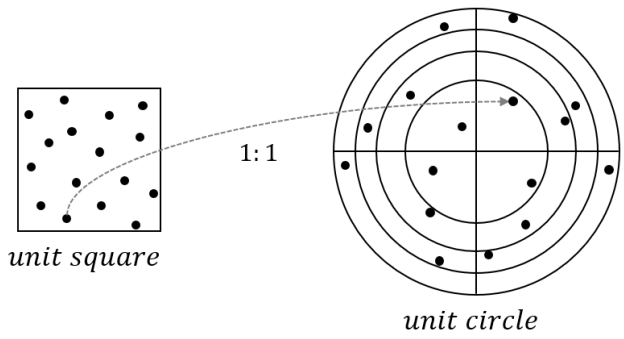


(a) (b) (c) (d)

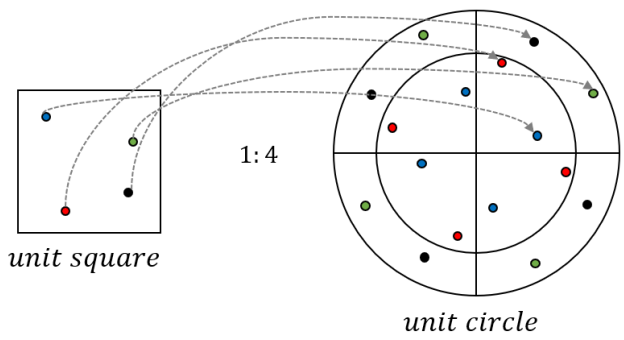
Figure 3.14 Disk sampling strategies: (a) pad-zero, (b) rejection, (c) polar mapping, (d) concentric mapping. Image courtesy of Christensen [72].

mapping is the most commonly used disk sampling that converts a point (u, v) on a unit square to a unit circle with $r = \sqrt{u}$ and $\phi = 2\pi v$. This method does not discard sample points, but creates a rather irregular pattern with a poor aspect ratio. *Polar4 mapping* is designed with some modifications to the polar mapping to overcome this problem. In polar4 mapping, samples are mapped from the unit square to the quarter-circle wedge in the first quadrant. Then, the samples in the first quadrant are rotated by 90, 180, and 270 degrees, adding new samples to the remaining quadrants. Figure 3.15 shows the process of polar and polar4 mapping. we can see that polar4 mapping maps the samples more evenly than polar mapping.

Even a sample pattern showing an even distribution on a unit square may have a poor distribution on unit circle after disk mapping. That is, the combination of sample pattern and disk mapping is important. Recently, Christensen [72] proposed a method for effectively sampling a disk light source in ray tracing. This study



(a)



(b)

Figure 3.15 Comparison of two disk sampling strategies. (a) Polar mapping. (b) Polar4 mapping.

confirmed that Owen-scrambled Sobol'(0,2) sequence was one of the best progressive sample sequences with a uniform distribution when used with the polar4 mapping. Based on Christensen's work, we expected this combination to provide the best results among progressive disk sampling strategies. Therefore, in this paper, we use the combination of polar4 mapping and Owen-scrambled Sobol'(0,2) sequence for progressive lens sampling.

3.4 CoC-Based Multi-Pass Rendering

GPU-based volume ray casting uses the front-to-back compositing to accumulate the volume-rendering integral. With this compositing scheme, ERT acceleration can speed up rendering significantly by taking advantage of the fact that the opacity of rays is often saturated before reaching the end of the volume. Our method applies ERT by default. However, this basic acceleration method alone cannot handle the amount of computation that increases in proportion to the number of lens samples. Therefore, we designed an acceleration technique, which is CoC-based multi-pass rendering with progressive lens sampling to use different numbers of lens samples per pixel. The progressive sample sequence and disk mapping mentioned in Section 3.3 generate lens samples evenly distributed over the lens, and the DoF volume ray casting of Section 3.2 is performed in multi passes on the GPU. This makes GPU memory access more efficient compared to computing multiple rays at once in a single pass. This section covers the sample sequence used in our DoF rendering method and how to determine the final lens pass per pixel in multi-pass rendering.

3.4.1 Progressive Lens Sample Sequence

Our method uses a progressive sample sequence, which is split into multiple render passes. The sample sequence is created in advance according to the minimum and maximum number of samples required for sampling the lens. The number of lens

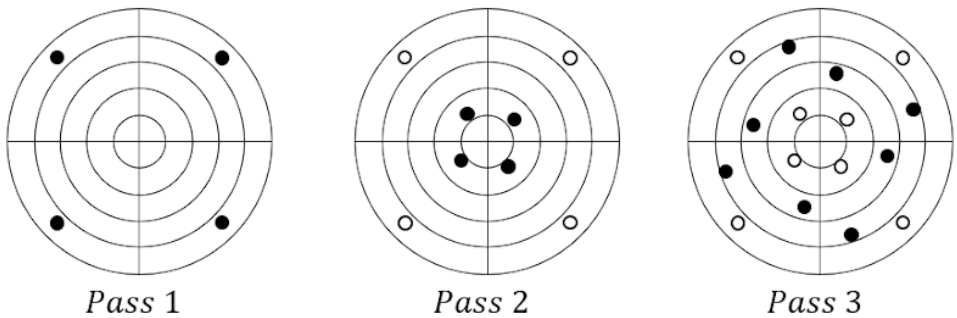


Figure 3.16 Progressive lens sample sequence for three-pass rendering. 16 samples generated with *Owen-scrambled Sobol'(0,2)* are mapped to a unit circle using *polar4 mapping*. Filled circles represent the sample points newly added in each render pass. Open circles represent the sample points used in the previous pass.

samples is determined by the characteristics of the data used. The minimum and maximum number of lens samples and the number of render passes were determined empirically by applying a various combination of number of samples to the volumetric data that are use a lot. The acceleration technique proposed in this paper uses 16 lens samples over three render passes and is scalable according to hardware performance (see Chapter 4).

The process of generating the progressive lens sample sequence is as follows. First, a progressive sample sequence consisting of the maximum required number of samples is generated with Owen-scrambled Sobol'(0,2) [73]. The generated sample sequence is mapped onto the lens by polar4 mapping. Then, the sample sequence needs to be split into subsequences with the number of render passes. The first

subsequence used for the first render pass has the minimum number of samples required to sample the lens. Then the remaining subsequences are sequentially assigned to the next render pass. Figure 3.16 shows the lens sample pattern for each render pass. The length of each subsequence is set to the number of samples accumulated up to the previous pass, which makes each render pass use twice the number of samples used up to the previous pass. The samples newly used in each render pass are shown as filled circles, and the samples already used in the previous passes are shown as open circles. Each render pass adds new samples to be distributed well including the previous ones, producing a natural result image even if the final render pass is different for each pixel.

3.4.2 Final Render Pass Determination

Each pixel in our method can have a different final render pass. The number of lens samples used for the DoF effects is determined by which pass the rendering ended. In the first render pass, the DoF effects are generated with the minimum number of lens samples. And in additional render passes, the aliasing is suppressed by adding lens samples to some pixels that are expected to have a large CoC. The final render pass of each pixel can be determined based on the expected maximum CoC size. As described in Section 3.1.2, the CoC size increases dramatically in front of the focal plane. Therefore, this area is most likely to form the largest CoC and should be rendered with enough lens samples to avoid aliasing. On the other hand, in the DoF

range where the CoC is smaller than the pixel size (the distance between q_1 and q_2 in Figure 3.11), all sample rays pass through the volume in a path similar to the chief ray. That is, if the boundary of the volume where the chief ray originates is in the DoF range, it is not necessary to generate as many sample rays. At a distance greater than the DoF, the CoC size slowly increases and the rays are likely to be saturated. The DoF effects in this distance is considered sufficient with a minimum number of lens samples. Based on these facts, we assume that for each pixel the chief ray has a maximum CoC size at the distance it intersects the volume boundary. When constructing the chief ray, GPU-based volume ray casting uses rasterization to calculate the start position at the volume boundary. Since we already know the distance of the nearest volume boundary for each pixel, we can easily estimate the maximum CoC size using Equation (3.2).

The final render pass of each pixel is determined by checking which distance range the start point of the chief ray is within. The condition that a pixel ends on the first render pass is when its chief ray starts at a point farther than the z_{front}. Pixels in this range show the object in focus or a small blur. The z_{front} can be obtained from Equation (3.3), which is the front boundary of the DoF of the thin lens model described in Section 3.1.2. Pixels with the start point of the chief ray closer than z_{front} perform a second render pass to reduce aliasing. The problem is that the size of the CoC increases dramatically as the object moves toward the camera from the focal distance z_f. Therefore, we add a distance boundary z_ρ using Equation (3.7) in front of z_{front}, so that the pixels whose chief ray starts near the camera perform

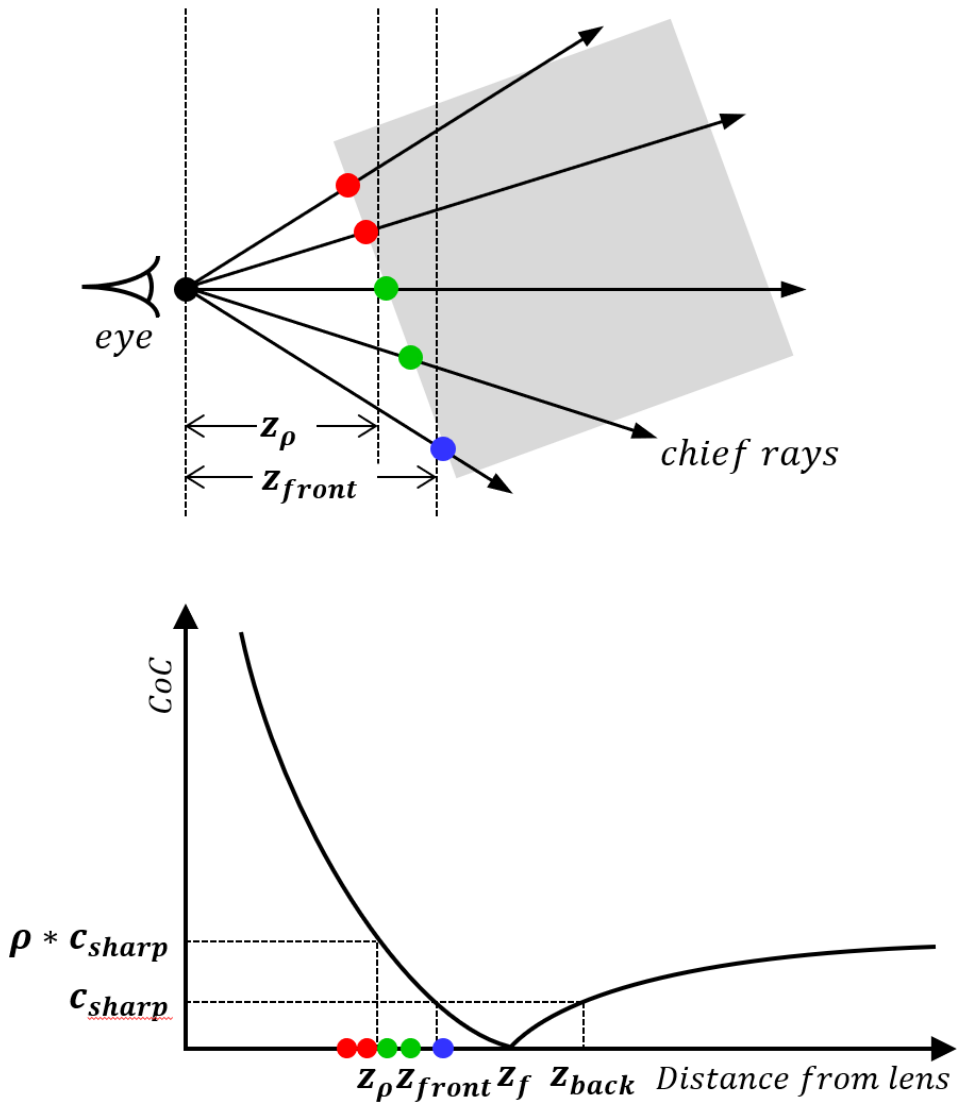


Figure 3.17 CoC-based progressive rendering determines the number of render passes, based on the start point of the chief ray in the CoC graph (bottom): blue points go through one pass, green points do two passes, and red points do three passes.

the third render pass. z_ρ is set by the user, and is calculated as follows:

$$z_\rho = \frac{Afz_f}{Af - \rho \times c_{sharp}(z_f - f)}. \tag{3.7}$$

$\rho\ (\geq 1)$ in Equation (3.7) is a constant that determines the level of progressive renderings. If the CoC is larger than ρ times the pixel size, the rendering goes up to three passes. ρ is empirically set to obtain an image without aliasing, depending on the data context and the transfer function used (described in detail in Section 5.2).

Figure 3.17 illustrates how to set up a final render pass based on CoC. The color of the starting point of the chief ray at each pixel represents the final render pass: 1st pass for blue, 2nd pass for green, and 3nd pass for red. If the start point of each pixel's chief ray is farther than z_{front}, only one pass is rendered. But if the start point of the chief ray is between $[z_\rho,\ z_{front}]$, it renders up to two passes; and if less than z_ρ, it renders three passes. Assuming that z_s in each pixel is the distance between the point where the chief ray meets the volume, the final render pass determination criterion for CoC-base multi-pass rendering can be summarized as follows:

$$\begin{aligned} &1^{st}\text{ pass: } z_s > z_{front}, \\ &2^{nd}\text{ pass: } z_\rho < z_s < z_{front}, \\ &3^{rd}\text{ pass: } z_s < z_\rho. \end{aligned} \tag{3.8}$$

The criteria for determining the final render pass are pre-calculated before rendering. The next chapter explains how this criterion is used in the GPU implementation.

CHAPTER 4

GPU IMPLEMENTATION

4.1 Overview

The DoF volume ray casting suggested in Chapter 3 can be implemented on both CPU and GPU, but GPU implementation is essential to obtain images in real time. Hence, the proposed method is based on the conventional GPU-based volume ray casting. The volumetric data and its transfer function are stored as textures in GPU memory, and the ray casting for each pixel is performed in the fragment shader. In addition to hardware interpolation of texture and parallel execution of fragment shaders, rasterization used to construct rays provides convenience in implementation and guarantees speed improvement.

In this chapter, in order to apply the DoF effects to the GPU-based volume ray casting, the process of generating sample rays with the focal distance and lens samples is added to the rendering pipeline. As described in Section 3.4, our method is designed with multi-pass rendering for acceleration. Except for the lens samples used, each render pass is performed in the same process. Therefore, the rendering process iterates over the number of the render passes, and the result is blended using a framebuffer object. This process is described in detail in Section 4.2. We use a

rasterization-based ray construction the same as in the accelerated GPU-base ray casting. This technique can be applied equally to calculating the focal point of each pixel at once, details are covered in Section 4.3. Also, since the start point of the chief ray calculated by rasterization is not located in the center of the lens, we describe how to recalculate the lens sample coordinates at the volume boundary in Section 4.4.

4.2 Rendering Pipeline

The proposed DoF ray casting algorithm has the same framework as GPU-based ray casting in computing a chief ray for each pixel. The only difference is that the final value of the pixel is determined by the sample rays, not the chief ray. To implement this, some processes have been added to the existing method to calculate the sample rays and the focal plane. Figure 4.1 shows a block diagram of progressive DoF volume ray casting proposed in this paper, which extends the GPU-based volume ray casting. The whole process consists of four steps, with the additional processes for the DoF effects shaded in Figure 4.1. Note that ERT, one of the ray casting acceleration techniques, is applied by default.

In the first step, the coordinates of the chief ray and focal plane are computed. The chief ray coordinates are obtained by rasterizing the front and back faces of a volume bounding box (VBB) with the corresponding distance to the camera [3]. All rays need to be represented with 3D texture coordinates between [0, 1] to

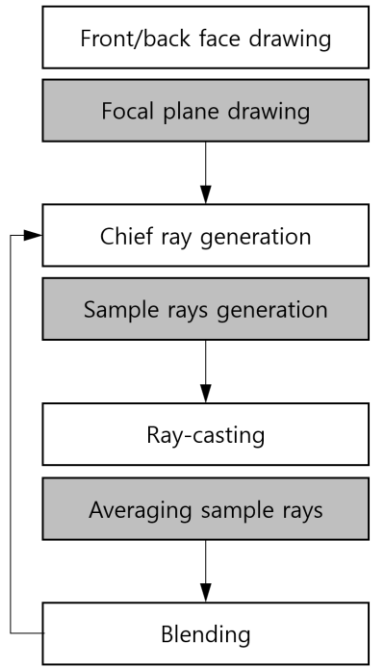


Figure 4.1 Rendering pipeline of the progressive DoF ray casting. Processes newly added to the conventional volume ray casting are shaded.

look up the volumetric data stored in the 3D texture. To represent the rays as the texture coordinates, we create a VBB of which the colors of the vertices are the 3D texture coordinates of the volume. Then in the view space, the front and back faces of the VBB are rendered into 2D textures. This process takes advantage of the GPU's hardware rasterization to quickly calculate the start and end positions of the chief rays that meet the boundaries of the volume. Considering for use in endoscopic views, the front faces of the VBB is rendered using culling boundary rasterization [74]. This technique treats the hole in the area where the near clipping plane meets the VBB.

The focal plane can also be calculated in the same way as the chief ray. For this, we create a quad that fills the entire screen. The color of each vertex in the focal plane is set by converting the focal distance to the 3D texture coordinates of the volume. Then, the quad is rendered into a 2D texture. Details are described in Section 4.3. Figure 4.2 illustrates the ray construction using rasterization and Figure 4.3 shows an example of the result textures. The volume entry and exiting positions of the chief ray can be obtained by rendering the front (blue line) and the back (green line) of the VBB respectively. The focal point of each chief ray is also obtained using rasterization of the focal plane (red line).

From the second step to the last step, each pixel is calculated within the fragment shader on GPU. The process is provided as pseudocode at the end of this section (see Algorithm 2). All render passes perform these steps identically, only the subsequence of lens samples used to generate sample rays is different. Therefore, by updating the lens sample coordinates, our method can be extended to a structure that performs more render passes through iterations with the same fragment shader.

The second step consists of generating sample rays using the chief ray and focus lens samples. The chief ray is computed by extracting the start point coordinates from the VBB front texture and the end point coordinates from the VBB back texture. Then, according to the criteria in Section 3.4.2, determine whether to perform the current render pass by checking the location of the start point of the chief ray. Out of execution conditions, the GPU fragment of this pixel is discarded. If the execution condition is met, the focal point coordinate is obtained from the focal plane texture.

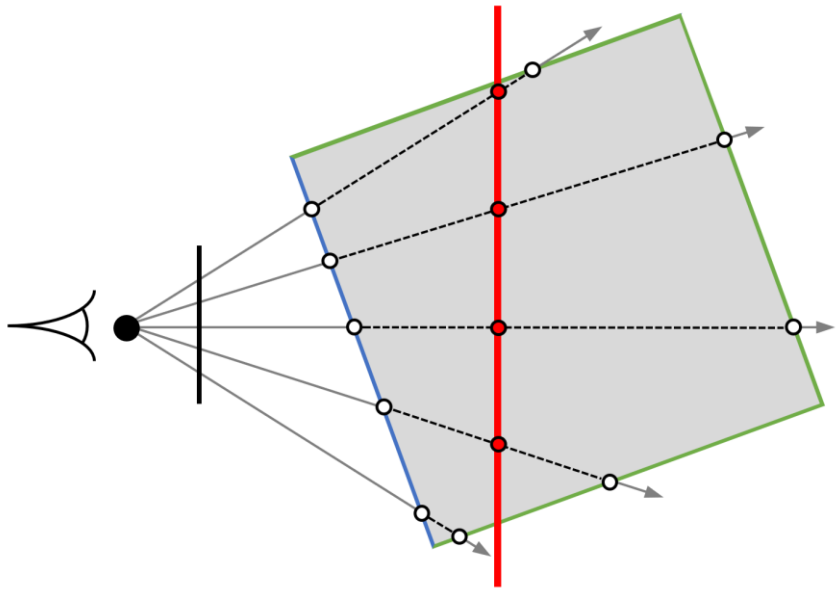


Figure 4.2 In ray casting, the coordinates of the chief rays (white dots) and focal point (red dots) are obtained through rasterization.

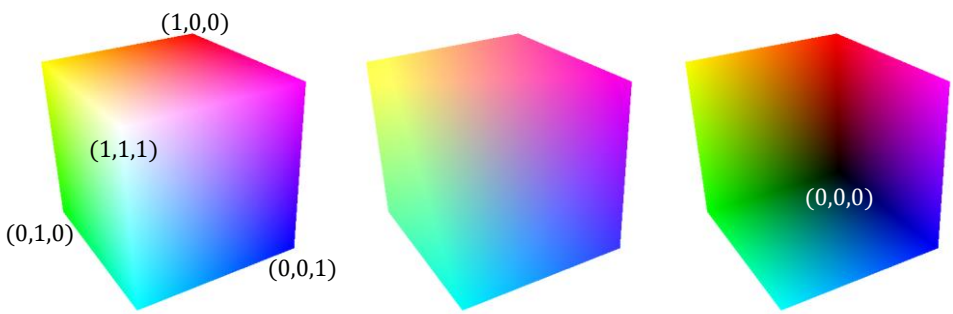


Figure 4.3 Rasterization of ray setup. The front face coordinates (left) yield chief rays with the back face coordinate (right). And the focal plane coordinates (middle) are used to calculate sample rays. In this example, the focal plane is located between the front face and the back face. 3D volume coordinated in $[0,1]^3$ are illustrated as RGB color.

And, sample rays are calculated starting from the lens samples and passing through the focal point. The lens samples of Figure 3.16 in Section 3.4.1 are used according to the current render pass. Lens samples coordinates are defined on the unit circle. So, these are scaled to the aperture size to find coordinates in 3D. However, since the start point of the chief ray is calculated at the volume boundary, the aperture size at the camera position cannot be applied as it is. Therefore, at each pixel, the relative size of the lens aperture at the start point of the chief ray is calculated. Further details are covered in Section 4.4.

In the third step, ray casting is performed for each sample ray. When sample rays are cast, their volume-rendering integrals are calculated and accumulated individually. In each ray casting iteration, the opacity of all sample rays is summed to ensure that they are saturated. The reason this approach is used is that checking the opacity of each ray individually would cause a lot of branches in the shader. The chief ray determines the maximum number of ray casting iterations. That is, if the sample rays are not terminated prematurely by the ERT, the ray casting loop ends when the chief ray exits the volume. When ray casting is finished, the results of all sample rays are averaged. Note that the chief ray is only used for geometry setup and is not used to calculate the final color of pixel.

The final step is to blend the final color of the current render pass into the render buffer. Since our method is multi-pass rendering, we store the result of each render pass in the framebuffer object instead of displaying it directly on the screen. Therefore, if the framebuffer object has the result of the previous render pass, it is

averaged with the result of the current render pass and stored in the render buffer. The same process is repeated going back to the second stage until the final render pass is performed.

Algorithm 1 Main Rendering Algorithm

```
      Data: volume, transferFunction, lensSamples
1     frontTexture, backTexture ← DrawFrontAndBackFaceTexture()
2     focusTexture ← DrawFocalPlaneTexture()
3     zFront, zRho ← Compute parameters as in Equation (3.3) and (3.7)
4     prevFrameBuffer, currentFrameBuffer ← 0
5     for currentPass = 1 to 3 do // 3-pass rendering
6         bind currentFrameBuffer
7         prevTexture ← texture from prevFrameBuffer
8         for all fragments f do
9             DoFRayCasting(volume, transferFunction, currentPass,
                            frontTexture, backTexture, focusTexture, prevTexture)
10        end for
11        unbind currentFrameBuffer
12        prevFrameBuffer ← currentFrameBuffer
13    end for
14    display prevFrameBuffer on the screen
```

Algorithm 2 DoFRayCasting (*volume, transferFunction, currentPass, frontTexture, backTexture, focusTexture, prevTexture*)

```
1   chiefRay ← calculate from frontTexture and backTexture
2   prevColor ← get from prevTexture
3   stopCondition ← check if currentPass has passed the final render pass
4   if stopCondition then
5       fragColor ← prevColor
6   p ← get from focusTexture
7   N ← number of lens samples
8   lensSampels[N] ← lens sample coordinates based on the lens aperture size
9   sampleRays[N] ← sample rays from lensSamples[N] passing through p
10  composited[s]rgba ← 0
11  while a ray sample position x of chiefRay is inside volume do
12      accumulateda ← 0
13      for each s = 0 to N do
14          y ← a ray sample position of sampleRays[s]
15          intensity, gradient ← Texture3D(volume, y)
16          shadedrgba ← Shading(TransferFunction, intensity, gradient)
17          composited[s]rgba ← Compositing(shadedrgba, composited[s]rgba)
18          accumulateda += composited[s]a
19          march sampleRays[s]
20      end for
21      if accumulateda >= (N – 0.1) then
22          goto line 25
23      end if
24      march chiefRay
25  end while
26  fragColor ← mean of composited[N]rgba and prevColor
```

4.3 Focal Plane Transformation

To generate sample rays at each pixel, we first need to know the coordinates of the focal point of the chief ray. The focal point of the chief ray can be obtained by calculating a point where the chief ray and the focal plane meet. Alternatively, it can be computed by projecting the focal point on the optical axis onto the direction vector of the chief rays. The latter method is easy to implement because the focal point on the optical axis is easily calculated from the camera's direction vector and the focal distance. However, these geometry approaches require adding multiple redundant vector operations for each pixel. Therefore, we chose a quick and simple method of getting focal plane coordinates using rasterization, just like when calculating the chief ray. This session details the process of calculating the focal plane coordinates in the first step of the DoF volume ray casting pipeline.

Figure 4.4 illustrates how to draw a focal plane that fills the screen in the view frustum. The focal point p on the optical axis is at the focal distance z_f in the direction the camera is looking at. When c is the lens center and v is the camera's view vector, the coordinates of the focal point in the view space can be expressed by $p = c + z_f \times v$. The focal plane is the plane perpendicular to the camera view vector v passing through the focal point p. The volume texture coordinates of the focal plane are obtained by drawing a quad that fills the screen. The texture image in the middle of Figure 4.3 is an example of the focal plane obtained by rasterization. The example shows only the area where the focal plane intersects the volume, but

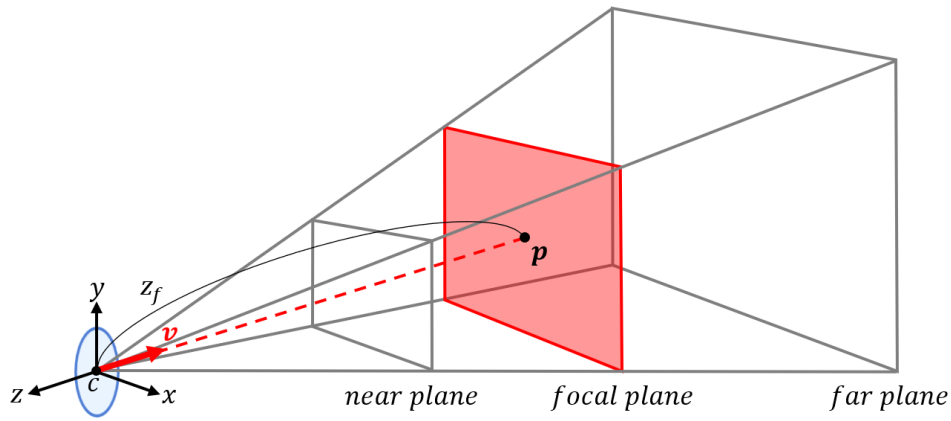


Figure 4.4 Illustration of the focal plane quad for calculating the focal point of all chief rays at once. The quad perpendicular to the camera view vector v is drawn to fill the screen at the focal distance z_f.

actually the area outside the volume displayed on the screen also has values. Areas outside the volume do not generate a chief ray, so the values of that area in the texture of the focal plane is not used.

Note that this approach assumes that the focal plane is always behind the near clipping plane. If the user places the focal plane in front of the near clipping plane, all sample ray is generated in the opposite direction of the camera. Adding a procedure to check if the focal plane is located closer than the near clipping plane will cause a branch divergence within the GPU pipeline, affecting performance. To avoid this, it is appropriate to limit the focal length so that the user enters a distance larger than the distance of the near clipping plane.

4.4 Lens Sample Transformation

As described in Section 3.3, lens samples are defined on a square or unit circle. Therefore, the coordinates of each sample need to be recalculated according to the size of the lens aperture received as user input. This session explains how to find the start points of the sample rays based on the start point of the chief ray.

The process of using lens samples defined on a unit circle to implement a thin lens model is as follows. Given a 2D plane consisting of the camera's x-axis and y-axis, the 2D coordinates of the lens sample can be considered as an offset from the origin. First, place the center of the unit circle at the camera's center of projection. And the lens sample coordinates are scaled by the lens aperture size. These scaled sample coordinates are used as the offset between the start point of the chief ray and the start point of the sample ray. The problem is that the start point of the chief ray of each pixel is not on the center of the camera. Our method does not put the start point of the chief ray on the camera, but calculates it at the volume boundary for each pixel using GPU rasterization. Therefore, in order to find the start point of the sample rays at each pixel, we need to calculate the relative size of the aperture at the start point of the chief ray.

Figure 4.5 shows the lens aperture size relative to the start point of the chief ray for a single pixel. The relative aperture size at the start point of the chief ray is computed as follows:

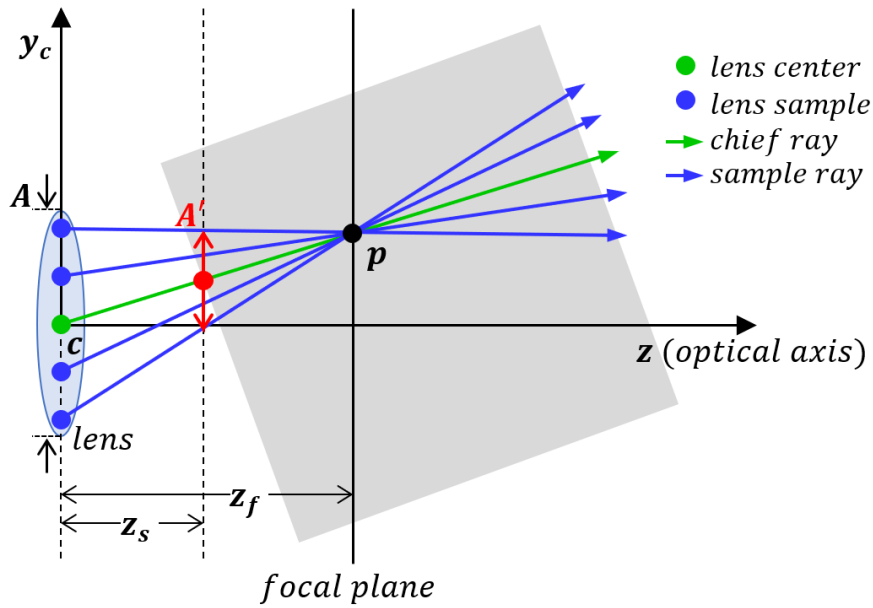


Figure 4.5 Relative lens aperture size at the start point of the chief ray. By scaling the coordinates of the lens samples defined in the unit circle by A', the offset of the sample ray can be obtained from the start point of the chief ray.

$$A' = A\frac{|z_f - z_s|}{z_f}, \tag{4.1}$$

where A is the aperture size, z_f is the focal distance, and z_s is the distance of the chief ray start point. Assuming that the start point of the chief ray (the red dot in Figure 4.5) is at the center of the unit circle, the sample coordinates on the circle scaled by A' is the 2D offset of the sample ray from the chief ray. And, we can calculate the 3D coordinates of the sample ray start point using the two vectors of the camera's x-axis and y-axis.

CHAPTER 5

EXPERIMENTAL RESULTS

Depth of field volume rendering was performed on volumetric data that is frequently used as test data in volume rendering literatures, and virtual endoscopic simulations with DoF was performed on a CT scan of the human body. We compared the proposed method with two other method: the DRT [16] and the DoF slice-based volume rendering [13]. The proposed method and the DoF slice-based volume rendering were implemented using OpenGL/GLSL and performed on a personal computer with an Intel i7-7700K 4.2GHz CPU (32GB RAM) and NVIDIA GeForce GTX 1080 GPU. DRT images were rendered using the Intel OSPRay [28][75] ray tracing engine on the same environment. All images were rendered at 512×512 resolution and a simple one-dimensional transfer function used by default for each volumetric data was applied. The DoF effects of the DoF slice-based method [13] were generated by taking 2×2 samples for blurring during slice compositing. Perspective projection and *Phong shading* are applied by default. In all methods, gradients for shading were estimated on the fly.

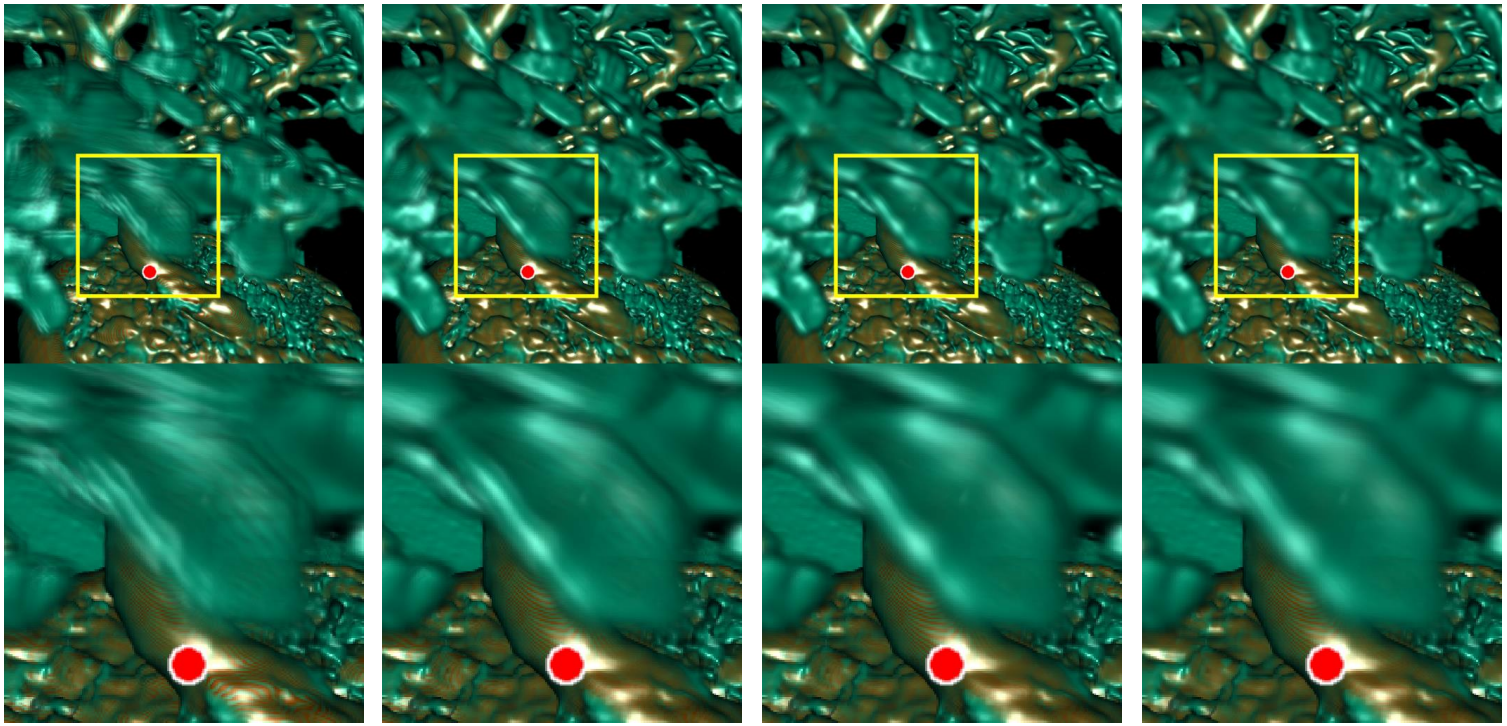
Our DoF volume rendering algorithm aims to generate high-quality DoF images in real time. There are three factors that determine quality and rendering speed: the number of lens samples, the number of render passes, and parameters that determine the final render pass of each pixel. This chapter finds the optimal values of these

three factors through experimentation. And we evaluate our method in terms of image quality and rendering speed compared to the previous DoF rendering techniques.

5.1 Number of Lens Samples

The number of lens samples is a factor that most affects image quality and performance. Direct volume rendering, as described in Section 3.1.3, computes the volume-rendering integral through sampling the viewing ray. This means that an additional ray per pixel adds a huge amount of computation. Therefore, if a large number of lens samples are used to generate a good quality DoF image, it is difficult to render in real time. Finding the optimal number of samples that guarantees image quality is an important task. The number of lens samples is determined by the size of the lens aperture and the focal distance. Unlike surface rendering, which often composes a large scene containing multiple objects, volume rendering mainly renders one volumetric data, so the camera setting is somewhat limited. Therefore, the optimal number of lens samples can be determined empirically.

We set up an experiment to find the optimal number of samples in single-pass DoF volume ray casting. Since the disk mapping technique used to generate the lens samples produces 4 samples at once, we experimented with increasing the number of samples by 4 (see Section 3.3.2). Figure 5.1 shows images of the bonsai data set rendered by single-pass DoF volume ray casting with various sample counts. The



(a) 147.71 fps (b) 56.44 fps (c) 29.24 fps (d) 15.25 fps

Figure 5.1 Renderings of bonsai data set (256^3 voxels) generated by DoF volume ray casting. (a), (b), (c) and (d) single-pass DoF volume ray casting with 4, 8, 12, and 16 lens samples, respectively. The bottom row shows a close-up image of the yellow square region in the top row. The red dot indicates the point in focus.

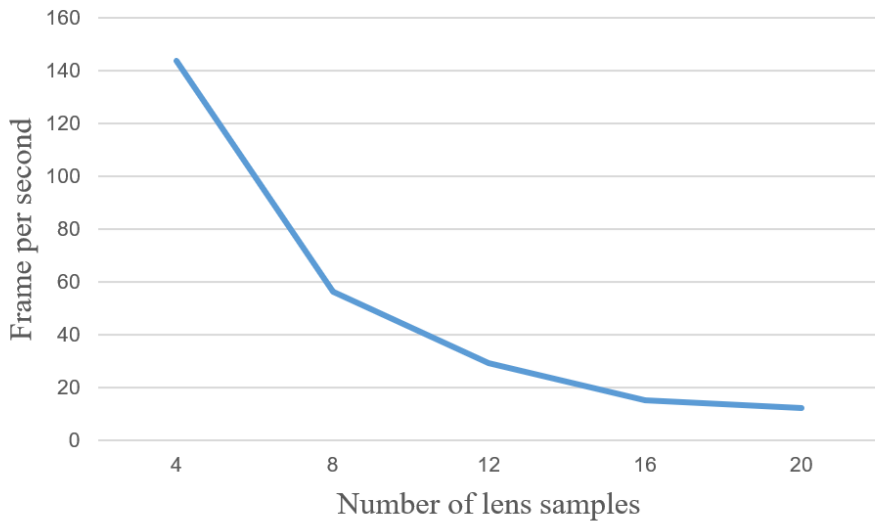


Figure 5.2 Frame rate according to the number of lens samples in single-pass DoF volume ray casting.

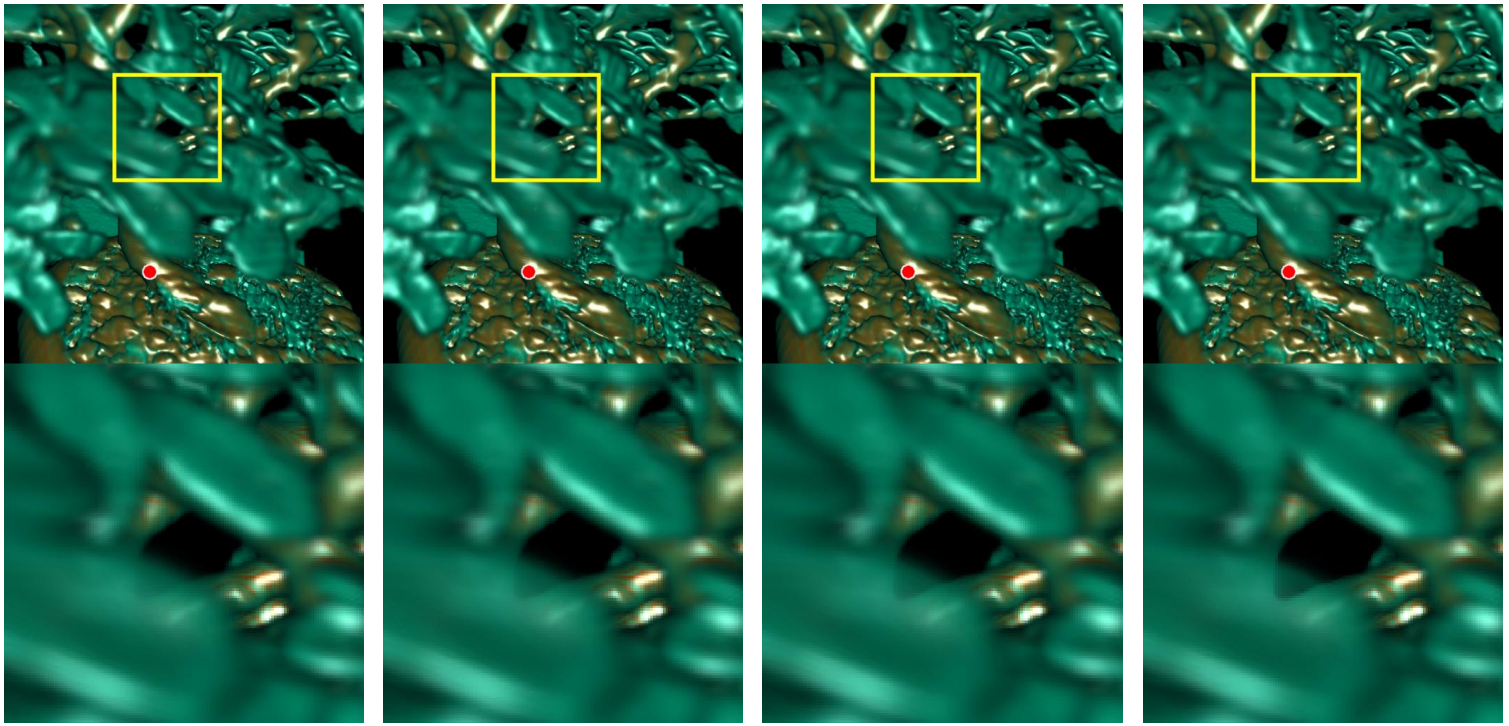
number of lens samples used in (a), (b), (c) and (d) are 4, 8, 12 and 16 respectively. The focus is on the tree trunk so that the leaves closer to the camera are blurred. As shown in the close-up images of Figure 5.1, increasing the number of lens samples reduces aliasing and produces more natural blurs. Figure 5.2 is a graph showing the change in frame rate according to the number of lens samples. In the experiment, we performed DoF rendering using up to 20 lens samples. The graph shows the frame rate drop that is inversely proportional to the number of lens samples. Based on this, we set the optimal number of lens samples to 16, which shows the aliasing-free DoF effects at an acceptable frame rate. Figure 5.1(d) using 16 lens samples shows a rather low frame rate, but will be accelerated in the next section.

5.2 Number of Render Passes

Our DoF rendering algorithm is accelerated by multi-pass rendering, which uses a different number of lens samples for each pixel. Each pixel's process ends in the different render pass based on its expected maximum CoC size (see Section 3.4 for details). In this session, the progressive sample sequence for sampling lens is divided into subsequences to determine the number of render passes through experimentation.

The increment of the number of lens samples is 4. This is because polar4 mapping is used, which generates 4 samples at a time. We designed the experiment with a lens sample sequence consisting of 16 samples for 1, 2, 3, and 4 render passes. Figure 5.3 shows the experimental results for various number of render passes, and the used subsequence of lens samples is as follows: (a) 16 samples in a single pass, (b) 8 samples each in two render passes, (c) 4 samples in the first and second render passes, and 8 samples in the third render pass, (d) 4 samples each in four render passes. It shows that using multiple render passes increase the frame rate when the number of samples is fixed. This is because the number of rays calculated at once is reduced, which increase the cache hit rate during the texture lookup. Therefore, even if most of the pixels in the result image have the same final render pass, multi-pass rendering is more effective than single-pass rendering.

Increasing the number of render passes can reduce precision due to repeated averaging of the sample rays. As seen in the close-up image of Figure 5.3(d), the boundaries of the object may be slightly shrunken or translucent. Figure 5.3(c) with



(a) 15.25 fps (b) 28.54 fps (c) 32.77 fps (d) 37.74 fps

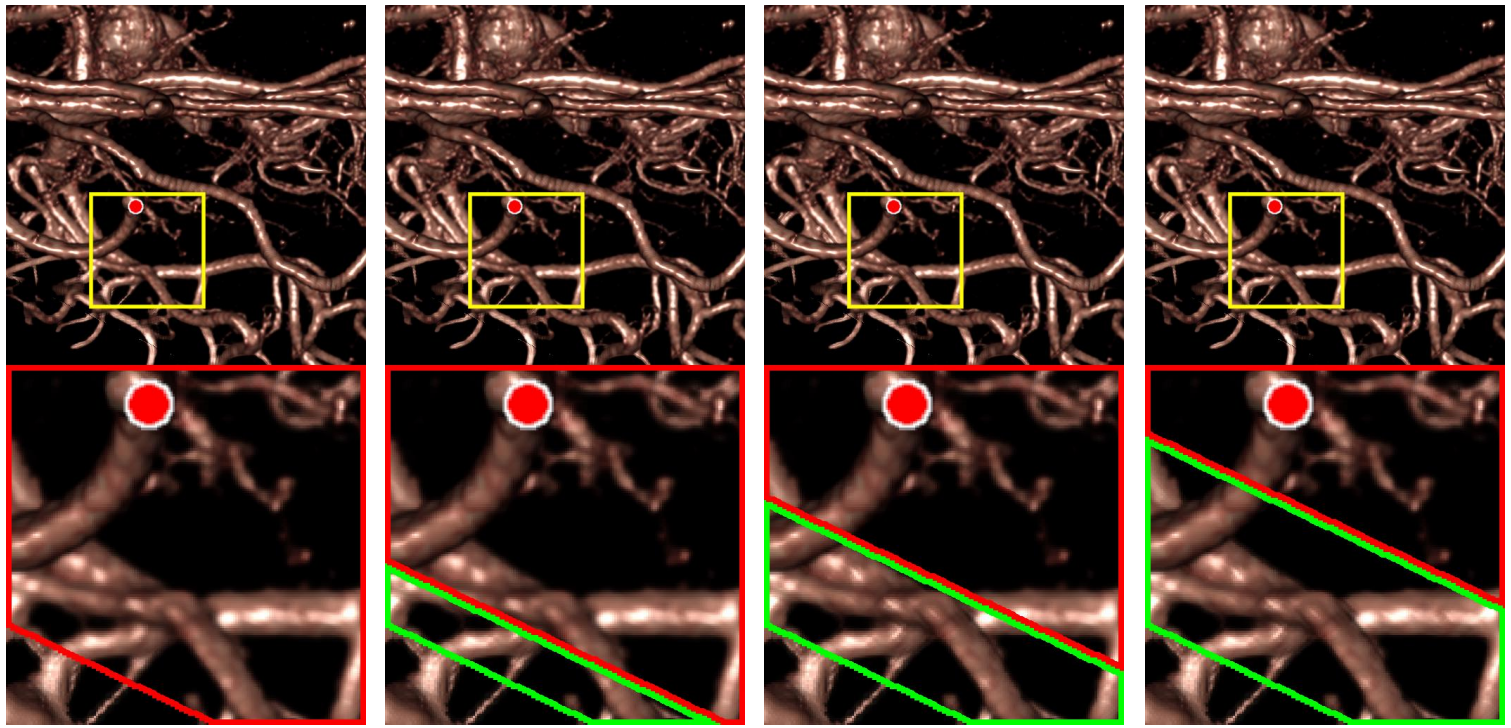
Figure 5.3 Renderings of bonsai data set (256^3 voxels) generated by progressive DoF volume ray casting with 16 lens samples. The number of render passes for (a), (b), (c) and (d) are 1, 2, 3 and 4 respectively. The bottom row shows a close-up image of the yellow square region in the top row. The red dot indicates the point in focus.

three-pass rendering shows that the frame rate is twice that of Figure 5.3(a) with single pass rendering while ensuring image quality. Therefore, we concluded that three-pass rendering adequate to minimize the loss of image detail and show sufficient speedup.

5.3 Render Pass Parameter

As mentioned in Section 3.4, the final render pass of each pixel is determined by the distance at which the chief ray enter the volume. If this distance is farther than the z_{front}, the final pixel value is determined on the 1st render pass. z_{front} is criterion for separating the 1st render pass and the 2nd render pass. And z_ρ is the boundary between the 2nd render pass and the 3rd render pass. z_{front} is given by the thin lens model equation, but z_ρ is calculated by Equation (3.7) using the parameter ρ set by user. This section describes a parameter study to find the optimal value of ρ.

Figure 5.4 shows the change in image quality and frame rate as the parameter ρ in Equation (3.7) changes. The yellow square in each image in the top row of Figure 5.4 contains pixels whose final render pass varies greatly depending on the ρ value. In the bottom row images of Figure 5.4, the area enclosed by the red border represents the pixels rendered in up to the 3rd render pass using the maximum number of lens samples (total 16 samples). The area enclosed by the green border contains the pixels rendered in up to the 2nd render pass (total 8 samples). The areas not enclosed by borders contain pixels that end in the 1st render pass. When the value of



(a) 25.57 fps (b) 25.97 fps (c) 26.47 fps (d) 27.05 fps

Figure 5.4 Renderings of aneurysm data set (256^3 voxels) generated by progressive DoF volume ray casting. The ρ values for (a), (b), (c), and (d) are 1.0, 1.5, 2.0, and 2.5, respectively. The bottom row shows a close-up image of the yellow square region in the top row. The areas enclosed by the green and red borders in the bottom row rendered up to the 2nd render pass and up to the 3rd render pass, respectively. The red dot indicates the point in focus.

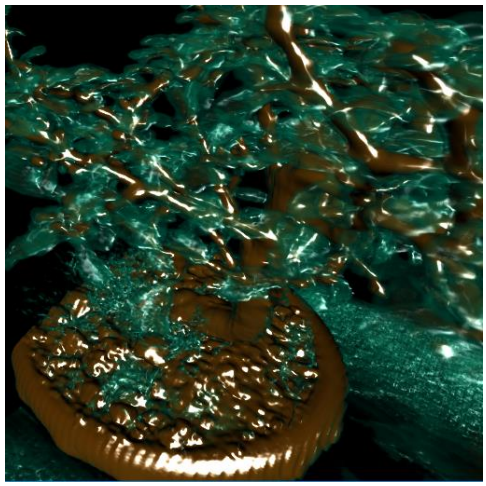
ρ is 1.0, pixels with a maximum CoC size larger than the pixel size are rendered up to the 3rd render pass (see Figure 5.4(a)). $\rho = 2.0$ means pixels with a maximum CoC size larger than twice the pixel size are performed up to the 3rd render pass, and pixels with a maximum CoC size is larger than the pixel size and less than twice the pixel size are performed up to the 2nd render pass (see Figure 5.4(c)). As the value of ρ increases from left to right, the final render pass for some areas changes from the 3rd render pass to the 2nd render pass. The frame rate slightly increases with ρ, but the image quality deterioration is not noticeable. This is due to the fact that the number of lens samples is determined relative to the volume boundary as shown in Figure 3.17, whereas the voxels affecting the result image are concentrated near the center of the volume.

The frame rate of multi-pass rendering is more affected by increasing the number of render passes than by using a different number of lens samples for each pixel. And it turns out that every pixels of the result image have been performed up to the 3rd render pass in more scenes than expected. To tighten the criteria for determining the final pass, it is more effective to find the closest voxel that is actually rendered. We can consider more accurately estimating the maximum CoC size of each pixel by pre-rendering the depth map of the first hit voxel using the chief ray. However, this approach can produce erroneous images because it does not include cases where the sample ray first refers to the volume region closer to the camera than the chief rays. In conclusion, we decided to use a fixed ρ value of 1.4 in the proposed method, rather than giving up accuracy to improve performance.

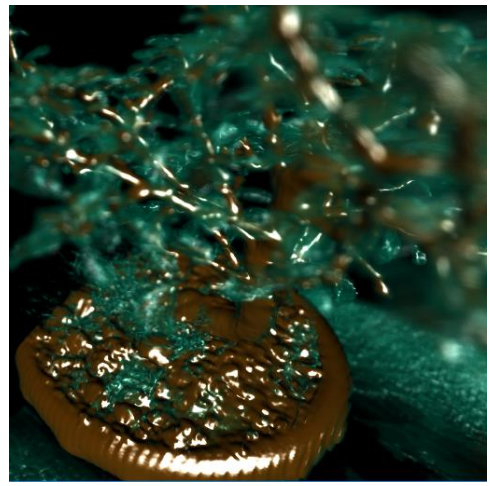
5.4 Comparison with Previous Methods

There are user studies evaluating the accuracy and response speed of depth perception for the DoF effect using an eye-tracking system [56], but no studies present a metric to assess the quality of the DoF effect. Most DoF rendering papers have evaluated the quality and rendering speed of DoF images compared to previous studies. In the field of surface rendering, a lot of research has been done on the DoF effect, which can be compared to various techniques to evaluate its quality. On the other hand, the only comparable research in the field of volume rendering is Schott's DoF rendering method based on slice-based volume rendering [13]. Therefore, we compared our method with the ground truth image generated by DRT to make sure it produces the correct DoF effects. And, we demonstrated the superiority of our method in quality and rendering speed compared to Schott's method, which is the only way to produce the most faithful DoF effects in volume rendering. On various volumetric data set, our DoF rendering method showed results similar in quality to images generated by DRT. And our method produced a faster, more realistic DoF effect than Schott's method, and showed high performance in specific-purpose medical data.

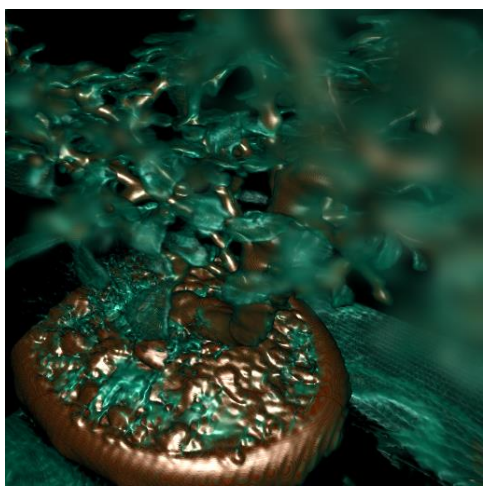
Figure 5.5 shows the results of the three DoF rendering methods performing the same DoF effects on the bonsai data set. Figure 5.5(a) is a pinhole image, and Figure 5.5(b) is created by DRT with the DoF effects [16]. Unlike Figure 5.5(a), which is focused on all distances, the leaves close to the camera in Figure 5.5(b) appear blurry



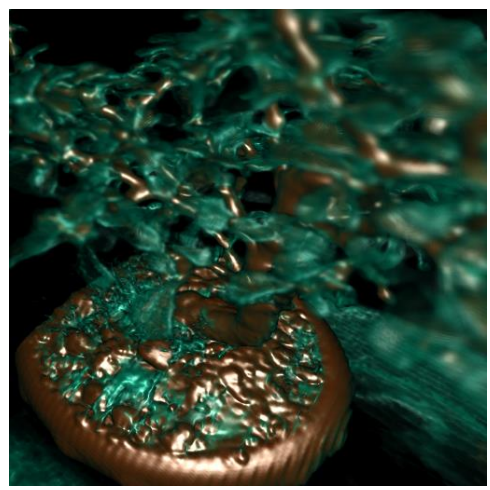
(a) 2.29 fps



(b) 2.42 fps



(c) 21.42 fps



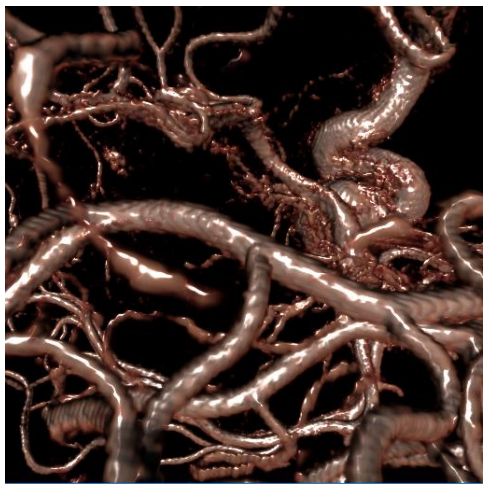
(d) 22.44 fps

Figure 5.5 Renderings of bonsai data set (256^3voxels) generated by (a) DRT with a pinhole camera, (b) DRT with the DoF effects [16], (c) DoF slice-based volume rendering [13] with $\alpha = 88^\circ$ and 2×2 samples taken during the incremental filtering, and (d) progressive DoF volume ray casting with $A = 0.03$. The focus in the images, except for (a), is on the bottom of the tree trunk and the depth of field is similar. In each image, gradients were computed on the fly.

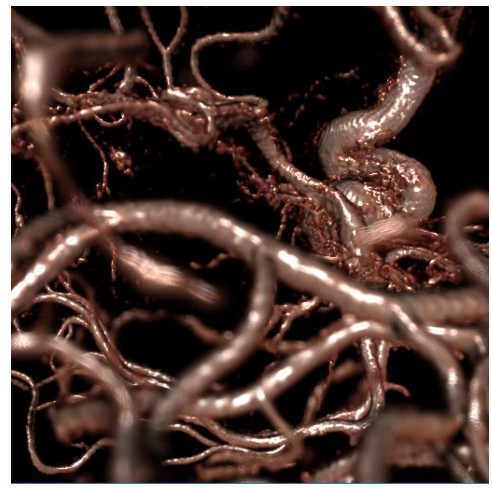
because the focus is on the tree trunk. As shown in Figure 5.5(d), the proposed method provides natural DoF blurring on out-of-focus leaves with a frame rate approximately 9 times faster than the ground truth image (Figure 5.5(b)). In contrast, in Figure 5.5(c) generated by the DoF slice-based volume rendering [13], most of the leaves out-of-focus are over-blurred. As authors mentioned in their paper [13], this is due to the incremental filtering mechanism in which previously synthesized slices continue to blur during the slice compositing.

Figure 5.6 shows the rendering results for the aneurysm data set. The data consists only of vascular bundles, and the associated voxels are mostly mapped to one color by the transfer function. Therefore, it is very difficult to recognize the context between complex blood vessels in a pinhole image as shown in Figure 5.6(a). In contrast, DoF images (Figure 5.6(b)-(d)) generated with focus on the thickest blood vessel at the top right of the screen enhance depth perception compared to the image using a pinhole camera. Schott's DoF method [13] (Figure 5.6(c)) and Our DoF method (Figure 5.6(d)) produces the DoF effects as realistic as DRT method (Figure 5.6(b)) and render them considerably faster (approx. 10 times). However, in Figure 5.6(c) rendered by the DoF slice-based volume rendering [13], there is a problem that blood vessels at the bottom of the screen are excessively blurred. On the other hand, in the Figure 5.6(d) created by our proposed method, the blood vessels are properly blurred with distance.

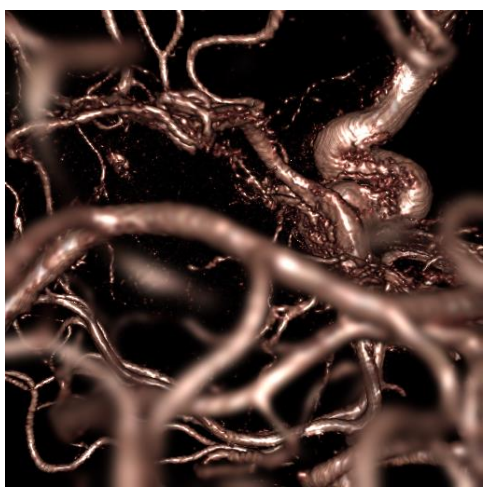
In Figure 5.5 and Figure 5.6, our method does not show a significant speedup compared to Schott's method [13] based on texture-based volume rendering.



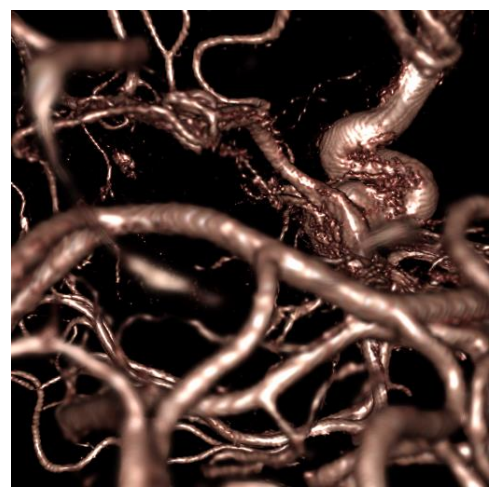
(a) 2.49 fps



(b) 2.44 fps



(c) 23.26 fps



(d) 25.15 fps

Figure 5.6 Renderings of aneurysm data set (256^3 voxels) generated by (a) DRT with a pinhole camera, (b) DRT with the DoF effects [16], (c) DoF slice-based volume rendering [13] with $\alpha = 70°$ and 2×2 samples taken during the incremental filtering, and (d) progressive DoF volume ray casting with the aperture size $A = 0.025$. The focus in the images, except for (a), is on the thickest blood vessel at the top right of the images and the depth of field is similar. In each image, gradients were computed on the fly.

In Figure 5.5, many voxels in the bonsai data set (green leaves) are mapped to translucent colors, which does not taking advantage of ERT. Therefore, the improvement in rendering speed of our proposed progressive method (Figure 5.5(d)) is not noticeable compared to the slice-based method [13] (Figure 5.5(c)). And, the reason that the effect of ERT does not appear in Figure 5.6 is that the ratio of objects drawn on the screen is small. The part that appears black in the image is mapped to zero opacity by the transfer function, so the ray is traversal to the end. To reduce this unnecessary operation, there is an acceleration technique called *empty space skipping*. However, it requires pre-processing with additional data structures and has the overhead of having to recalculate every time the transfer function changes. Therefore, when using an opaque transfer function or when the target object fills the screen, we can take advantage of our method without using additional data.

The proposed method is particularly useful when rendering complex organs such as blood vessels or endoscopic images in medical images. In the vessel visualization of Figure 6.4, we confirmed that the context can be grasped with the DoF effects without the need to rotate the view. Our method produces a much more reliable, high-quality image than the previous study [53] that mimic the DoF effects mentioned in Section 2.2. Endoscopic images that visualize organs such as the colon and brunch can be confusing to perceive depth because the organs are continuous and have a similar shape. In this case, other depth-enhanced volume rendering techniques, such as depth darkening or line drawing mentioned in 1.1, are not suitable. Rather, it can hinder depth perception by emphasizing unrelated parts.

Therefore, the DoF effect is the most effective way to provide depth information in endoscopic images. Figure 5.7 and Figure 5.8 are examples showing that the proposed DoF technique produces accurate DoF effects for endoscopic images at the frame rates available in immersive systems.

Figure 5.7 is a best example showing the advantages of the proposed method, as the long tube shape of the colon shows the DoF effects well. In Figure 5.7, the images in the top row were rendered by slice-based volume rendering, and the images in the bottom row were rendered by volume ray casting. Figure 5.7 (a) and (e) are pinhole images, and the rest are DoF images. For the DoF images, the focal distance is the same and the focus is on the protruding portion between the small pouches and the haustra in the center of the screen. This example is to examine the blurring change by increasing the size of the lens aperture while the focus is fixed. Figure 5.7(b)-(d) rendered by the slice-based method used incremental filtering [13] of different CoC angles α. And, the lens aperture size A of Figure 5.7(f)-(h) rendered by the proposed method was set to have the same DoF as in the DoF slice-based image in the same column. In the DoF images of the proposed method, the white implant and the lining of colon are blurred gradually as they move away from focus. And, as the lens aperture increases from (f) to (h) in Figure 5.7, blurring in out-of-focus areas naturally increases. In contrast, in the images of the DoF slice-based method, the out-of-focus areas appear to be somewhat discontinuously blurred as the CoC angle increases from (b) to (d) in Figure 5.7. The results show that volume ray casting using lens sampling has better control over the DoF effects than slice-

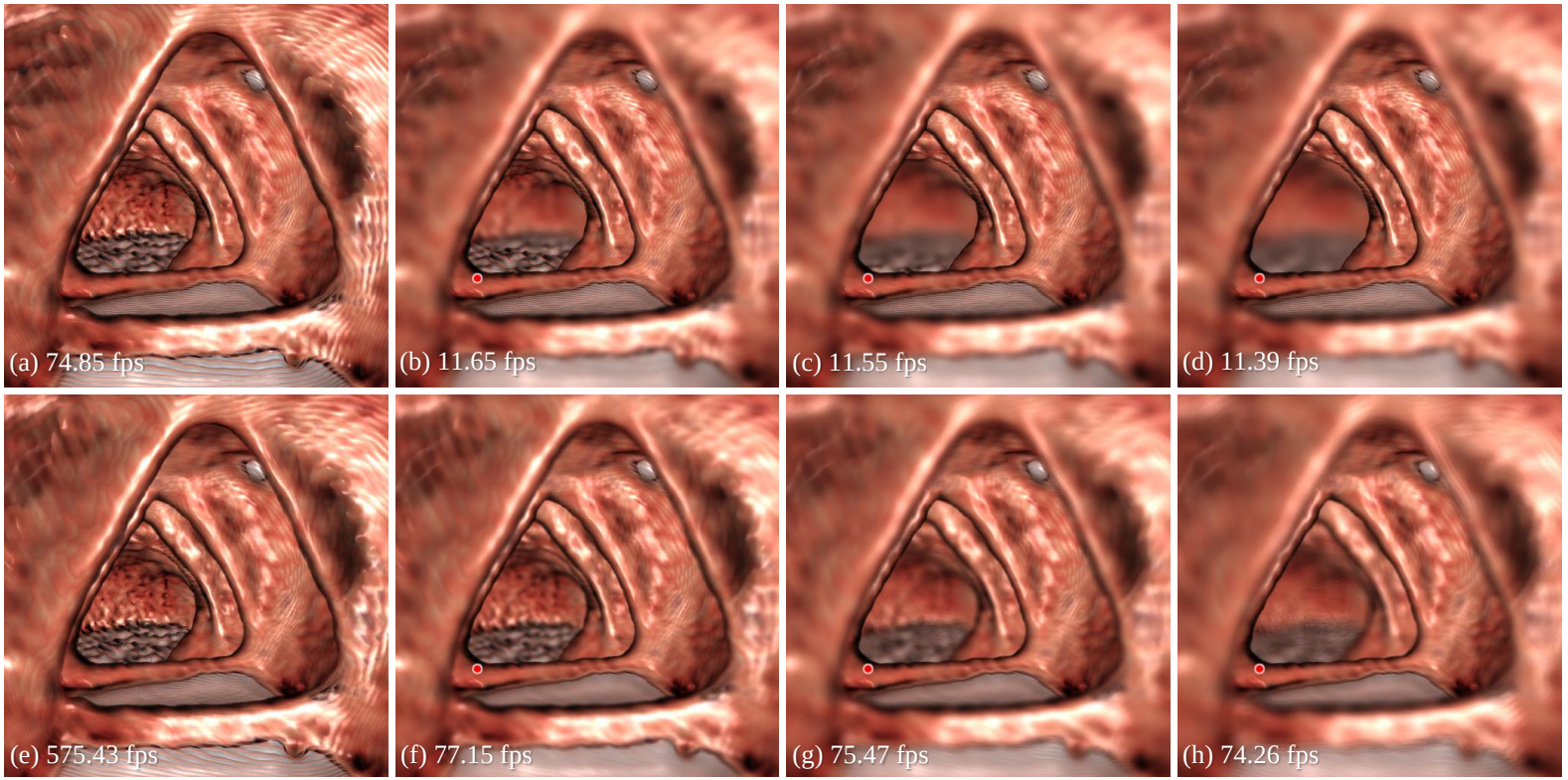


Figure 5.7 Renderings of colon CT data set ($512 \times 512 \times 672$ voxels). (a) Slice-based volume rendering with pinhole camera. (b), (c), and (d) DoF Slice-based volume rendering [13] with CoC angles α of $100°$, $110°$, and $120°$, respectively. (e) Volume ray casting with pinhole camera. (f), (g), and (h) Progressive DoF volume ray casting with aperture sizes A of 0.01, 0.02, and 0.03, respectively. The red dot indicates the point in focus.

based volume rendering using incremental filtering. In addition, the progressive DoF method shows a frame rate up to 6 times faster than the DoF slice-based method. In Figure 5.7, the camera is placed inside the volume to virtually navigate the interior of the colon, delivering the virtual colonoscopy images. Because this virtual endoscope creates an image that fills the entire screen, rays at each pixel are rapidly saturated in the ray casting technique. On the other hand, in the DoF slice-based rendering, a lot of slices are still required to be incrementally filtered for the rendering of enlarged small region, resulting in significantly reduced rendering speed.

Figure 5.8 shows images of a bronchoscope, another example of a virtual endoscope. With the size of the lens aperture fixed, mages were rendered with different focal points. This is to test whether the DoF blur changes naturally according to the change in focus. In the top row of Figure 5.8, the images were rendered by the slice-based method using pinhole camera and the DoF slice-based rendering using incremental filtering [13] (see Figure 5.8(b)-(d)). The focal distance is shortened from (b) to (d) in Figure 5.8: the focus in (b) is on the end of the left primary bronchus, and the focus in (d) is near the camera. When trying to set the DoF to a short range, the DoF slice-based method [13] tended to cause excessive blurring in out-of-focus areas. In the bottom of Figure 5.8, the images were rendered by volume ray casing using pinhole camera and our progressive lens sampling (see Figure 5.8(e)-(h)). The focal distance of each image is the same as in the DoF slice-based rendering image in the corresponding column. Comparing corresponding image pairs of the same focus, the proposed method produces images with improved

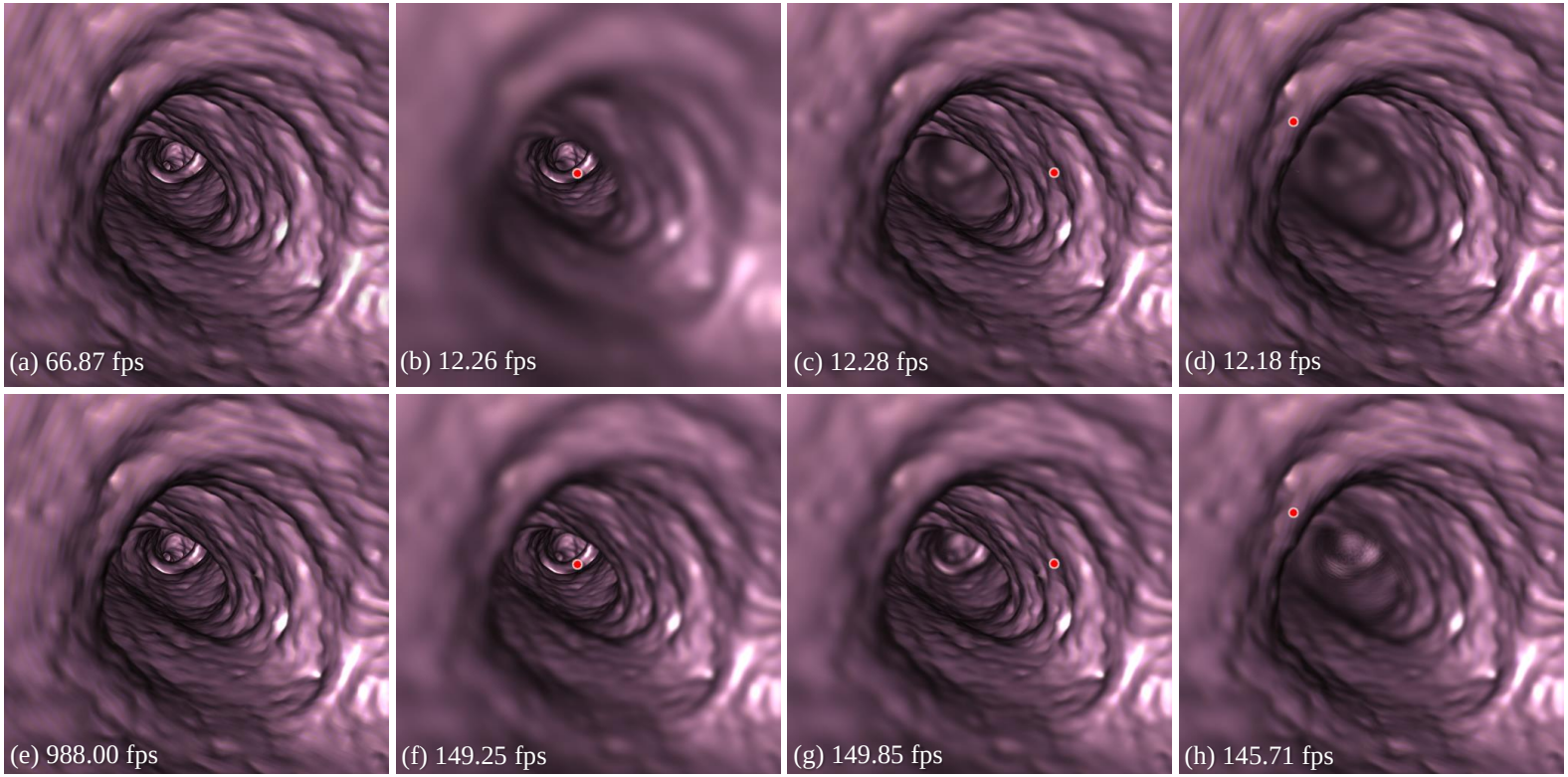


Figure 5.8 Renderings of bronchus CT data set ($512 \times 512 \times 768$ voxels). (a) Slice-based volume rendering with pinhole camera. (b), (c), and (d) DoF Slice-based volume rendering [13] with focal distances z_f of 1.1, 0.7, and 0.4, respectively. (e) Volume ray casting with pinhole camera. (f), (g), and (h) Progressive DoF volume ray casting. The red dot indicates the point in focus.

depth perception about 7 to 12 times faster the DoF slice-based volume rendering [13]. In Figure 5.8(h), aliasing appears in the region near the end of the left primary bronchus, which is caused by the lack of samples at a distance farther than the focal distance. In this case, adjusting the ray sampling interval may not produce a significant effect on the image quality. This is because, as mentioned in Section 3.4, the algorithm is designed to focus on the rapid change of the CoC size in front of the focal plane. We can solve this problem by increasing the number of lens samples used for the 1st render pass, but it slows down the rendering speed and requires a higher performance GPU. Alternatively, we can increase the number of render passes and add a criterion for determining the final render pass. a new criterion to be add is to consider estimating the maximum CoC size using the back faces of the VBB. In other words, at the point where the chief ray leaves the volume, the CoC size is calculated once more and becomes one of the conditions that determines the final render pass. The issue of extending our multi-pass rendering in this way remains a further work.

In summary, DoF volume ray casting using lens sampling enables high-quality and fast DoF rendering of volumetric data on modern GPUs. It is similar to the offline rendering technique in quality and faster than the existing method in speed. in experiments on various data, we have confirmed that our method can be used particularly well in medical images where depth perception is difficult due to a series of similar shapes. Therefore, it can be widely used as one of the DVR techniques for enhancing depth perception in medical image visualization.

CHAPTER 6

CONCLUSION

This paper presented a novel DoF rendering method for DVR. The thin lens model for realistic DoF effects is integrated into GPU-based volume ray casting to provide accurate perspective projection. The proposed DoF rendering method does not require preprocessing or reconstruction of the basic ray casting algorithm. It extends the volume ray casting pipeline by adding processes that simulate rays passing through the thin lens. One of the added process is to create the focal plane based on the focal distance. The focal plane can be obtained quickly and easily using GPU rasterization, such as when calculating the chief ray. Lens samples are generated evenly distributed over the lens before performing the algorithm. In the ray construction step, a process of calculating sample rays using lens samples is added. Each pixel generates sample rays starting from the lens samples and passing through the focal point of the chief ray. In the ray casting step, all sample rays individually compute the volume-rendering integral. When ray casting is finished, the final pixel color is determined by averaging the results of all sample rays. As described, our method preserves the algorithm structure of volume ray casting. And there are no restrictions on using the known acceleration techniques of volume ray casting. Hence, any application based on ray casting can use our DoF rendering method.

Direct volume rendering is essentially a computationally intensive rendering

method. The proposed DoF volume rendering method calculates the volume-rendering integral as a multiple of the underlying algorithm operation according to the number of lens samples. Because of this, users may not be able to get the rendering results interactively. Therefore, we have also described how to accelerate time-consuming DoF rendering in real time. Various accelerations are possible in the volume ray casting algorithm, where all rays are calculated independently. First, ERT, the most basic acceleration technique of volume ray casting, is applied to all sample rays. The use of ERT alone increases rendering speed considerably compared to conventional methods. However, additional acceleration is needed to ensure real-time DoF rendering even in large medical images. Thus, CoC-based multi-pass rendering technique using progressive lens sampling is proposed for further acceleration. Our acceleration technique uses a different number of lens samples per pixel by inferring the maximum CoC size of each pixel. As the chief ray's start point is closer to the camera, each pixel value is updated with more lens samples during the three render passes. This multi-pass rendering also has the effect of increasing the cache hit ratio when performing texture lookups on the GPU, providing a much faster frame rate than rendering all lens samples in a single pass.

We evaluated our DoF ray casting method for various volumetric data. As demonstrated in experimental results, the proposed method provides real-time DoF images of similar quality to ray tracing, an offline image generation technique [16]. Compared with the existing DoF volume rendering, Schott's incremental filtering [13], Our method also showed better quality and frame rate. Schott's method repeats

filtering in the slice compositing process, creating excessive blur of out of focus areas. This DoF slice-based volume rendering is difficult to accelerate because it has a structure that relies on previous blurred slices. On the other hand, our proposed method creates an optical-based realistic DoF effect, and can apply acceleration in various ways. Also, unlike Schott's method, which can focus only within the volume, the proposed method has the advantage of being able to focus anywhere. Especially in virtual endoscopic images where the camera position moves inside the volume, the DoF effects greatly helps to identify the context of complex structures at a fairly high frame rate without excessive blurring of unfocused areas. In summary, the proposed method allows natural control over the blurring of out-of-focus areas and thus it can be used to visualize medical images where the context around the region of interest is very important. In the experiments using various data, the proposed method quickly simulated the DoF effects of better quality than the existing DoF volume rendering technique [13], and showed up to 12 times higher frame rate in some cases.

In this dissertation, we predicted the maximum CoC size at the volume boundary during acceleration process. In the future work, we are planning to exploit various existing acceleration techniques such as deferred rendering and empty space skipping. These techniques can help to more accurately infer the maximum CoC size of each pixel by finding the voxels closest to the camera. And CoC-based multi-pass rendering with progressive lens sampling needs one more improvement in that each render pass uses a fixed number of lens samples. In fact, as the lens aperture size

increases, the minimum number of lens samples should increase. Therefore, it is necessary to improve our algorithm by expanding the number of render passes and applying a different minimum number of samples according to the lens aperture size. Aside from that, we are planning to implement our method on modern graphics cards that support fast ray tracing. With multi-pass rendering in the current GPU pipeline, the next render pass cannot be performed until the previous render pass is complete. Using the latest GPU with high-performance memory support, we expect to be able to asynchronously render with different numbers of lens samples per pixel to generate larger resolution DoF images in real time.

Bibliography

[1] W.E. Lorensen and H.E. Cline, “Marching cubes: A high resolution 3D surface construction algorithm,” *Comput. Graph. (ACM SIGGRAPH)*, vol. 21, no. 4, pp. 163-169, Aug. 1987.

[2] C.D. Hansen and C.R. Johnson, *The Visualization Handbook*. Burlington, MA, USA: Elsevier Inc., 2011.

[3] J. Kruger and R. Westermann, “Acceleration techniques for GPU-based volume rendering,” *VIS. 2003*, IEEE, pp. 287-292, 2003.

[4] K. Engel, M. Hadwiger, J.M. Kniss, C.R. Rezk-Salama, and D. Weiskopf, *Real-Time Volume Graphics.* A K Peters, Ltd., Wellesley, 2006.

[5] M. Meißner, H. Pfister, R. Westermann, and C.M. Wittenbrink, “Volume visualization and volume rendering techniques,” *Eurographics tutorial*, 2000.

[6] R.A. Drebin, L. Capenter, and P. Hanrahan, “Volume rendering,” *Comput. Graph. (ACM SIGGRAPH)*, vol. 22, no. 4, pp. 65-74, Jan. 1988.

[7] B. Preim, A. Baer, D. Cunningham, T. Isenberg, and T. Ropinski, “A survey of perceptually motivated 3D visualization of medical image data,” *Comput. Graph. Forum*, vol. 35, no. 3, pp. 501-525, Jun. 2016.

[8] R. Englund, and T. Ropinski, “Evaluating the perception of semi-transparent structures in direct volume rendering techniques,” in *SIGGRAPH ASIA 2016 Symposium on Visualization*, pp. 1-8, 2016.

[9] N. A. Svakhine, D. S. Ebert and W. M. Andrews, “Illustration-Inspired Depth Enhanced Volumetric Medical Visualization,” *IEEE Trans. Vis. Comput. Graphics (TVCG)*, vol. 15, no. 1, pp. 77-86, Jan.-Feb. 2009.

[10] S. Bruckner and E. Gröller, “Enhancing Depth-Perception with Flexible Volumetric Halos,” *IEEE Trans. Vis. Comput. Graphics (TVCG)*, vol. 13, no. 6, pp. 1344-1351, Nov.-Dec. 2007.

[11] P. Rheingans and D. Ebert, “Volume illustration: nonphotorealistic rendering of volume models,” *IEEE Trans. Vis. Comput. Graphics (TVCG*, vol. 7, no. 3, pp. 253-264, July-Sept. 2001.

[12] M. Burns, J. Klawe, S. Rusinkiewicz, A. Finkelstein, and D. DeCarlo, “Line drawings from volume data,” *ACM Trans. Graph. (TOG)*, vol. 24, no. 3, pp. 512–518, 2005.

[13] M. Schott, A.V.P. Grosset, T. Martin, V. Pegoraro, S.T. Smith, and C.D. Hansen, “Depth of field effects for interactive direct volume rendering,” *Comput. Graph. Forum*, vol. 30, no. 3, pp. 941-950, Jun. 2011.

[14] J. Kang, J. Lee, Y. Shin and B. Kim, “Depth-of-field rendering using progressive lens sampling in direct volume rendering,” *IEEE Access*, vol. 8, pp. 93335-93345, 2020.

[15] R. Englund and T. Ropinski, “Quantitative and qualitative analysis of the perception of semi-transparent structures in direct volume rendering,” *Comput. Graph. Forum*, vol. 37, no. 6, pp 174-187, Jan. 2018.

[16] R.L. Cook, T. Porter, and L. Carpenter, “Distributed ray tracing,” *Comput.*

Graph. (ACM SIGGRAPH), vol. 18, no. 3, pp. 137-145, Jan. 1984.

[17] M. Potmesil and I. Chakravarty, "A lens and aperture camera model for synthetic image generation," *Comput. Graph. (ACM SIGGRAPH)*, vol. 15, no.3, pp. 297-305, Aug. 1981.

[18] M. Potmesil and I. Chakravarty, "Synthetic image generation with a lens and aperture camera model," *ACM Trans. Graph. (TOG)*, vol. 1, no. 2, pp. 85-108, 1982.

[19] K. Abhari, John S. H. Baxter, E.S. Chen, A.R. Khan, C. Wedlake, T. Peters, R. Eagleson, and S. Ribaupierre, "The Role of Augmented Reality in Training the Planning of Brain Tumor Resection," in *Augmented Reality Environments for Medical Imaging and Computer-Assisted Interventions*, Springer, Berlin Heidelberg, 2013.

[20] S. Hillaire, A. Lécuyer, R. Cozot, and G. Casiez, "Depth-of-field blur effects for first-person navigation in virtual environments," *IEEE Comput. Graph. Appl.*, vol. 28, no. 6, pp. 47-55, 2008.

[21] R.T. Held, E.A. Cooper, and M.S. Banks, "Blur and disparity are complementary cues to depth," *Current biology*, vol. 22, no. 5, pp. 426-431, 2012.

[22] R.T. Held, E.A. Cooper, J.F. O'Brien, and M.S. Banks, "Using blur to affect Perceive distance and size," *ACM Trans. Graph.*, vol. 29, no. 2, pp. 19:1-16, 2010.

[23] J. Demers, "Depth of field: A survey of techniques," *GPU Gems*, chapter 23,

Addison-Wesley Longman, 2004.

[24] B.A. Barsky and T.J. Kosloff, “Algorithms for rendering depth of field effects in computer graphics,” in *Proc. ICCOMP'08*, World Scientific and Engineering Academy and Society (WSEAS), 2008.

[25] B.A. Barsky, D.R. Horn, S.A. Klein, J.A. Pang, and M. Yu, “Camera models and optical systems used in computer graphics: part I, object-based techniques,” in *International conference on computational science and its applications*, Springer, Berlin, Heidelberg, 2003.

[26] B.A. Barsky, D.R. Horn, S.A. Klein, J.A. Pang, and M. Yu, “Camera models and optical systems used in computer graphics: part II, image-based techniques,” in *International conference on computational science and its applications*, Springer, Berlin, Heidelberg, 2003.

[27] M. Pharr, W. Jakob, and G. Humphreys, *Physically Based Rendering: From Theory to Implementation*. Morgan Kaufmann, 2016.

[28] *Intel OSPRay: The Open, Scalable, and Portable Ray Tracing Engine*. Accessed: Aug. 14, 2020. [Online]. Available: https://www.ospray.org

[29] P. Haeberli and K. Akeley, “The accumulation buffer: hardware support for high-quality rendering,” *Comput. Graph. (ACM SIGGRAPH)*, vol. 24, no. 4, pp. 309-318, Sep. 1990.

[30] E. Angel and D. Shreiner, *Interactive computer graphics: a top-down approach with shader-based OpenGL*. Boston, Addison-Wesely, 2012.

[31] Y. Jeong, K. Kim, and Sungkil Lee, “Real-time defocus rendering with level

of detail and sub-sample blur," *Comput. Graph. Forum*, vol. 32, no. 6, pp. 126-134, Mar. 2013.

[32] X. Liu and J.G. Rokne, "Depth of field synthesis from sparse views," *Computers & Graphics*, vol. 55, pp. 21-32, 2016.

[33] J. Křivánek, J. Zara, and K. Bouatouch, "Fast depth of field rendering with surface splatting," in *Proc. CGI 2003*, IEEE, pp. 196-201, 2003.

[34] M. Zwicker, H. Pfister, J. Van Baar, and M. Gross, "Surface splatting," in *Proc. SIGGRAPH 2001*, pp. 371-378, 2001.

[35] M. Zwicker, H. Pfister, J. Van Baar, and M. Gross, "EWA splatting," *IEEE Trans. Vis. Comput. Graphics (TVCG)*, vol. 8, no. 3, pp.223-238, 2002.

[36] S. Lee, G.J. Kim, and S. Choi, "Real-time depth-of-field rendering using point splatting on per-pixel layers," *Comput. Graph. Forum*, vol. 27, no. 7, pp. 1955-1962, 2008.

[37] P. Rokita, "Fast generation of depth of field effects in computer graphics," Computers & Graphics, vol. 17, no.5, pp. 593-595, 1993.

[38] P. Rokita, "Generating Depth-of-Field Effects in Virtual Reality Applications," *IEEE Comput. Graph. Appl.*, vol. 16, no. 2, pp. 18-21, 1996.

[39] J.D. Mulder and R. van Liere, "Fast Perception-Based Depth of Field Rendering," *Proc. ACM Symp. Virtual Reality Software and Technology (VRST '00)*, pp. 129-133, 2000.

[40] P.J. Burt, "Fast filter transforms for image processing," *Computer Graphics, Image Processing*, vol. 16, no. 1, pp. 20-51, 1981.

[41] P.J. Burt and E.H. Adelson, “The Laplacian pyramid as a compact image code,” *IEEE Trans. Commun.*, vol. 31, no. 4, pp. 532-540, 1983.

[42] G. Riguer, N. Tatarchuk, and J. Isidoro, “Real-Time Depth of Field Simulation,” *ShaderX2: Shader Programming Tips and Tricks with DirectX 9*, W.F. Engel, ed., chapter 4, pp. 529-556, Wordware, 2003.

[43] T. Scheuermann, “Advanced Depth of Field,” Proc. Game Developers Conf. (GDC), 2004.

[44] J. Hammon, “Practical Post-Process Depth of Field,” *GPU Gems 3*, H. Nguyen, ed, chapter 28, pp. 583-606, Addison-Wesley, 2007.

[45] T. Zhou, J.X. Chen, and M. Pullen, “Accurate depth of field simulation in real time,” *Comput. Graph. Forum*, vol. 26, no. 1, pp. 15-23, Mar. 2007.

[46] M. Bertalmío, P. Fort, and D. Sanchez-Crespo. “Real-time, accurate depth of field using anisotropic diffusion and programmable graphics cards,” in *Proc. 3DPVT 2004*, pp. 767-774, 2004.

[47] S. Lee, G.J. Kim, and S. Choi, “Real-time depth-of-field rendering using anisotropically filtered mipmap interpolation,” *IEEE Trans. Vis. Comput. Graphics (TVCG)*, vol. 15, no. 3, pp. 453-464, May. 2009.

[48] C. Scofield. “$2\frac{1}{2}$ Depth of field simulation for computer animation,” *Graphics Gems III (IBM Version)*, pp. 36-38, Morgan Kaufmann, 1992.

[49] B.A. Barsky, “Vision-realistic rendering: simulation of the scanned foveal image from wavefront data of human subjects.” in *Proc. APGV'04*, pp. 73-81, 2004.

[50] M. Kass, A. Lefohn, and J. Owens, “Interactive depth of field using simulated diffusion on a GPU,” Tech. rep., Pixar Animation Studios, 2006.

[51] M. Kraus and M. Strengert, “Depth-of-field rendering by pyramidal image processing,” *Comput. Graph. Forum*, vol. 26, no. 3, pp. 645-654, Sep. 2007.

[52] S. Lee, E. Eisemann, and H. Seidel, “Depth-of-field rendering with multiview synthesis,” *ACM Trans. Graph.*, vol. 28, no. 5, pp. 134:1-6, Dec. 2009.

[53] T. Ropinski, F. Steinicke, and K. Hinrichs, “Visually supporting depth perception in angiography imaging,” *Smart Graphics*, Springer, Berlin Heidelberg, pp. 93-104, 2006.

[54] A. Sharifi and P. Boulanger, “Using stochastic sampling to create depth-of-field effect in real-time direct volume rendering.” in *Proc. Graphics Interface 2014*, pp. 77-85, 2014.

[55] C. Ware, *Information Visualization: Perception for Design, 3rd edition*. Waltham, USA, Elsevier, 2013.

[56] A.V.P. Grosset, M. Schott, G. Bonneau and C. D. Hansen, “Evaluation of depth of field for depth perception in DVR,” *PacificVis*, IEEE, pp. 81-88, 2013.

[57] P. Grosset, *Investigating Depth of Field in Volume Rendering and Distributed Volume Rendering on High Performance Computing Systems*. Doctoral dissertation, Department of School of Computing, University of Utah, 2016.

[58] S. Marschner and P. Shirley, *Fundamentals of Computer Graphics, 4th edition*. New York, USA: A K Peters/CRC Press, 2015.

[59] K. Hata and S. Savarese, *Course Notes 1: Camera Models*. Advanced Topics in Signal Processing: 3D Image Processing and Computer Vision, 24 Sep. 2018, University of California, Berkeley, Lecture.

[60] K. Suffern, *Ray Tracing from the Ground Up*. Wellesley, USA: A K Peters, Ltd., 2016.

[61] N. Max, "Optical models for direct volume rendering," *IEEE Trans. Vis. Comput. Graphics (TVCG)*, vol. 1, no. 2, pp. 99-108, 1995.

[62] K. Engel and T. Ertl, "Interactive high-quality volume rendering with flexible consumer graphics hardware," *Eurographics (STARs)*, 2002.

[63] E. LaMar, B. Hamann, and K.I. Joy, "Multiresolution techniques for interactive texture-based volume visualization," *VIS. '99*, IEEE, pp. 355-361, 1999.

[64] H. Scharsach, M. Hadwiger, A. Neubauer, S. Wolfsberger, and K. Bühler, "Perspective isosurface and direct volume rendering for virtual endoscopy applications," in *Eurovis'06*, vol. 6, pp. 315-322, 2006.

[65] M. Meißner, D. Bartz, "Translucent and opaque direct volume rendering for virtual endoscopy applications," in *Proc. Volume Graphics 2001*, Springer, pp.375-384, 2001.

[66] P. Christensen, A. Kensler, and C. Kilpatrick, "Progressive multi-jittered sample sequences," *Comput. Graph. Forum*, vol. 37, no. 4, pp. 21-33, Jul.

2018.

[67] D. Mitchell, “Spectrally optimal sampling for distribution ray tracing,” *Comput. Graph. ACM SIGGRAPH*, vol. 25, no. 4, pp. 157-164, 1991.

[68] J. Halton, “Algorithm 247: Radical-inverse quasi-random point sequence,” *Communications of the ACM*, vol. 7, no. 12, pp. 701-702, 1964.

[69] I. Sobol’, “On the distribution of points in a cube and the approximate evaluation of integrals,” *USSR Computational Mathematics and Mathematical Physics*, vol. 7, no. 4, pp. 86-112, 1967.

[70] A. Owen, “Monte Carlo variance of scrambled net quadrature,” *SIAM Journal on Numerical Analysis*, vol. 34, no. 5, pp. 1884-1910, 1997.

[71] A. Owen, “Quasi-Monte Carlo sampling,” in *SIGGRAPH Monte Carlo Ray Tracing Course Notes*. ACM, 2003.

[72] P. Christensen, “Progressive sampling strategies for disk light sources,” *Pixar Animation Studios*, Tech. rep., 18-02, Sep. 2018.

[73] J. Matoušek, “On thel2-discrepancy for anchored boxes,” *Journal of Complexity*, vol. 14, no. 4, pp. 527-556, Dec. 1998.

[74] M. Hadwiger, P. Ljung, C. R. Salama, and T. Ropinski, “Advanced illumination techniques for gpu-based volume raycasting,” in *Proc. ACM SIGGRAPH Courses*, SIGGRAPH, 2009.

[75] I. Wald, G.P. Johnson, J. Amstutz, C. Brownlee, A. Knoll, J. Jeffers, J. Gunther, and P. Navratil, “OSPRay-a cpu ray tracing framework for scientific visualization,” *IEEE Trans. Vis. Comput. Gr.*, vol. 23, no. 1, pp.

931-940, Jan. 2017.

초 록

직접 볼륨 렌더링(direct volume rendering, DVR)은 측정 또는 수치 시뮬레이션으로 얻은 3차원 공간의 스칼라 필드(3D scalar fields) 데이터에서 정보를 추출하는데 널리 사용되는 기술이다. 볼륨 내부의 구조를 가시화하기 위해 복셀(voxel)의 스칼라 값은 종종 반투명의 색상으로 표현된다. 이러한 직접 볼륨 렌더링의 반투명성은 중첩된 구조 간 깊이 인식을 어렵게 한다. 깊이 인식을 향상시키기 위한 다양한 볼륨 렌더링 기법들은 주로 삽화풍 렌더링(illustrative rendering)을 기반으로 하며, 피사계 심도(depth of field, DoF) 효과와 같은 물리 기반 렌더링(physically based rendering) 기법들은 계산 시간이 오래 걸리기 때문에 적용이 어렵다. 가상 및 증강 현실과 같은 몰입형 시스템의 발전과 인간의 지각에 기반한 의료영상 시각화에 대한 관심이 증가함에 따라 직접 볼륨 렌더링에서 피사계 심도를 구현할 필요가 있다.

본 논문에서는 직접 볼륨 렌더링의 깊이 인식을 향상시키기 위해 볼륨 광선투사법에 피사계 심도 효과를 적용하는 새로운 방법을 제안한다. 픽셀 당 여러 개의 광선을 사용한 광선투사법(ray casting)을 수행하여 초점이 맞는 거리에 있는 물체는 선명하게 표현되고 초점이 맞지 않는 거리에 있는 물체는 흐리게 표현된다. 이러한 효과를 얻기 위하여 렌즈의 서로 다른 부분을 통과하는 광선들을 시뮬레이션 하는 얇은 렌즈 카메라 모델(thin lens camera model)이 사용되었다. 그리고 성능에 직접적으로 영향을 끼치는 렌즈 샘플은 최적의 렌즈 샘플링 방법을 사용하여 최소한의 개수를 가지고 앨리어싱(aliasing)이 없는 이미지를 생성하였다. 제안한 방법은 기존의 GPU 기반 볼륨 광선투사법

파이프라인 내에서 전처리 없이 구현된다. 따라서 볼륨 광선투사법의 모든 가속화 기법을 제한없이 적용할 수 있다.

또한 가속 기술로 누진 렌즈 샘플링(progressive lens sampling)을 사용하는 다중 패스 렌더링(multi-pass rendering)을 제안한다. 더 많은 렌즈 샘플들이 여러 렌더 패스들을 거치면서 점진적으로 사용된다. 각 픽셀은 착란원(circle of confusion)을 기반으로 예측된 최대 흐림 정도에 따라 다른 최종 렌더링 패스를 갖는다. 이 기법은 거리에 따른 피사계 심도 효과의 흐림 정도에 따라 각 픽셀에 다른 개수의 렌즈 샘플을 적용할 수 있게 한다. 이러한 가속화 방법은 불필요한 렌즈 샘플링을 줄이고 GPU의 캐시(cache) 적중률을 높여 직접 볼륨 렌더링에서 상호작용이 가능한 프레임 속도로 피사계 심도 효과를 렌더링 할 수 있게 한다.

다양한 데이터를 사용한 실험에서 제안한 방법은 실시간으로 사실적인 피사계 심도 효과를 생성했다. 이러한 결과는 우리의 방법이 오프라인 이미지 합성 방법과 유사한 품질의 피사계 심도 효과를 생성하면서 직접 볼륨 렌더링의 기존 피사계 심도 렌더링 방법보다 최대 12배까지 빠르다는 것을 보여준다.

주요어 : 직접 볼륨 렌더링, 광선투사법, 볼륨 가시화, 피사계 심도 효과, 컴퓨터 그래픽스, 이미지 생성

학 번 : 2013-20736