공학박사 학위논문

Slab-based Intermixing for Multi-Object Rendering of Heterogeneous Datasets

다중 이종 객체들의 렌더링을 위한 슬랩 기반 혼합 기법 연구

2016 년 8 월

서울대학교 대학원

전기.컴퓨터공학부

김 동 준

Abstract

The visualization of multiple 3D objects has been increasingly required for recent applications in many industrial, biomedical, and scientific fields. Due to the heterogeneity in data representation or data configuration, it is difficult to efficiently render multiple objects in high-level performance. In this dissertation, we propose a multi-object rendering method based on a novel intermixing model for high performance visualization of multiple volumetric and polygonal objects. To represent multiple layers of polygonal surfaces and volumetric fuzzy surfaces, we introduce zSlab model which is defined with visibility color, slab depth and finite thickness per pixel. zSlab is based on radiosity spreading out the viewing direction, which is a special ray-segment used for an intermixing unit.

As the proposed zSlab model is designed to treat multiple polygonal and volumetric geometries which may be unordered, we apply the order-independent-transparency (OIT) concept to the construction of zSlabs. With the proposed zSlab-based intermixing method, this enables image-level intermixing for a variety of object combinations; traditional polygon rendering for many transparent surfaces, hybrid rendering for polygon and volume, and multi-volume rendering.

First, we present how the zSlab is applied to volumetric fuzzy surface as well as infinitely thin surface of polygonal geometry with the proposed virtual zSlab concept. And, we introduce a novel z-thickness buffer that stores the zSlabs as an array, which is used for the rendering input and output. We introduce two versions

of in-slab visibility interpolation methods and verify which model is well suit for the proposed intermixing algorithm. Finally, based on zSlab model and the in-slab visibility interpolation, we propose an efficient slab-based visibility intermixing algorithm so that the entire intermixing leads to a high performance rendering result (acceptable image result and fast rendering speed).

Experimental results demonstrate that the proposed method delivers more effective multi-object rendering in terms of taking advantages of the image-level intermixing especially for a rendering scene that includes at least one volumetric object, providing acceptable image quality compared to the image based on the conventional on-the-fly intermixing. And the proposed intermixing method is able to resolve traditional intermixing artifacts such as aliasing intersection and z-fighting plane occurred in intersecting or overlapping surface region. Moreover, our experimental results manifest a potential of the proposed method that it can deliver a substantial aid in scientific visualization of entire context including hidden (or, inside) structures in the multi-object scene.

Keyword: multi-object rendering, zSlab, z-thickness buffer, order-independent transparency, visibility interpolation, image-level intermixing

Student Number: 2008-30868

Table of Contents

Chapter 1. Introduction 1

1.1. Motivation 1

1.2. Backgrounds 2

1.2.1. Rasterization graphics 4

1.2.2. Volume ray-casting 5

1.2.3. Visibility function 6

1.3. Technical issues of multi-object rendering 8

1.3.1. Multi-volume rendering 9

1.3.2. Order-independent transparency 11

1.3.3. Hybrid rendering of polygon and volume 12

1.4. Main contribution 13

1.5. Organization of the dissertation 15

Chapter 2. zSlab and z-thickness buffer 16

2.1. Concept of zSlab and z-thickness buffer 16

2.2. Construction of zSlab array 18

2.2.1. zSlab compression 20

2.2.2. Virtual zSlab 22

Chapter 3. Visibility Interpolation 26

3.1. zSlab subdivision 27

3.2. In-slab visibility interpolation 27

3.3. Modified version of visibility interpolation 29

Chapter 4. Slab-based Visibility Intermixing 34

4.1. Intermixing of different zSlab arrays 35

4.2. Intermixing complexity 37

4.3. Merging operation over the overlapping zSlabs 38

Chapter 5. Implementation 40

5.1. Implementation of zSlab 41

5.2. Overall intermixing pipeline 41

5.2.1. Buffers 42

5.2.2. Polygon rendering stage 43

5.2.3. zSlab intermixing stage 44

5.2.4. Volume rendering stage 44

5.3. Intermixing kernel for zSlab merging 45

5.4. Thickness determination of virtual zSlab 47

5.5. Optimization advantages 49

Chapter 6. Case Studies and Results 51

6.1. Experiment condition 51

6.2. Smooth visibility transition study 52

6.2.1. High-zoom and high-opacity of MDVR 52

6.2.2. Opaque surface rendering of stitching volumes 55

6.2.3. Z-fighting artifact on overlapping surfaces 59

6.3. Transparent MDVR study 62

6.4. Visibility interpolation study: original vs. modified 67

6.5. Fusion rendering study for the hidden structures 69

6.5.1. Implicit attribute volume visualization 69

6.5.2. zSlab blending for visual-cues of hidden structures 7 1
6.6. Evaluation of the intermixing speed 7 6
6.6.1. The overhead of the intermixing operations 7 6
6.6.2. The overhead according to the number of zSlab layers 8 1
6.7. Evaluation of memory usage 8 4

Chapter 7. Conclusion..... 8 5

Bibliography 8 8

초 록 9 7

List of Tables

Table 6.1. Volume size and rendering type information used in the experimental results. 8 3

List of Figures

Figure 1.1. Rasterization graphics depicting a polygonal mesh to a framebuffer... 4

Figure 1.2. Four steps of the basic form of the volume ray-casting algorithm..... 5

Figure 1.3. Merging pipelines during the volume ray-casting..... 9

Figure 1.4. MDVR issues caused by different resolution, different geometry, or different modality..... 10

Figure 1.5. Semi-transparent polygonal surfaces along three pixel rays..... 11

Figure 1.6. Workflow of the image-level intermixing 14

Figure 2.1. Thickness boundary model represented in volumetric object..... 16

Figure 2.2. Example showing two different zSlab arrays of which each is stored in a z-thickness buffer that has 8 depth layers 17

Figure 2.3. Illustration of zSlab construction for polygonal surface and volumetric surface geometries along a ray that starts at the camera origin and passes through a pixel in an image plane, accompanied by visibility function on each pixel 19

Figure 2.4. Illustration depicting the zSlab's workflow 20

Figure 2.5. Example of zSlab array generated from a ray-casting..... 22

Figure 2.6. Illustration showing an intersection between two overlapping polygonal-objects; a blue cube and a red sphere..... 25

Figure 3.1. Illustration of the opacity behavior of original zSlab and sub-zSlab divided by an intersecting boundary, according to opaqueness and in-slab visibility interpolation model..... 26

Figure 3.2. An intermixing example based on different in-slab visibility interpolation models; the upper based on the original version of in-slab visibility interpolation and the lower based on the modified version of in-slab visibility interpolation..... 30

Figure 3.3. Example showing the intermixing results including a volumetric object with a transparent thin layer of outer surface..... 32

Figure 4.1. Illustration of the intermixing process of two different zSlab arrays,

which is locally performed over the overlapping region 34
Figure 4.2. Thee intermixing cases of two different zSlab arrays 35
Figure 5.1. The intermixing pipeline that treats the proposed zSlab model and intermixing operations 40
Figure 5.2. Diagram for the intermixing kernel function for each overlapping zSlab 46
Figure 5.3. Three examples of two overlapping polygonal-objects (a blue sphere and a red cube) that depict the virtual zSlab effect according to the virtual thickness 48
Figure 5.4. Comparison between the conventional intermixing pipeline and the proposed intermixing pipeline 49
Figure 6.1. Comparison of high zoom-in rendering results of two high-opacity volumetric geometries 53
Figure 6.2. Comparison of rendering results of two dental CT scans of each volume is rendered through high-opacity transfer function 54
Figure 6.3. Comparison of iso-surface rendering results of two stitching cases 56
Figure 6.4. Comparison of rendering results of transparent surfaces overlapping each other 58
Figure 6.5. Comparison of rendering results of planar objects sharing a same depth; measurement widgets on the superimposed plane or clipping plane of the main volumetric object 59
Figure 6.6. Comparison of rendering results visualizing the clipping planes and the artificial objects 61
Figure 6.7. Comparison of rendering results of two transparent volumes composed of multiple thin surfaces according to the number of zSlab layers 62
Figure 6.8. Comparison of transparent multi-volume rendering results; scientific, stitching, and multi-modal visualization 66
Figure 6.9. Comparison of rendering results of intersecting surfaces along with a clipping plane according to in-slab visibility interpolation models 67
Figure 6.10. Comparison of rendering results according to in-slab visibility interpolation models 68

Figure 6.11. An example of the intermixing scene that needs different volume rendering techniques 71
Figure 6.12. Comparison of rendering results for the visualization of the hidden structures according to the representation method of the hidden objects..... 73
Figure 6.13 An intermixing example of the hidden or intra structures inside a volume 74
Figure 6.14. Comparison of rendering results of a dental CT and several polygonal objects with a variety of transparency and virtual thickness on the polygon surfaces 75
Figure 6.15 A chart depicting the rendering speed of the experimental cases, comparing between the individual objects rendering and the hybrid-level intermixing 77
Figure 6.16 Rendering results of the transparent volume embedding superimposed planes, according to the number of the superimposed planes..... 78
Figure 6.17 A plot showing the rendering time of Fig.6.16 (special version of hybrid-level intermixing) to measure the computational time of the proposed slab-based intermixing kernel function 79
Figure 6.18 Plots for the speed of the transparent multi-volume rendering according to the number of zSlab layers (the full image-level intermixing), comparing to the conventional MDVR (without ESS). 81

Chapter 1. Introduction

1.1. Motivation

The visualization of different types of multiple 3D objects has found its increasing usage in many biomedical, industrial, and scientific fields [1]. With the advance of the scanning device in the biomedical field, it is becoming one of common practice to simultaneously examine multi-modality medical datasets such as the interior of the human body obtained by CT (Computed Tomography)-PET (Positron Emission Tomography), MR (Magnetic Resonance)-CT, and MR-SPECT [1, 2, 3, 4]. Moreover, the application for the simultaneous evaluation is also necessary in the scientific datasets such as computational fluid dynamics (CFD), geological and seismic data, as well as abstract mathematical data such as 3D probability distributions of pseudo random numbers. Those scientific and medical applications also require user interaction 3D widgets or artificial models (e.g., 3D UI for simulation and surgery tools usually represented in polygonal models or prior-segmented parts in voxels, attributed volume [5]) to be rendered along with different types of 3D objects [6, 7, 8, 9]. Moreover, for the hybrid inspecting and modeling applications of CT-scanned volume and laser-scanned mesh or CAD models [41, 42], the simultaneous rendering of volumes and polygons is often required. With the evolution of e cient volume rendering techniques, volumetric data is becoming more and more important also for visual arts and computer games

which need to combine the different representation data such as polygonal object. In all of those multi-object rendering applications, the faster and the more accurate (i.e., rendering in a correct z-depth order and representing all of visible structures) is the rendering, the higher is the user satisfaction.

Those 3D objects often come in various representations such as multi-modality, volumetric and polygonal, scan and attribute (often derived from a scan data) datasets [23]. This heterogeneity in the data representation makes it difficult to render the multiple objects in a single rendering pipeline as each type of dataset requires a different visualization technique on its own. Even with multiple homogeneous datasets, the simultaneous rendering is not that simple when there is a difference in data configuration (e.g. acquisition precision, coordinate system, etc.) between them [1, 10, 11, 12]. And more objects with such difference participate in the 3D scene, more complicated the rendering optimization is.

Those reasons are why it is difficult and complex to construct the unified-rendering framework of such heterogeneous 3D objects [13, 14, 28]. To the best of our knowledge, there has been no image-level intermixing approach that is able to well represent such a complicated objects scene and resolves the follow-up technical issues in the multi-object rendering.

1.2. Backgrounds

3D objects may be created using surface representations or volumetric representations. A typical representation type of the surface is polygonal meshes.

Since light transport is evaluated only at points on the polygonal surface, the polygon rendering is generally achieved by GPU-built-in rasterization graphics [53] rather than polygon ray-tracing algorithm [54] due to its real-time performance. The method does not account for light interaction which is taking place in the atmosphere or in the interior of an object.

Voxel is a typical representation unit of the volumetric object, which is defined in sample field. Contrary to polygonal surfaces, sampling and optical transfer function define the surface during the rendering process. Volume rendering [15] describes a wide range of techniques for generating images from the three-dimensional sample filed data. These techniques are originally motivated by scientific visualization, where volume data is acquired by intra-structure scanning, measurement or numerical simulation of natural phenomena. As volumetric data are more di cult to visualize than surfaces, data conversion from the volumetric data into surfaces or 2D subsets is often used, referred as indirect volume rendering. Since light transport is evaluated over the entire target sample field, volume data is ideal to describe fuzzy objects, such as fluids, gases and natural phenomena like clouds, fog, and fire [55, 56]. Thus, it is both worthwhile and rewarding to render them as truly three-dimensional entities without falling back to 2D subsets, referred as direct volume rendering (DVR). Generally, this is achieved by volume ray-casting algorithm.

1.2.1. Rasterization graphics

In order to visualize the polygonal data, the rasterizer of the graphics rendering pipeline takes each polygon as an input primitive and perform raster-scan to produce a set of grid-aligned fragments enclosed within the primitive. A fragment is 3-dimensional, with a (x, y, z) position. The (x, y) are aligned with the 2D pixel-grid. The z-value (along a viewing direction in an image space) denotes its depth. The z-values are needed to capture the relative depth of various primitives, so that the occluded objects can be discarded or alpha-blended according to the sorting order of the transparent layers. This process is optimized to visualize the foremost surface of the polygon object. For the representation of multiple transparent surface-layers, the raster-scan-based rendering needs to be multi-pass processing, storing the rendering result of each rendering pass.

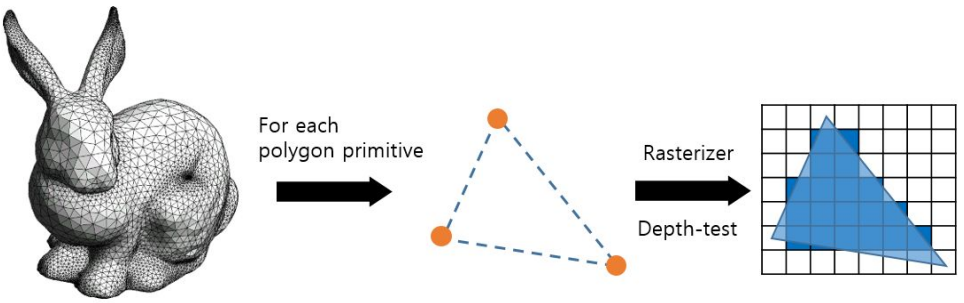


Figure 1.1. Rasterization graphics depicting a polygonal mesh to a framebuffer. Fragments are produced through interpolation of the vertices of a primitive. Thereby, a fragment has all the vertex's rendering attributes such as color, normal vector, and texture coordinates.

1.2.2. Volume ray-casting

Ray casting is a typical graphics rendering algorithm that uses the geometric algorithm of ray tracing. Ray tracing-based rendering algorithms operate in image order to render three-dimensional scenes to two-dimensional images. Contrary to the ray-casting in the sense used in polygonal ray-tracing which processes polygonal surfaces, volume ray-casting passes through the sample field, while sampling values and shading the illumination along a ray without ray-interaction (e.g., ray reflection, secondary ray simulation).

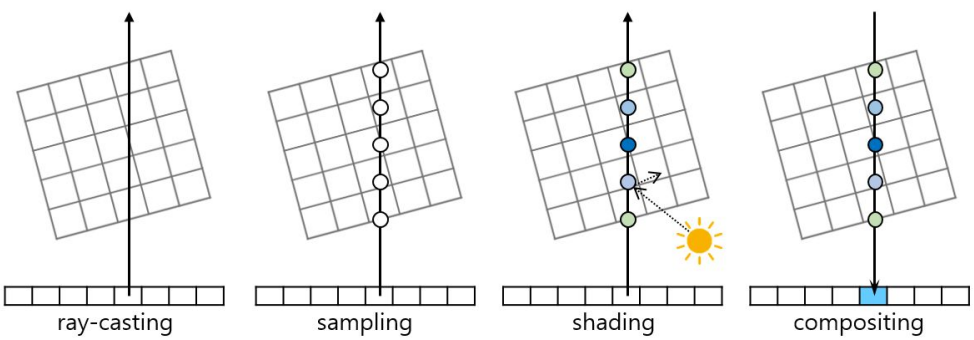


Figure 1.2. Four steps of the basic form of the volume ray-casting algorithm.

Fig.1.2 illustrates the basic algorithm of the volume ray-casting. In the first step of volume ray-casting, a ray along a pixel on the image plane is shot (or cast) through the voxels sample-field. At this step, it is useful to consider the first hit point of the bounding geometry, usually a cuboid shape, so that the unnecessary sampling can be avoided. In the next step, sampling is performed at every sampling position (normally, equidistant sampling points are selected.). In general, the

volume is not aligned with the ray and image plane, and sampling points will usually be located in between voxels. Because of that, it is necessary to interpolate the values of the samples from its surrounding voxels (commonly using trilinear interpolation). And then, for each sampling point, a gradient of illumination values is computed. This represents the orientation of local surface defined in the volume. Each sample is then shaded according to its surface orientation and the location of the light source in the scene. Finally, after all sampling points have been shaded, they are composited along the ray of each pixel, resulting in the final color value for the pixel that is currently being processed. The composition is derived directly from the rendering equation based on the blending operations considering the depth-sorted order.

1.2.3. Visibility function

A slab is a separated piece of a z-directional (or, viewing directional) ray, which stores visibility information to represent a thickness range of a boundary such as a volumetric fuzzy surface. The function of the visibility information is determined by an optical model describing the transport of light that passes along a ray. Under the simple-emission-absorption model [16], the overall fraction of light that penetrates a slab filled with a medium from a given depth t to front depth t_0, referred as a transmittance (i.e. $1 - \text{opacity}$), is given by the following formula:

$$T(t) = exp\left(\int_{t_0}^{t} \tau(z)dz\right) \tag{1.1}$$

where $\tau(t)$ is an extinction coefficient function of the medium, and the integrating direction is front to back along a viewing z-direction for the ease of explanation in this dissertation.

If the emitted radiance at a depth z is $\tilde{c}(z)$, the total amount of color radiance, C, contributing to the final visibility of the medium is determined by integrating the emitted radiance attenuated by the participating medium from the front depth to the given depth along the viewing direction:

$$C = \int_{\mathrm{t_0}}^{\mathrm{t}} \tilde{c}(z)T(\mathrm{z})dz \tag{1.2}$$

In this dissertation, a slab represents a region interval filled with an optical medium (which has homogeneous property in modified version for more continuous and smoother visibility transition), which has sampled visibility. And, thus we can integrate the color radiance across successive slabs as a Riemann sum. The final visibility (associated color [30] or alpha-multiplied color [31] and alpha) integrated from the l_{th} to m_{th} slabs, C_{sum}, can be calculated by the discretized version of eq. (1.2):

$$C_{sum} = \sum_{i=l}^{m} C_i \prod_{j=l}^{i-1} T_j = \sum_{i=l}^{m} C_i \prod_{j=l}^{i-1}(1 - A_j) \tag{1.3}$$

Eq. (1.3) can be evaluated iteratively by performing alpha blending in either

front-to-back or back-to-front order. In this dissertation, we integrate the slabs in a front-to-back order along the viewing direction (for more intuitive description and affordable to the acceleration for the early-ray-termination [32]) as follows:

$$\sum_i C = \sum_{i-1} C + (1 - \sum_{i-1} A)C_i \tag{1.4}$$

$$\sum_i A = \sum_{i-1} A + (1 - \sum_{i-1} A)A_i \tag{1.5}$$

Note that, in all blending equations and visibility descriptions in this dissertation, we use the associated color.

1.3. Technical issues of multi-object rendering

Many of previous multi-object rendering approaches targeted at polygonal or volumetric datasets only. For the polygonal dataset, lots of approaches have been reported to resolve transparency issues of many (low-alpha) polygon surfaces. Commonly, in a rasterization graphics for the polygonal data, the 3D geometry with transparency is rendered by blending all surfaces into a framebuffer. For a correct result, surfaces must be blended from farthest to nearest (back-to-front blending) or from nearest to farthest (front-to-back blending), depending on the alpha compositing operation [15, 16]. As a correct rendering result requires buffer-stored geometry in sorted order, many (close low alpha) surfaces are problematic due to bounded memory capacity and low sorting speed. Order-independent

transparency (OIT) [17, 18, 19] is a class of techniques in rasterization graphics for treating this problem [20, 21, 22].

For the volumetric datasets, instead, lots of previous methods adopted the volume ray-casting algorithm [15, 16] while intermixing the samples from all the objects at each sampling position [24, 25, 26]. This on-the-fly intermixing method exhibits accurate rendering result; however, it has some limitations when the datasets have different data configuration. Several approaches for rendering polygonal and volumetric objects simultaneously have been also presented, but they compute the intersection of polygonal and volumetric objects at each sample, are also an on-the-fly intermixing, resulting in high computational cost.

1.3.1. Multi-volume rendering

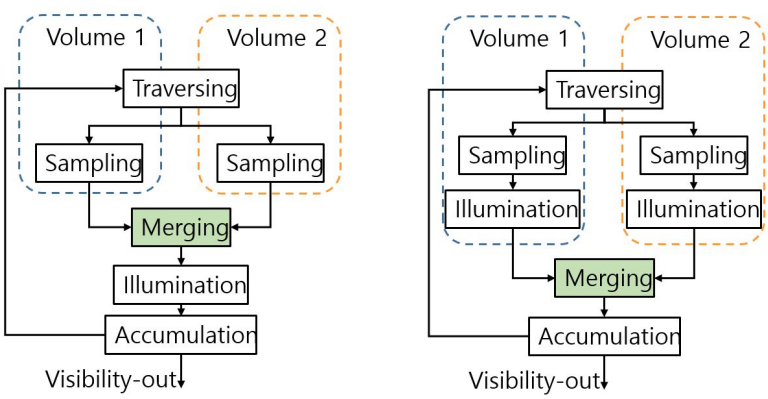


Figure 1.3. Merging pipelines during the volume ray-casting. Two approaches are possible to determine a visibility of a sampling point. The right is the merging pipeline that is used in the proposed intermixing method.

For multi-volume direct rendering (MDVR), a multi-volume ray-casting [24] was introduced, which samples all participating volumes at a single ray step with simple merging rules. With regard to specific merging rule of different volume modalities (such as CT, MRI or PET), the intermixing methods were classified according to merging operation level in the intermixing stages [25, 27]. Also different rendering styles as the merging rule [26] was suggested as an accumulation level intermixing [27] (see Fig.1.3), which is used in the proposed intermixing method. However, the approaches above only focused on how to merge and fuse different volumes along a single ray.

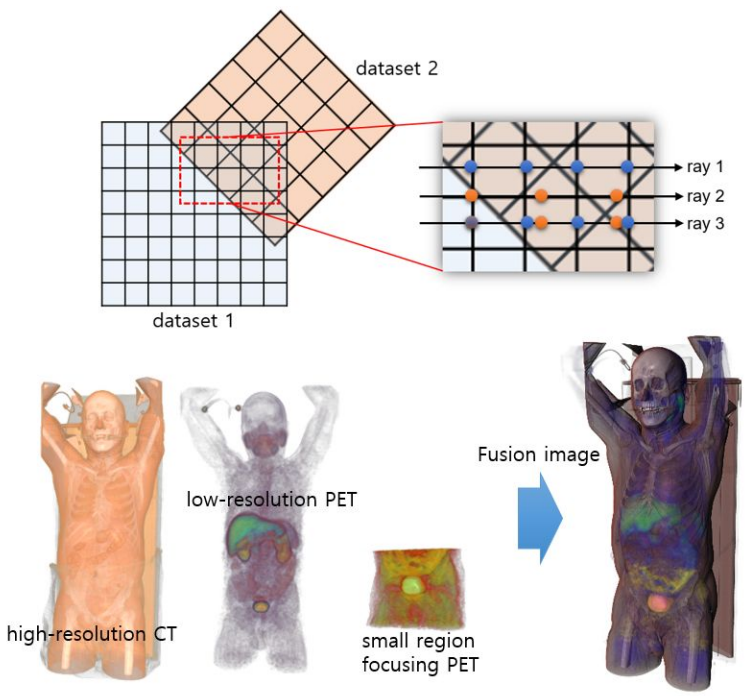


Figure 1.4. MDVR issues caused by different resolution, different geometry, or different modality. Because the sampling distance is determined by volume's resolution, the conventional MDVR brings about oversampling (ray 1 and ray 2) or misaligned sampling problem (ray 3).

Each volumetric dataset can have different resolution or different geometry. This causes computational complexity and inflexible implementation for the rendering of multiple volumes (see Fig.1.4). To resolve this problem, several specified data structures were suggested [10, 11] introduced as a hierarchical block model. BSP tree [11] is one of typical data structures to resolve this problem. However, those approaches still require a lot of high complexity if there are more than two overlapping volumes with different resolutions and orientations. In addition, because the minimum sampling distance among participating volumes is generally used, oversampling which causes performance overhead occurs.

1.3.2. Order-independent transparency

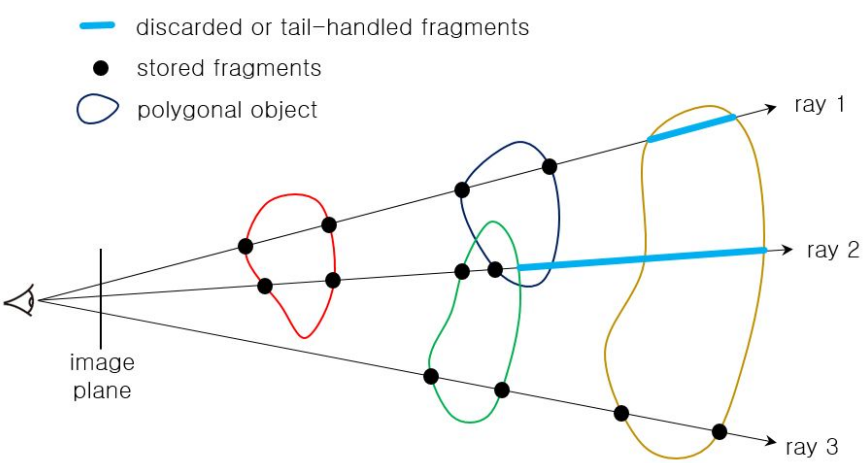


Figure 1.5. Semi-transparent polygonal surfaces along three pixel rays. Fragments stored in the buffer should be sorted in depth-order for correct blending result. Assuming four layers are available in the framebuffer, some fragments need to be discarded or tail-handled.

Polygonal data has obviously discrete surface geometry which is well suit to conventional rasterization graphics. Each surface occludes existing color and adds some of its own color depending on its alpha value, a ratio of light transmittance. Sorting polygons by depth can take a significant amount of time and the implementation is complex [20]. Instead, sorting geometry per-pixel after rasterization is generally used [20]. For exact results, this requires storing all fragments (using a special linked list [38], A-buffer [37] in graphics, data structure is generally used for the storage) before sorting and alpha compositing [17] which results in trade-off issue; accurate rendering result with unbounded size of memory and approximated rendering result with bounded sized of memory and interactive rendering speed [17, 29] (see Fig.1.5). A number of approaches have been suggested to approximate the compositing without explicit ordering, which is emphasized by its name, order-independent transparency (OIT) [18].

However, this method cannot be easily extended to volumetric datasets because it just stores the information for discrete positions and thus it is not appropriate for representing volumetric surfaces including fuzzy boundary [24].

1.3.3. Hybrid rendering of polygon and volume

Many approaches for the simultaneous rendering of polygonal and volumetric datasets, called hybrid rendering, have also been reported [7, 8, 9]. Since polygonal data and volumetric data have different rendering pipeline, almost of hybrid

renderers consist of two different rendering passes [7]; polygon rendering is first executed by using rasterization graphics with depth peeling [9, 20], and then volume rendering is executed by using the volume ray-casting algorithm while referring to the depth values obtained from the previous polygon rendering. In addition to the overhead of the z-depth test at every sampling position, these methods suffer from aliasing artifact at the intersection when a polygon and a voxel intersect with each other. Sub-voxel level computation [7] is a general way to resolve this aliasing problem, which subdivides the intersecting voxel into a set of sub-voxels and performed the z-depth test on a sub-voxel basis. This offered high-quality rendering of the intersection between the polygon and volume; however, it requires high computation. And, as complex geometry leads to the previously mentioned transparency problem [17, 20], the exact rendering that includes such complicated transparent geometry still remains unsolved problem in many applications.

1.4. Main contribution

In this dissertation, we propose a novel multi-object rendering based on deferred rendering approach (or, image-level intermixing). The proposed method is able to represent infinitely-thin-surface geometry as well as fuzzy boundaries in rendering buffers so that image-level intermixing even for the scene including volumetric object is possible with acceptable rendering accuracy.

For this, we propose a special ray-segment based on radiosity spreading out the viewing direction, called zSlab, which is able to represent multiple layers of polygonal surface geometry as well as volumetric fuzzy boundaries. We also propose an in-slab visibility interpolation and its modified version for more natural visibility transition and present a merging method for overlapping zSlabs, so that zSlab-based intermixing is available. As the proposed method is based on buffer-based approach, we apply order-independent transparency concept to the constructing zSlabs of multiple polygonal and volumetric geometries. This enables image-level intermixing for all kinds of object combinations; conventional polygon rendering for many transparent surfaces, hybrid rendering for polygon and volume, and multi-volume rendering while taking advantages of image-level intermixing (see Fig.1.6).

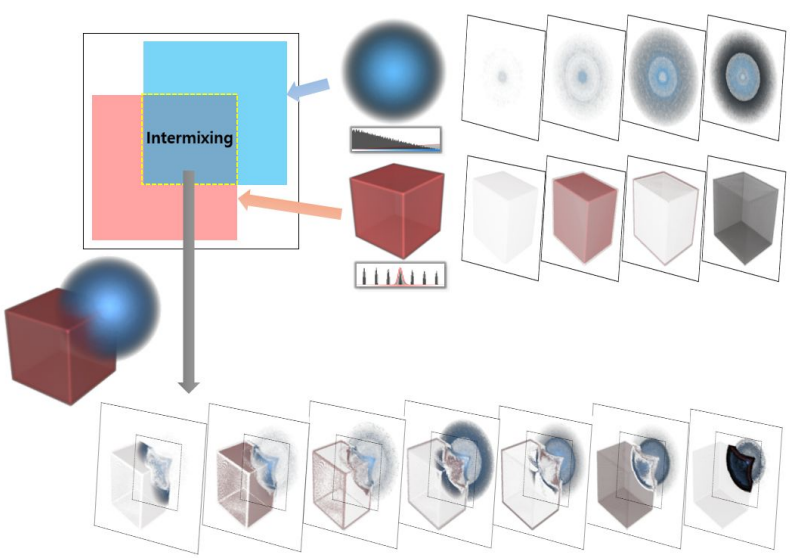


Figure 1.6. Workflow of the image-level intermixing. Each object participating in the scene is represented as multi-layer buffer storing foremost surface fragments. By using the multiple depth information of the buffer, the

intermixing is performed to generate new multi-layer buffer.

The proposed method enables faster multi-object rendering than conventional method based on the on-the-fly approach. By preserving the independency of each renderer, the proposed method can avoid complicated geometry optimization dependent on the combination of the participating objects, and can maintain the optimized rendering without additional optimization techniques. Moreover, the proposed method delivers additional advantages of representing some intermixing-object scenes; reduction of traditional mixing artifacts such as intersection aliasing and z-fighting on overlapping surfaces, and visual-cue of hidden structures.

1.5. Organization of the dissertation

The remainder of this dissertation is organized as follows. In the next Chapter, we review a visibility function and introduces the proposed zSlab and z-thickness buffer and presents how to construct the zSlab and its array. Chapter 3 describes the concept for the elaborate in-slab visibility interpolation in sub-divided zSlabs. Chapter 4 presents zSlab-based intermixing algorithm along with the proposed zSlab model. In Chapter 5, an overall resources and implementation scheme for the proposed rendering pipeline are described in detail. In Chapter 6, we present the results of the experimental case studies of the proposed method to several datasets along with intermixing speed evaluation, comparing to the conventional method. Finally, we conclude the proposed method and discuss future work in Chapter 7.

Chapter 2. zSlab and z-thickness buffer

2.1. Concept of zSlab and z-thickness buffer

zSlab is a piece of a ray that is able to represent a range of boundary, called thickness boundary [33], with specified z-depth visibility described by the visibility function. This is suitable to represent not only an infinitely thin surface geometry but also a volumetric fuzzy surface boundary. In volume rendering, partial volume effect and reconstruction sampling make this thickness boundary, which has same property as a fuzzy surface boundary (see Fig.2.1).

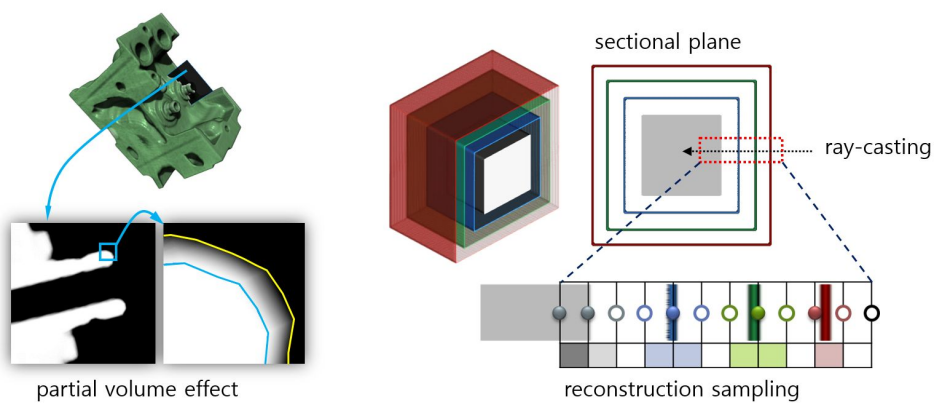


Figure 2.1. Thickness boundary model represented in volumetric object. Partial volume effect or reconstruction sampling as an intrinsic property of the sample-field-based data (e.g., volume) makes this boundary.

zSlab is a z-directional slab that covers a finite interval filled with a single homogeneous medium. This can be divided into sub-zSlabs by other intersecting geometries. The zSlab can be merged with its neighbor zSlabs into a new zSlab preserving the integrated visibility. Each zSlab has a visibility function defined by its integrated visibility and thickness at a specific depth along the slab. In order to define a zSlab, it is necessary to store z-depth of back-side of zSlab, thickness, and its integrated visibility as an association color.

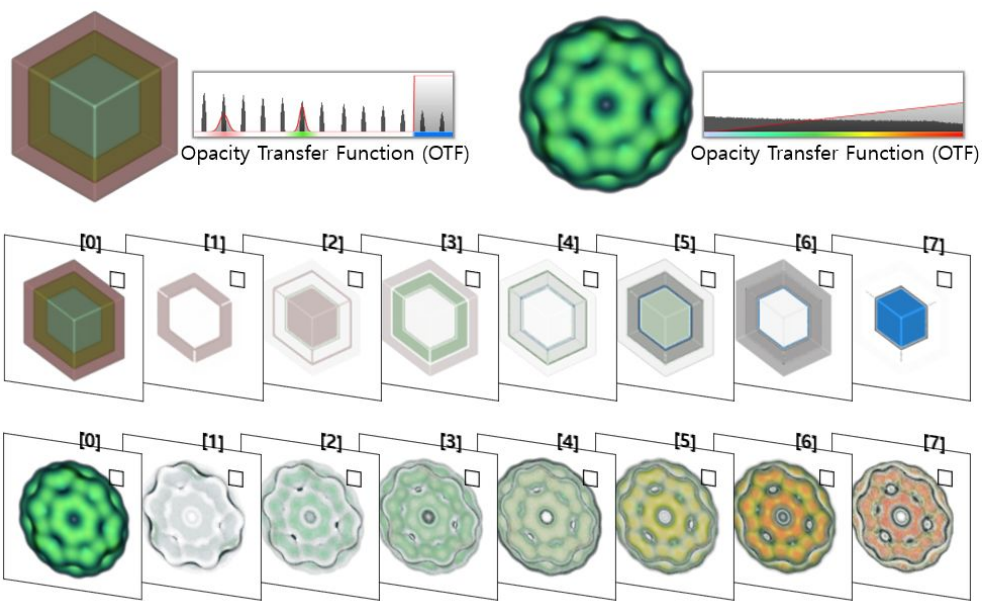


Figure 2.2. Example showing two different zSlab arrays of which each is stored in a z-thickness buffer that has 8 depth layers. 0th index layer stores the final accumulated image. The 1st to 7th layers are sorted order zSlabs which do not have self-intersection. The last zSlab represents the rest of surfaces and ray-segments as a tail-handling.

Z-thickness buffer is a framebuffer in which every pixel stores a zSlab array. Note that there is no self-intersecting zSlabs of a zSlab array stored in a z-thickness buffer. Fig.2.2 shows the layers of multiple transparent or translucent fuzzy surfaces generated from direct volume rendering (DVR), which are stored in the z-thickness buffer. We model the first layer of z-thickness buffer that stores the overall integrated zSlab, which is used as a rendering out.

2.2. Construction of zSlab array

A per-pixel ray that starts at camera origin may generate several z-directional boundaries passing through the transparent geometry boundaries. At each boundary or ray-segment, zSlab of which the opacity is not zero is defined. Rather than storing a single zSlab at each pixel, the proposed method stores multiple layers of zSlabs (normally 8 or 16) for well representing the z-directional boundaries. Fig.2.3 gives three examples, showing how visibility functions over the proposed zSlabs can account for opaque or transparent surfaces, and thickness boundaries. For an infinitely thin surface generally defined by polygonal geometry, the proposed zSlab model can optionally add a finite thickness, referred as ‘virtual zSlab’, which provides several visualization advantages in the multi-object rendering as well as fusion rendering. For a fuzzy surface generally defined by volumetric geometry, the proposed zSlab model can represent it as a thickness boundary. As complex geometry scene makes unbounded number of zSlabs and the capacity of the z-thickness buffer is bounded, zSlabs need to be compressed preserving the original visibility over the entire ray.

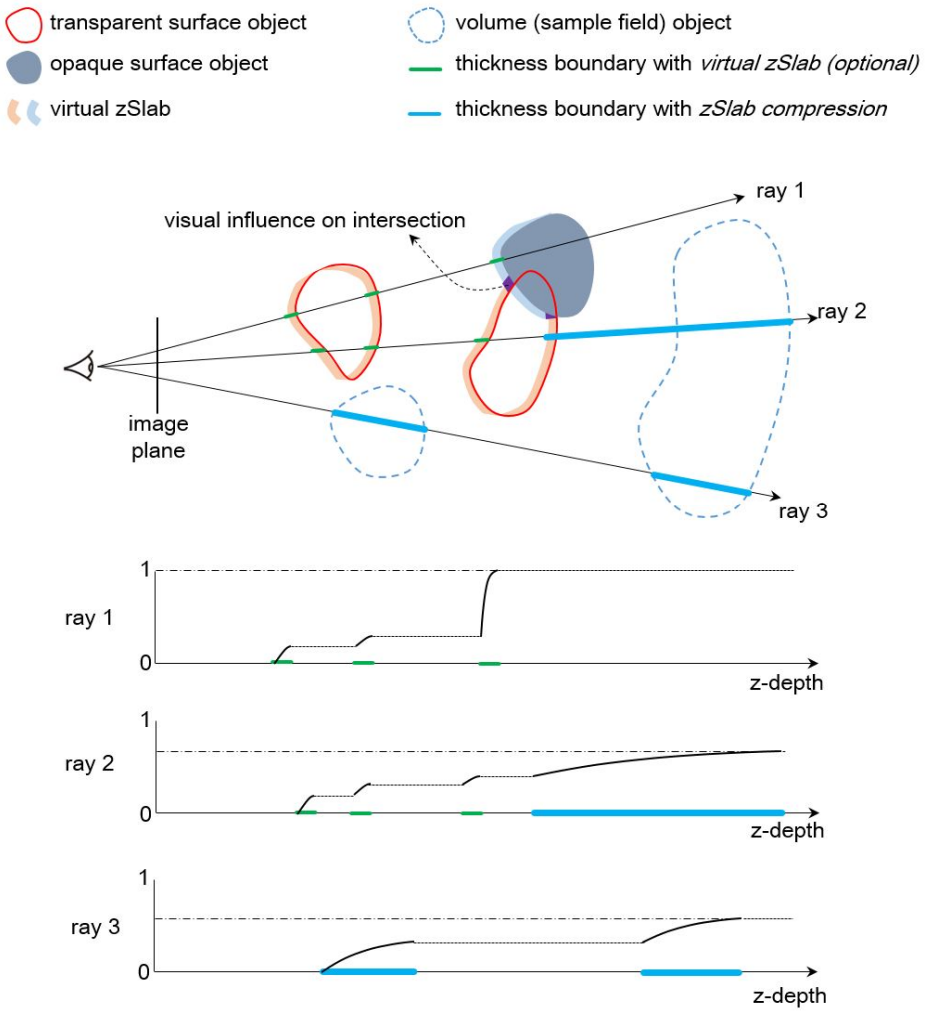


Figure 2.3. Illustration of zSlab construction for polygonal surface and volumetric surface geometries along a ray that starts at the camera origin and passes through a pixel in an image plane, accompanied by visibility function on each pixel. Here, each pixel stores a maximum of 4 zSlabs.

2.2.1. zSlab compression

It is ideal to have unbounded number of zSlabs for correct result [17]. However, in most applications, due to the limitation of memory capacity, the z-thickness buffer is allowed to have a bounded number of zSlabs [18]. Thus, when there are a large number of zSlabs over the number of depth layers of z-thickness buffer, zSlabs should be compressed (see the ray 2 example in Fig.2.3). In this dissertation, we take two compression models for this; tail-handling and zSlabs-merging.

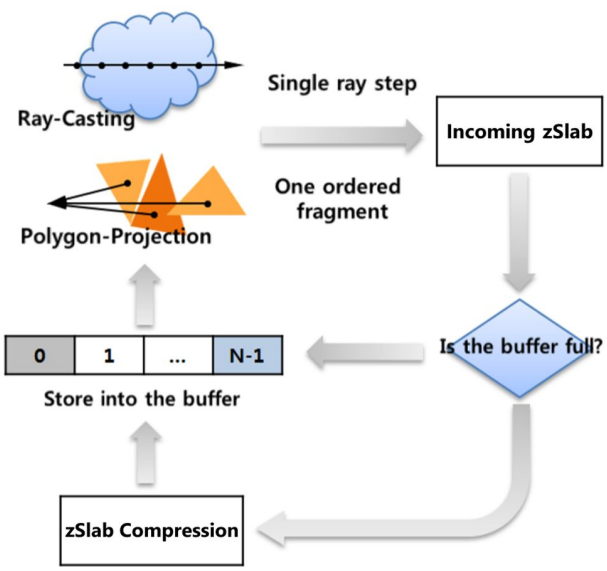


Figure 2.4. Illustration depicting the zSlab's workflow. Each renderer generates the incoming zSlabs. Before storing the zSlab to the buffer, zSlab compression is performed if the buffer is filled up with the zSlabs. The buffer is reused for the next generating of zSlabs.

Tail-handling [18] is a typical method to reduce the number of buffer layers storing z-directional boundaries, which is generally used in order-independent transparency techniques. Generally, this technique stores backward-remaining z-depth layers (after storing the foremost z-depth layers that the bounded size of buffer allows) into a single z-depth layer optionally in sorted order. The ray 2 example in Fig.2.3 illustrates how this tail-handling is achieved over the proposed zSlab model. Note that, in this compression, the target zSlabs may not be physically neighbored.

To intuitively address the zSlab-merging compression model, consider a volume ray-casting [35]. During the ray-casting, a slab [39] can be defined at every sample step and the zSlab is defined as a slab which has non-zero opacity (no air). Fortunately, the visibility along the ray-casting is smoothly integrated over the successive zSlabs, making the zSlabs easily merged. As strict and lossless compression algorithm needs lots of overhead, we use zSlab-merging compression using a simple greedy algorithm [34] described below.

Given a gradient of integrated visibility between two physically successive zSlabs, $\tilde{V}$, at sample step i and a merging criteria threshold ε (initially set as 0), our algorithm merges the successive zSlabs according to Eq. (2.7), which updates the threshold to maximum value at each sample step.

$$\tilde{V}_{\mathrm{i}} = \frac{|\sum_i V - \sum_{i-1} V|}{(entire\ thickness\ over\ two\ zSlabs)} \tag{2.6}$$

$$\tilde{V}_{\mathrm{i+1}} - \tilde{V}_{\mathrm{i}} < \varepsilon \tag{2.7}$$

Note that the compression occurs when the number of zSlabs is over the number of depth-layers of z-thickness buffer (see Fig.2.4). Fig.2.5 shows this example of a zSlab array constructed during the ray-casting.

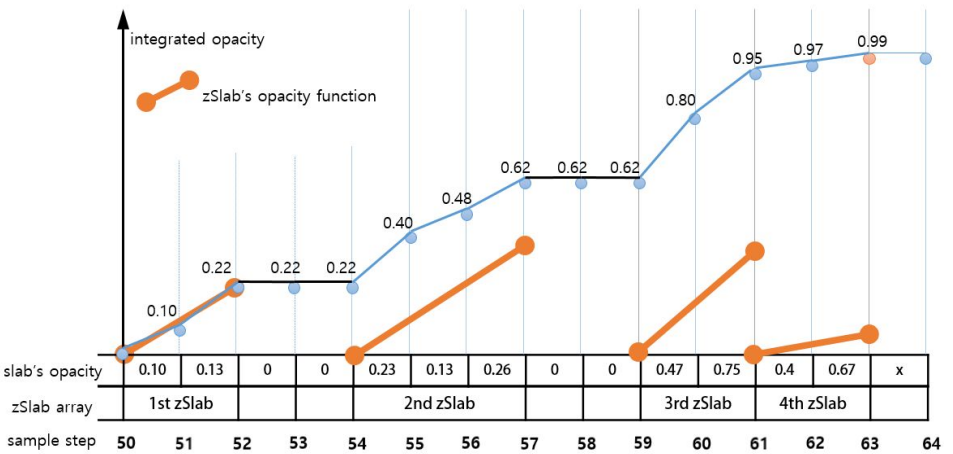


Figure 2.5. Example of zSlab array generated from a ray-casting. At each sample step, a slab is defined by an opacity value and its sampling interval. The derivative of accumulated opacity over a ray determines the zSlab compression rather than the difference between the neighboring zSlabs.

2.2.2. Virtual zSlab

In a rendering of polygonal object, a surface geometry is conventionally represented as an infinitely thin boundary. Thus, when rendering overlapping and intersecting the surfaces, they contribute to the final visibility in a binary way based on a single z-test, resulting in rendering artifacts on the intersecting and overlapping surfaces (e.g., randomly alternate dots [27] or jagged lines [37] at the

intersecting).

Based on the proposed zSlab model, we assign a finite thickness that extends towards the camera (only applied in the camera space, not in the physical space), making virtual zSlab, so that the visual influence of the surface is spread out across a finite thin distance (see Fig.2.3). With the proposed intermixing method mentioned in Chapter 3 and 4, this makes the smooth visibility transition (rather than binary visibility decision) on the intersecting and overlapping surface region, delivering anti-aliasing intersection in the multi-object rendering. Fig.2.6 describes how the virtual zSlab model contributes the anti-aliasing intersection using the smooth visibility transition. Note that the visual influence of the virtual zSlab derived from virtual thickness appears only in the intersecting or overlapping region of surfaces, which means the virtual zSlab has an effect locally on the problematic region with the smooth visibility transition, preserving the details of other region without the overlapping.

Fig.2.6 explains how the virtual zSlab provides the smooth and continuous visibility transition so that the intersection region exhibits continuous visibility change across thick boundary regions, of which the range is determined by the thickness of virtual zSlab. This makes the jagged lines on intersection (see Fig.2.6 (a)) disappears, providing anti-aliasing effect (see Fig.2.6 (c)) as good as the reference result of multi-sampling visualization (see Fig.2.6 (b)). One notable thing is that the virtual zSlab with too large thickness results in over-blurring (think that virtual zSlab model with large thickness in Fig.2.6(c)). To avoid this, the thickness of virtual zSlab should be carefully determined considering rendering conditions (e.g., in most cases, the smallest voxel pitch size is used when the scene includes

volumetric object, and in some special cases, user's manually setting thickness is used.)

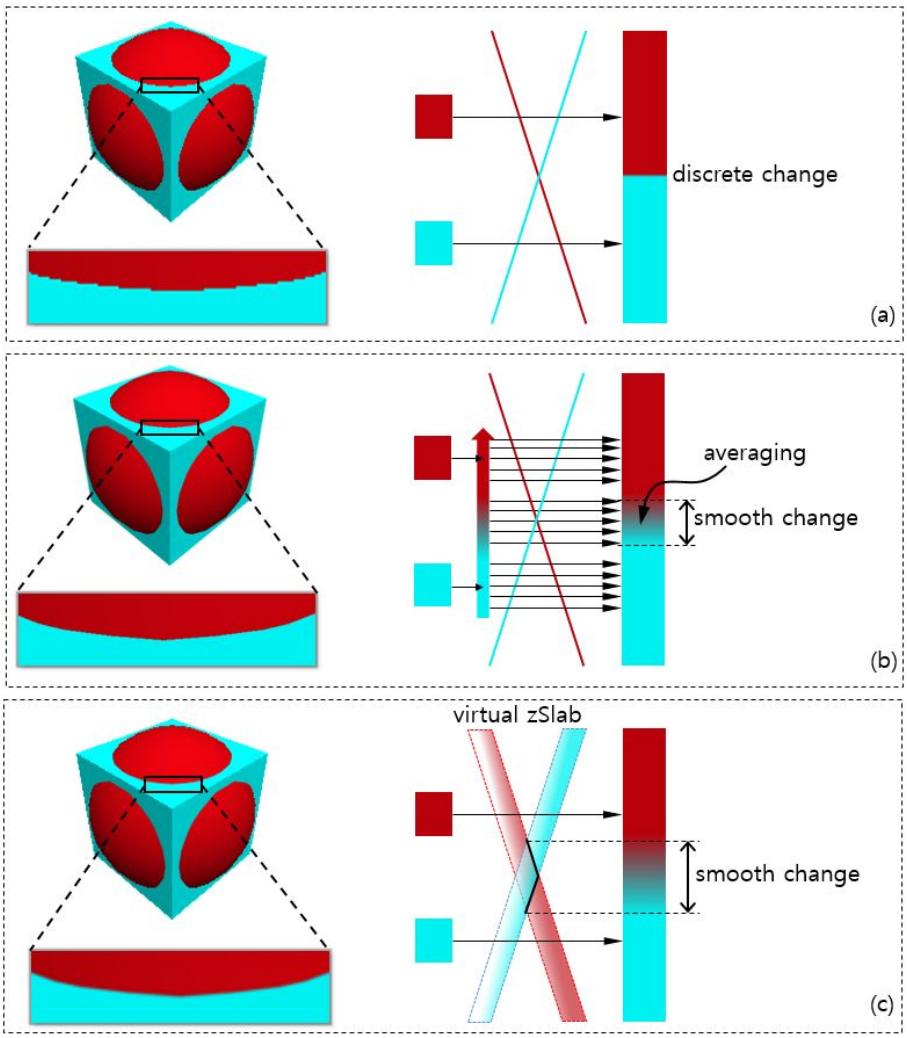


Figure 2.6. Illustration showing an intersection between two overlapping polygonal-objects; a blue cube and a red sphere. (a) is a conventional intermixing result with normal sampling (i.e., without scaling of resolution). (b) is a conventional intermixing result with (expensive) 4x4 multi-sampling (i.e., with super-scaling of resolution) as a reference image. (c) is our result based on the proposed virtual zSlab model with normal sampling (i.e., without scaling of resolution) showing anti-aliasing effect on the intersecting region as good as the reference image.

Chapter 3. Visibility Interpolation

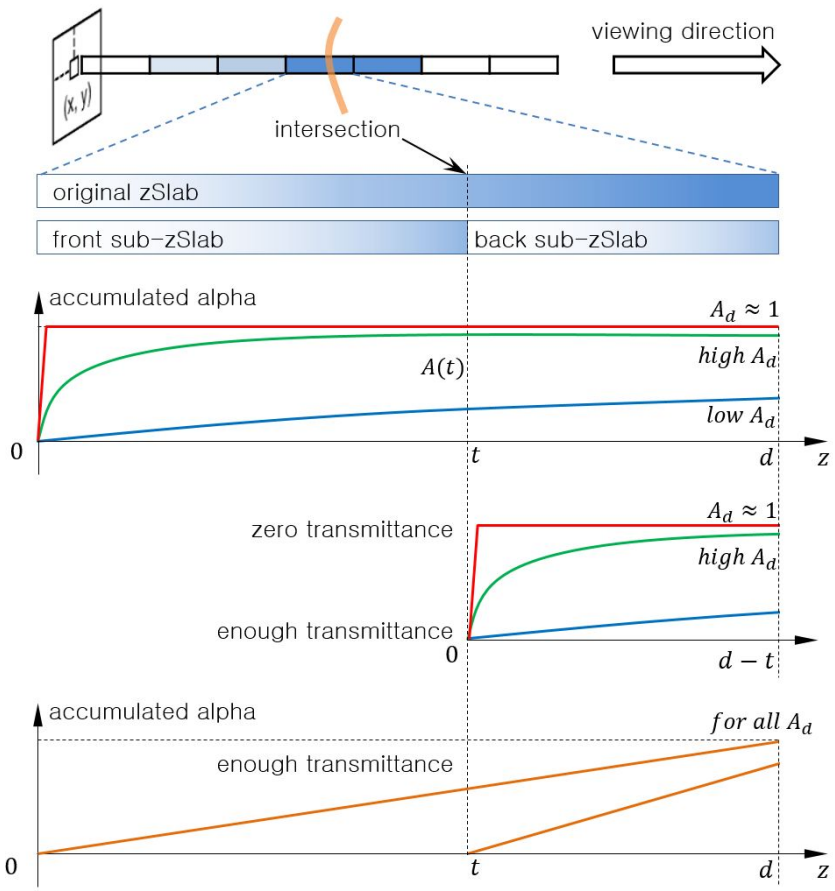


Figure 3.1. Illustration of the opacity behavior of original zSlab and sub-zSlab divided by an intersecting boundary, according to opaqueness and in-slab visibility interpolation model. The upper and middle plots show the opacity behavior of the original version of in-slab visibility interpolation. The lower shows the linear-like opacity behavior of the modified version of in-slab visibility interpolation.

3.1. zSlab subdivision

In the multi-object rendering of a scene that includes objects intersecting and overlapping regions on the objects, we need to consider zSlab subdivision in the proposed zSlab model. For example, when a zSlab has an intersecting surface represented as infinitely thin boundary (e.g., polygonal surface) in it, the zSlab should be subdivided to correctly visualize the intermixing units (i.e., zSlab and surface). For fragmented zSlabs resulting from the subdivision, referred as 'sub-zSlabs', it is necessary to determine the visibility of a sub-zSlab such that the visibility of the original zSlab is well preserved in its sub-zSlabs. During this, we try to avoid additional samplings causing performance loss and to provide consistent visibility transition on the interesting position. Fig.3.1 illustrates the zSlab subdivision into two successive sub-zSlabs; front sub-zSlab that is closer to the camera and back sub-zSlab that is far from the camera.

3.2. In-slab visibility interpolation

Since there is no information on the medium filling a zSlab, we have to take an assumption for the medium. Assuming homogeneous medium, the extinction coefficient $\tau(t)$ at a position t in the zSlab is constant τ under the simple emission-absorption optical model. Using Eq. (1.1) and the relationship between transmittance and opacity, the accumulated opacity up to the depth t within a given slab (t = 0 at the front z-depth in the slab) is defined as:

$$A(t) = 1 - exp(-\tau t) \tag{3.1}$$

For the zSlab having the thickness of d and its accumulated opacity of A_d, Eq. (3.1) is rewritten as:

$$A(t) = 1 - (1 - A_d)^{\frac{t}{d}} \quad (0 \leq t \leq d) \tag{3.2}$$

Since this is derived from transmittance of homogeneous medium of Eq. (1.1), opacities of both sub-zSlabs (i.e., front sub-slab and back sub-slab) can be defined using this equation. This is a function for the visibility interpolation (original homogeneous version). As the color radiance $\tilde{c}$, and the extinction coefficient function τ, and material's optical property are constant in homogeneous medium, the associated color of the front sub-slab is derived from eq. (1.2) and Eq. (3.2) with an original slab's visibility, (C_d, A_d), as follows:

$$\begin{aligned} C(t) &= \frac{\tilde{c}}{\tau}(1 - \exp(-\tau t)) \\ &= IA(t) = C_d \times \frac{A(t)}{A_d} \quad (0 \leq t \leq d) \end{aligned} \tag{3.3}$$

where I is a medium color that is also an intrinsic color property of the medium, which is computed by $\frac{\tilde{c}}{\tau} = \frac{C_d}{A_d}$ over the medium.

The plots in Fig.3.1 show two visibility interpolation models in terms of the

opacity along a ray through a zSlab. The upper plot in Fig.3.1 depicts homogeneous medium model that shows three cases of exponential behaviors based on Eq. (3.2). Assume that the surface intersection occurs at t in a zSlab and that the front sub-zSlab ranges from $z = 0$ to $z = t$ (illustrated in Fig.3.1). If the zSlab is filled with a high-opacity medium, the accumulated (or integrated) opacity of $A(t)$ for a front sub-zSlab gets close to the high opacity A_d (see the red and green lines in Fig.3.1). In this case, the front sub-zSlab with such high opacity blocks the visibility of the back sub-zSlab to contribute to the final visibility, even when its thickness is very small. This brings about discrete visibility transition over the slab intersection. The upper in Fig.3.2 shows the discrete visibility transition (see also Fig.3.3 (a)).

3.3. Modified version of visibility interpolation

In order to visualize the intersecting structures smoothly and continuously even in the opaque zSlab without such discrete visibility transition, we introduce a new visibility interpolation which makes the accumulated opacity of the front sub-zSlab increases linearly (see the bottom plot in Fig.3.1). Actually, this can be regarded as an approximation model of the homogeneous medium model of Eq. (3.2). However, with the linear increment behavior of the front sub-zSlab's opacity and the blending rule for the back sub-zSlab's opacity, this enables smooth and continuous visibility transition for all kinds of zSlab-based intermixing. Fig.3.2

shows how the smooth and continuous visibility transition is achieved by our modified version of the visibility interpolation.

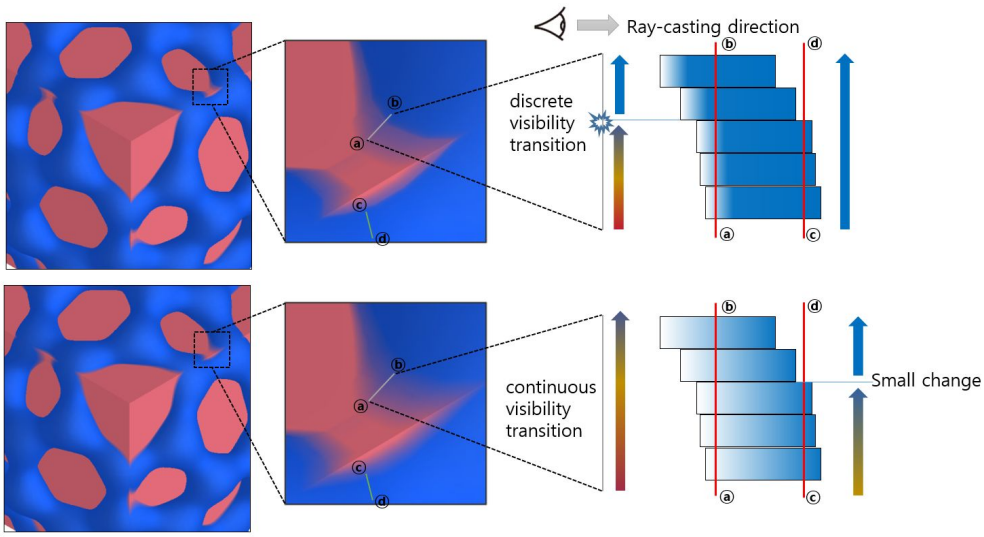


Figure 3.2. An intermixing example based on different in-slab visibility interpolation models; the upper based on the original version of in-slab visibility interpolation and the lower based on the modified version of in-slab visibility interpolation. The right most illustration shows how the in-slab visibility interpolation model determines the visibility transition; discrete or continuous (smooth).

The front sub-slab's opacity is simply defined in a linear form of the ratio of the sub-slab's thickness to the original slab's thickness.

$$A(t) = \frac{t}{d} A_d \quad (0 \leq t \leq d) \tag{3.4}$$

Exploiting the homogenous property of the material's color intensity and our approximation model of the opacity behavior over a single homogeneous medium, the associated color of the visibility is also defined similarly as in Eq. (3.4) by:

$$C(t) = \frac{A(t)}{A_d} C_d = \frac{t}{d} C_d \quad (0 \leq t \leq d) \tag{3.5}$$

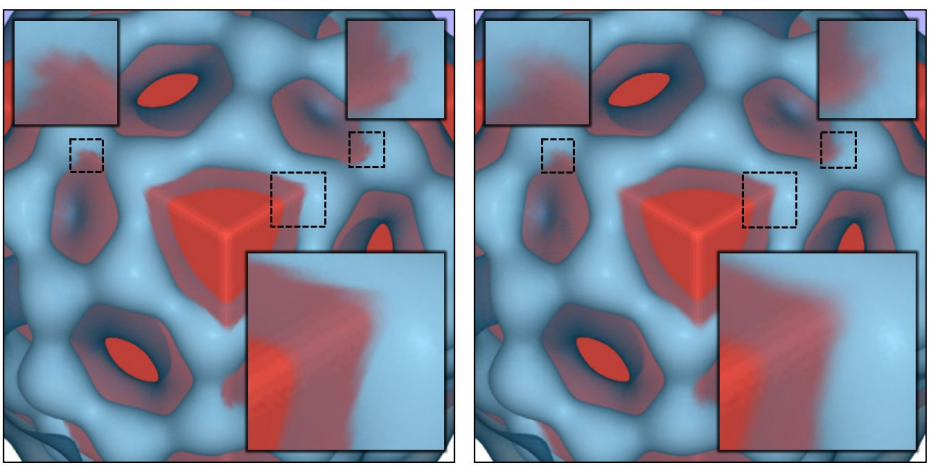
Because the opacity of Eq. (3.4) is not derived from the strict optics transmittance model, the opacity of the back sub-zSlab has to be determined in order to make the opacity accumulated across the front and back sub-zSlabs identical to the accumulated opacity of the original zSlab for correct visibility. If the opacity of the front sub-zSlab intersected at the depth position t, having the value of $A(t)$ by Eq. (3.4), the back sub-zSlab's opacity A_b can be derived from the blending operation of two successive slabs by Eq. (1.4) and (1.5). Thus, the visibility (associated color and opacity) of the back sub-slab is calculated as:

$$A_b = \frac{A_d - A(t)}{1 - A(t)}$$

$$C_b = \frac{C_d - C(t)}{1 - A(t)} \tag{3.6}$$

Eq. (3.6) is derived from the modified opacity based on the visibility correction such that the visibility of the original zSlab is well preserved in its sub-zSlabs. In this modified version of visibility interpolation, because both of the opacity and the associated color have the same form (Eq. (3.4) and eq. (3.5) for a

front sub-zSlab's visibility and Eq. (3.6) for a back sub-zSlab's visibility), the visibility of each sub-zSlab can be efficiently computed with vector parallel operation in contrast to separate computations for the accumulated color and the opacity as in the original visibility interpolation.



(a) (b)

Figure 3.3. Example showing the intermixing results including a volumetric object with a transparent thin layer of outer surface. (a) is based on the original visibility interpolation showing some intermixing artifacts such as banding and moire pattern caused by discrete visibility transition. (b) is based on the modified version of visibility interpolation showing smooth visibility transition, thereby reducing the intermixing artifacts of volume rendering.

Regardless of the original zSlab's opacity, our modified version of in-slab visibility interpolation enables enough transmittance for the back sub-zSlab to contribute the intermixing visibility while delivering smoothly increasing visibility contribution across the front sub-zSlab (see Fig.3.2). About the sampling theorem

point of view, the original in-slab visibility interpolation's behavior of the high-opacity zSlab is similar with the point sampling, which causes such discontinuity in visibility transition (see the upper in Fig.3.2) as well as moire aliasing artifact (see Fig.3.3 (a)). Whereas, the modified version of in-slab visibility provides smooth and continuous visibility transition, which leads to anti-aliasing effects on the high-opacity slabs intersection (see the lower in Fig.3.2 and see Fig.3.3 (b)).

The modified version of in-slab visibility interpolation is also available in the back-to-front integration model which is often used in many DVR applications. The blending equation of the back-to-front integration is $\sum_i C = C_i + (1 - A_i) \sum_{i+1} C$. Based on this equation, we can approximate the back sub-zSlab and derive the visibility of the front sub-zSlab using similar visibility interpolation approach through Eq. (3.5) and (3.6).

Chapter 4. Slab-based Visibility Intermixing

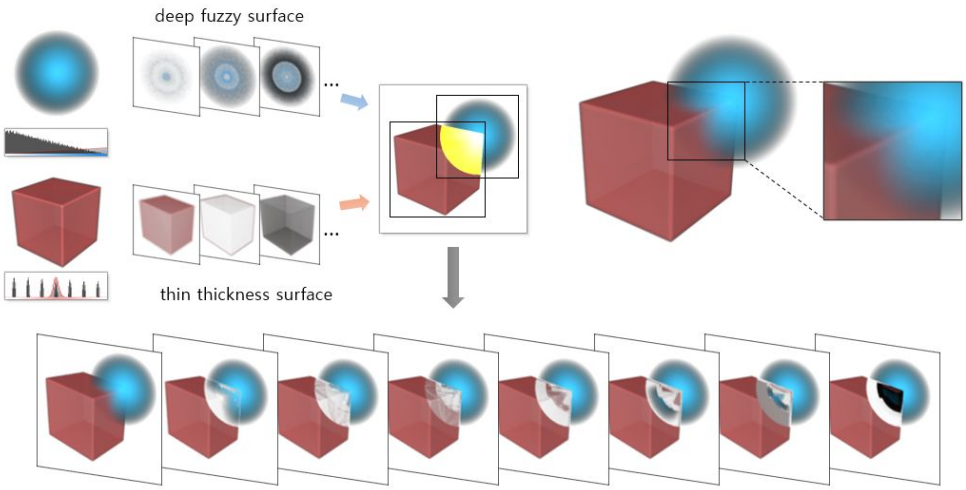


Figure 4.1. Illustration of the intermixing process of two different zSlab arrays, which is locally performed over the overlapping region (marked as yellow). The intermixing result is stored as a zSlab array stored in the z-thickness buffer, in which the layers are shown in the lower. The proposed method can well represent the fuzzy surfaces' intermixing as well.

Fig.4.1 illustrates the intermixing process of two different zSlab arrays of which each is from the diffident object rendering; deep-fuzzy surface rendering and thin thickness multi-layer surface rendering. In the proposed method, the intermixing proceeds only on overlapping region which is marked as yellow in Fig.4.1. The intermixing result is also composed of a zSlab array stored in z-thickness buffer, which is used for next intermixing with another participating object.

4.1. Intermixing of different zSlab arrays

Employing the in-slab visibility interpolation presented in Chapter 3.2, each zSlab is sub-divided by the another zSlab's front-side or back-side depth boundary. Aligned sub-zSlabs which share same front-side and back-side depth boundaries are supposed to be merged into an output zSlab array.

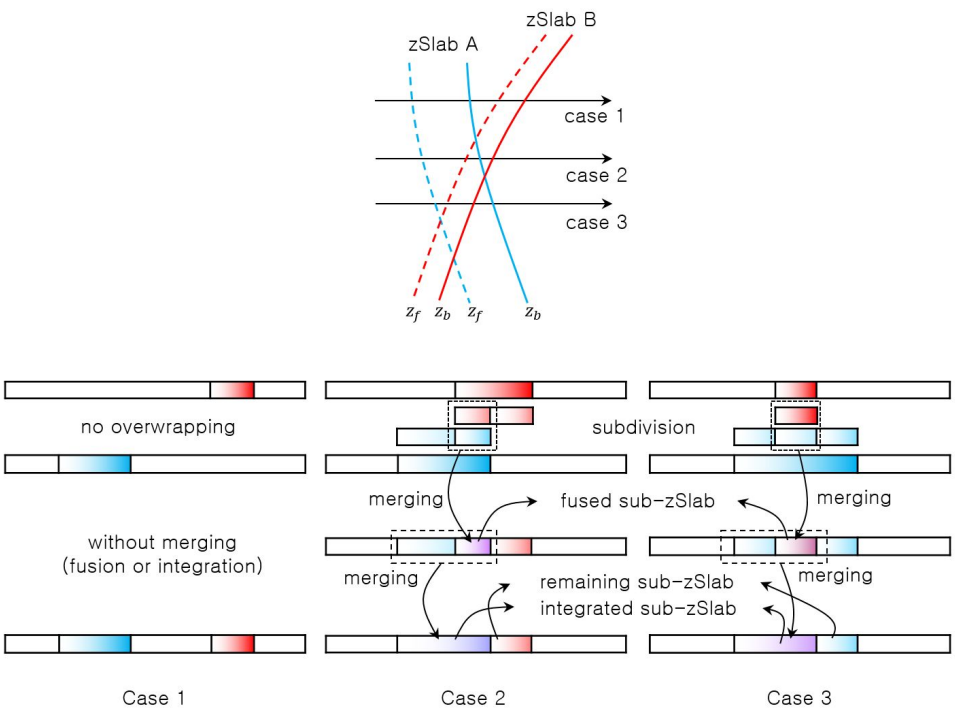


Figure 4.2. Thee intermixing cases of two different zSlab arrays; zSlab A and zSlab B. The bottom zSlab array is an output array stored in the z-thickness buffer.

Fig.4.2 illustrates how two different zSlab arrays intersect and merge each other. If a zSlab is intersected by the other zSlab's depth boundary, the intersected zSlab is subdivided into two sub-zSlabs. When there is no overlapping region (see case 1 in Fig.4.2), each participating zSlab composes the output sequence by sorting it in a depth-order. Note that the sorting must be computed according to the zSlab's back-side depth boundary.

When there is only a partial overlapping region on the side of each zSlab (see case 2 in Fig.4.2), each zSlab should be subdivided into an overlapping region and the other region. The visibility of each sub-zSlab can be determined by the in-slab visibility interpolation. The overlapping sub-zSlabs are merged to the fused sub-slab. Due to the limitation of memory capacity (note that the output zSlabs are supposed to be stored in the memory), the fused sub-zSlab doesn't need to directly join the output array supposed to be stored in buffer memory (or, z-thickness buffer). Instead, we employ the integrated sub-zSlab as an element of the output zSlab array, which merges the foremost sub-zSlab and the fused sub-zSlab. Accordingly, the number of elements of the output zSlab array is less than or equal to the number of the participating zSlabs. After generating the integrated sub-zSlab, the remaining part of the participating sub-zSlab, called remaining sub-zSlab, is supposed to join the output zSlab array.

When a participating zSlab is aligned inside the other zSlab region (see case 3 in Fig.4.2), the zSlab that covers the other one is supposed to be separated into three pieces of sub-zSlabs. In this case, the visibility of each sub-zSlab can be obtained by the iterative interpolation according to the front-to-back direction. The

middle one of the sub-zSlabs and the other participating zSlab are merged to the fused sub-zSlab. With the same merging manner of case 2 of Fig.4.2, the output zSlab array can be obtained.

If there are more than two participating zSlab arrays, first two arrays are merged into the output zSlab array, and then the other is supposed to sequentially participate in the intermixing with the previously generated output zSlab array, one by one. Even if this strategy brings about different results according to the participating order of the zSlab arrays, this makes the intermixing simple to implement. As long as the final output zSlab array is obtained, the final visibility is then computed by accumulating with the zSlabs in the output zSlab array (normally use the 0-index layer of z-thickness buffer).

4.2. Intermixing complexity

Since the more intermixing zSlab arrays, the more complicated slab-based intermixing cases, we have to focus on only two intermixing zSlab arrays by exploiting deferred rendering strategy [40] as multi-pass rendering stages. Each rendering stage has two participating z-thickness buffers of which one stores an input of previously rendered (or intermixed) zSlab array and the other one stores an output which is the intermixing zSlab array. The output is supposed to be sequentially the input of a next rendering stage. The intermixing of two different zSlab arrays proceeds with a single scanning of an input zSlab array during a

traversing of current rendering zSlabs, which requires only $O(N+M)$ complexity (M is the number of slabs in the input zSlab array and N is the number of current rendering zSlabs). Also, this provides simpler implementation and better optimization especially in GPU with limited branch operations and resources [43].

4.3. Merging operation over the overlapping zSlabs

There are many methods to merge the aligned zSlabs [26, 44] (to the fused zSlab). In this dissertation, we focus on more continuous and natural visibility transition on mixing structures. In order to achieve this, we use the accumulation level intermixing [27]. This is well suited for our slab-based intermixing since the visibility of zSlab is determined by illuminated associated-color. The intermixing opacity A_{mix} is calculated to represent the accumulative effect caused by the opacities A_i from the aligned overlapping zSlabs of different zSlab arrays as follows:

$$A_{mix} = 1 - \prod_{i=1}^{n}(1 - A_i) \tag{4.1}$$

where n represents the number of overlapping slabs while i represents i-th intermixing object.

The fused zSlab is filled with a different medium from the participating zSlabs, which has a new medium color intensity, I_{mix}. By using the normalized sum of

each medium color of the overlapping zSlabs, I_i, according to each alpha (i.e., opacity) value, the medium color intensity of the fused one is obtained by

$$I_{mix} = \frac{\sum_{i=1}^{n} I_i A_i}{\sum_{i=1}^{n} A_i} = \frac{\sum_{i=1}^{n} C_i}{\sum_{i=1}^{n} A_i} \tag{4.2}$$

and, thereby, the associated color, C_{mix}, is computed by

$$C_{mix} = I_{mix} A_{mix} \tag{4.3}$$

Note that our intermixing model considers only two participating zSlab arrays ($n = 2$), which enables the merging operation to be simple. And, during the intermixing operation, the opacity of subdivided zSlab may be zero, which leads to dividing by zero in Eq. (4.2). Thus, we use the modified formula using bidirectional blending and averaging for the association color of the intermixing visibility, C_{mix}, as follows:

$$\mathrm{C_{mix}} = \mathrm{C_1} + \mathrm{C_2} - \frac{1}{2}(\mathrm{C_1 A_2} + \mathrm{C_2 A_1}) \tag{4.4}$$

where C_1 and C_2 are associated color of each overlapping slab's visibility.

Chapter 5. Implementation

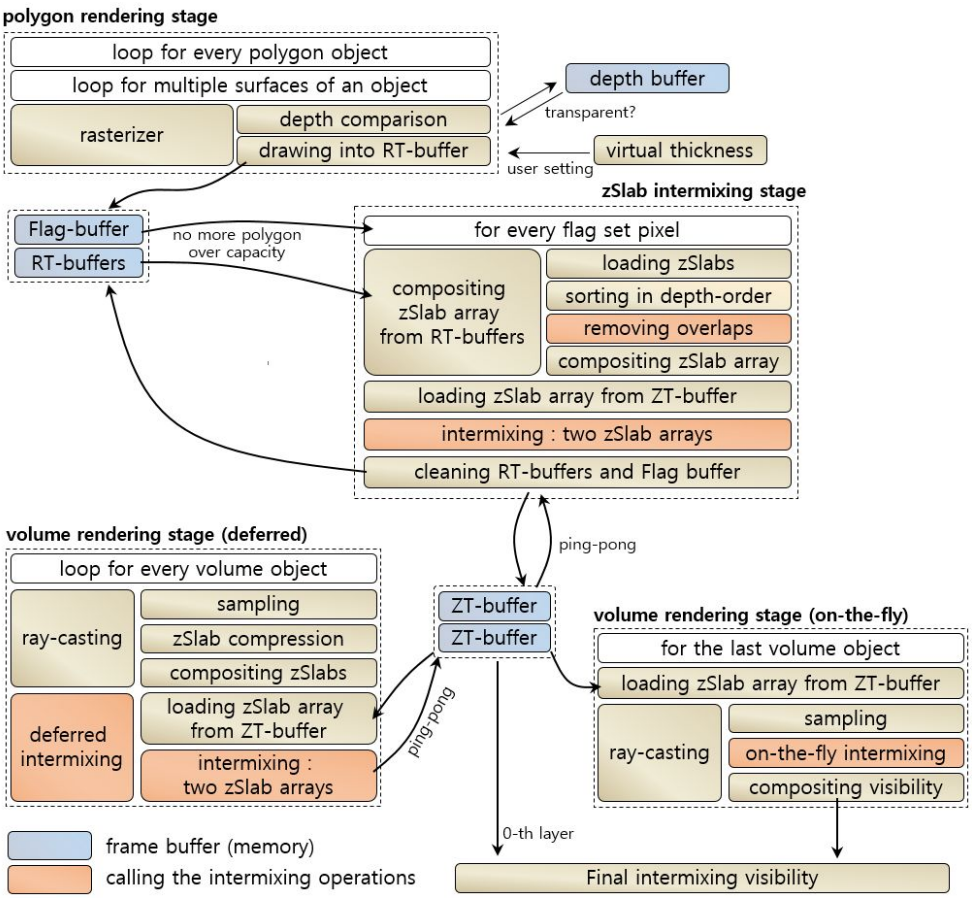


Figure 5.1. The intermixing pipeline that treats the proposed zSlab model and intermixing operations. There are three rendering stages according to the participating object type, in which each exchanges its input and output through the z-thickness buffer.

5.1. Implementation of zSlab

A zSlab consists of back-side boundary depth (float type), thickness (float type), and non-zero visibility (RGBA, optional for byte channels or float channels). The size is 12 bytes or 24 bytes per zSlab. Each pixel on z-thickness buffer as a frame-buffer stores such a zSlab array that the next rendering pass is able to use the zSlab array. As the storage size of a pixel is limited, the number of zSlabs has to be reduced by using zSlab compression; merging similar series of physically successive zSlabs into a single zSlab or integrating the posterior zSlabs into the last zSlab as tail-handling.

5.2. Overall intermixing pipeline

The proposed intermixing pipeline consists of three rendering stages to combine different rendering techniques. This is based on the image-level intermixing that the result of each rendering pass is used as an input of the intermixing pass similar to deferred rendering approach [40]. Fig.5.1 depicts a high-level overview of our intermixing pipeline including the three processing stages; 1. polygon rendering, 2. zSlab intermixing, and 3. volume rendering.

5.2.1. Buffers

In order to store the result of each rendering stage, we employ two types of frame buffer: RT-buffer and ZT-buffer. RT-buffer is a render-target-texture buffer that stores a zSlab of a surface per pixel through a single polygon rendering pass. Here, we employ M number of RT-buffers of which each stores a single surface of the rendering object. ZT-buffer is the proposed z-thickness buffer that stores the ordered zSlab array. Here, we designed the z-thickness buffer that stores N number of zSlabs. For skipping of trivial pixels, we mark a bit flag in every drawing pixel, which is stored in Flag-buffer.

Because ZT-buffer is used not only as an output of the current intermixing but also as an input of the next intermixing, two of ZT-buffers should be allocated in GPU implementation. We exploit the ping-pong strategy [43] to efficiently use ZT-buffer in the intermixing pipeline without additional allocation regardless of the number of rendering stages. Thus, each pixel in the frame buffer requires $(N \times 2 + M) \times sizeof(zSlab)$ bytes in the entire intermixing pipeline.

For the Flag-buffer, we use Integer-type array as a buffer rather than frame-buffer, of which an element encodes 32 number of pixels. Otherwise, for the rasterizer speed optimization, depth-stencil buffer can be used so that the fragment processing (or, pixel shader) on the trivial pixels is skipped during the rasterization processing.

5.2.2. Polygon rendering stage

For the polygon rendering of transparent geometry, there are two typical ways; polygon ray-tracing and raster graphics. For the polygon ray-tracing, efficient ray traversing technique should be needed such as BVH and global memory hierarchy on GPU implementation. Each ray per pixel computes the intersecting points of the polygons and gives ordered layers of surfaces, but this needs high-level performance GPU architecture for real time rendering, which is not suitable to many commercial applications. Thereby, we use raster graphics with multi-pass strategy for polygon rendering stage rather than the polygon ray-tracing.

In the polygon rendering stage based on the raster graphics, every polygon object calls L times of rendering passes, such that L number of transparent surfaces are stored into RT-buffers by using conventional depth test or dual-depth test scheme [44] with depth-peeling technique [19]. At every M times of polygon rendering passes, which implies that there could be pixels filled up with zSlabs in RT-buffers, zSlab intermixing stage is called up for generating a zSlab array.

The virtual thickness can be differently set at every rendering pass. Generally, the thickness is determined according to the object rendering condition such as object size (static parameter) or pixel size on the projecting region (dynamic parameter). For a special visualization technique that breaks the optics to show more perceptive image, user's manual setting is also available.

5.2.3. zSlab intermixing stage

As there are M number of RT-buffers, at every M times of polygon rendering, there could be pixels filled up with zSlabs in RT-buffers. zSlabs directly loaded from RT-buffers could not be sorted in depth-order and could have self-intersection regions. Thus, the zSlabs are to move into the ZT-buffer while sorting them in depth-order and removing the self-intersecting region in the zSlab intermixing stage. For the speed optimization, this stage is available only on the drawing pixels set in the Flag-buffer of the polygon rendering.

If there is an input ZT-buffer from the previous rendering stage, the intermixing of two zSlab arrays should be performed so that a new zSlab array is generated and then stored in the other ZT-buffer. Then, RT-buffers and Flag buffer are cleaned and recycled in the polygon rendering stage.

5.2.4. Volume rendering stage

Every volume object calls a single ray-casting pass. Each sampling step during the ray-casting generates a slab that is sorted in depth-order and has no overlapping, so that the additional sorting or removing overlaps operations are not required.

For the volume rendering with deferred intermixing, the ray-casting composites zSlab arrays using zSlab compression. Then, the zSlab array is

supposed to be mixed with the previous zSlab array from the ZT buffer. The intermixing result is stored in the other ZT-buffer. All of those operations are performed in a single pass so that the overhead of multi-pass rendering is reduce.

Otherwise, for the volume rendering with on-the-fly intermixing, the previously loaded zSlab array participates in the ray-casting and the intermixing is performed during the ray-casting, referred as 'on-the-fly intermixing', so that more accurate intermixing result is available than the result of deferred intermixing version of volume rendering. This is an optional stage of our intermixing pipeline and, generally, this is applied to the last volume rendering object.

5.3. Intermixing kernel for zSlab merging

The intermixing kernel targeted on our intermixing pipeline is a basis function that is used for the rearrangement of unordered zSlabs with sorting in depth-order and removing the overlapping region of the zSlabs, generating zSlab array, and the intermixing of two different zSlab arrays that are aligned in the same ray.

Fig.5.2 describes the intermixing kernel algorithm. Each zSlab is represented by visibility function, thickness, and back-depth defined in camera space. The intermixing kernel handles only two participating zSlabs that are sorted in depth-order according to the back-depth; a prior zSlab which is close to camera and a posterior zSlab which is far from camera. By using the intermixing strategy described in the previous Chapter, the output of the intermixing kernel is an

integrated zSlab with the back-depth of prior zSlab and a remaining zSlab with the back-depth of posterior zSlab. The integrated zSlab is supposed to join the zSlab array while the remaining zSlab participates in the intermixing kernel. Unless there is a zSlab as a participant, the remaining zSlab also joins the zSlab array.

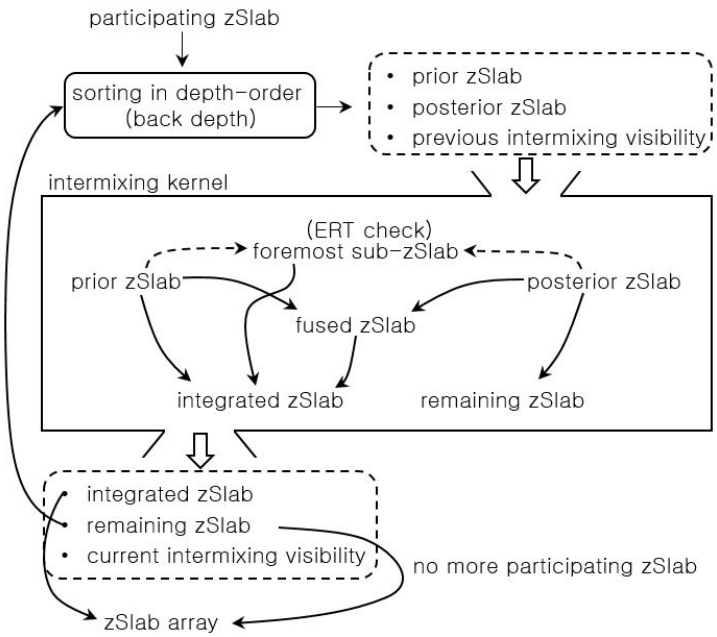


Figure 5.2. Diagram for the intermixing kernel function for each overlapping zSlab. The instructions of this kernel are for three branches, two of in-slab visibility interpolation operations, and one merging operation. At every processing of the kernel function, the kernel updates the intermixing visibility, checking the ERT condition.

Because the foremost sub-zSlab also contributes to the final visibility value during the intermixing kernel, the alpha of final visibility value can exceed

opaqueness threshold before the prior zSlab is fully integrated into the integrated zSlab. In this case, the entire intermixing process is terminated, called early-ray-termination [32].

An intermixing kernel includes three branches for classifying the intermixing cases and checking ERT, two of in-slab visibility interpolation operations, and one merging of overlapping zSlabs. Thus, the computational cost is small enough to preserve real-time performance. This enables our intermixing method to be acceptable even in the on-the-fly computation during the ray-casting.

5.4. Thickness determination of virtual zSlab

The thickness of the virtual zSlab is determined according to its rendering purpose and objects-compositing-scene. In Fig.5.3 (a), the surfaces of a red polygonal cube and a blue polygonal sphere suffer from jagged edges on intersection without virtual thickness based on the conventional z-testing. However, when they are represented as virtual zSlabs with finite thickness, the jagged edges disappear, providing better intersection as shown in Fig.5.3 (b). One notable thing is that virtual zSlabs with too large thickness introduce over-blurring as shown in Fig.5.3 (c). To avoid this, the thickness of virtual zSlab should be carefully determined considering rendering conditions.

The virtual thickness can be differently set at every rendering pass. Generally, the thickness is determined according to the object rendering condition such as

object size (static parameter) or pixel size on the projecting region (dynamic parameter). When there is at least one volume object as a scene participant, the smallest voxel pitch size is generally used as the virtual thickness. Otherwise, we can set the dynamic thickness value according to the pixel size of the projecting polygons, so that anti-aliasing effect is delivered over the pixels. For a special visualization technique that breaks the optics to show more perceptive image, user's manual setting is also available (see Fig.6.12 and Fig.6.14).

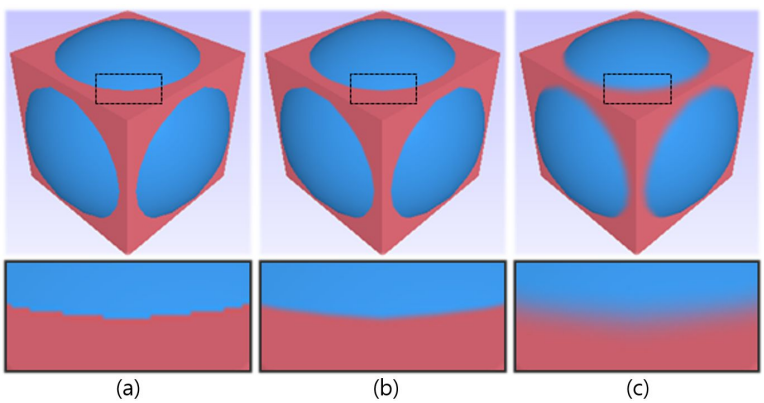


Figure 5.3. Three examples of two overlapping polygonal-objects (a blue sphere and a red cube) that depict the virtual zSlab effect according to the virtual thickness. (a) shows aliasing intersection as a result without virtual thickness same as the conventional intermixing result. (b) shows anti-aliasing intersection which is desirable result using the proper size of virtual thickness. (c) shows over-blurring intersection with over-estimated virtual thickness.

5.5. Optimization advantages

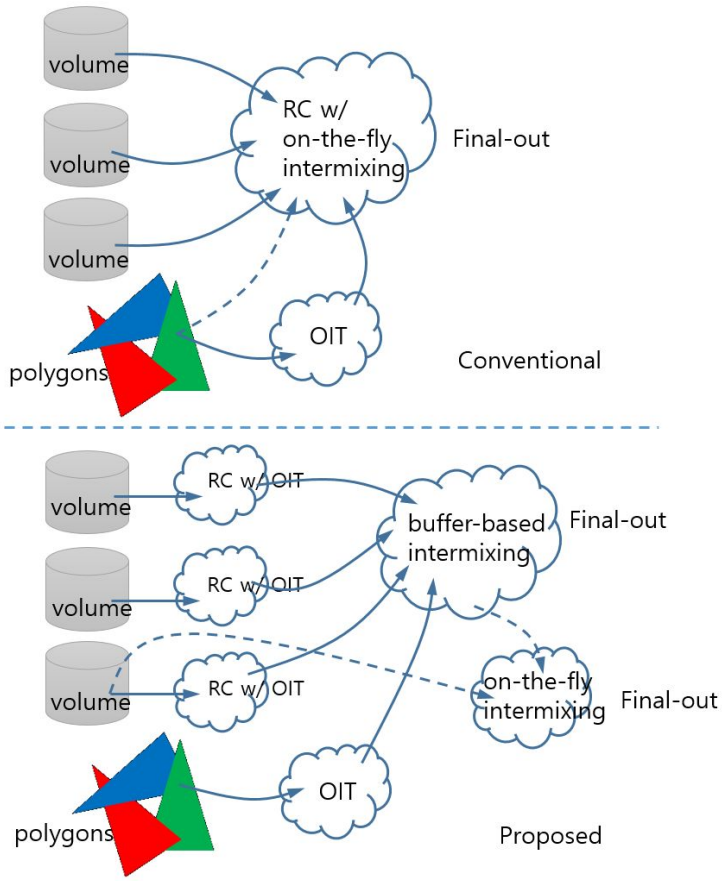


Figure 5.4. Comparison between the conventional intermixing pipeline and the proposed intermixing pipeline. the conventional intermixing pipeline focuses only on the on-the-fly intermixing without special cares for the image-level intermixing, whereas the proposed intermixing pipeline well treats the image-level intermixing (full image-level intermixing) as well as on-the-fly intermixing (hybrid-level intermixing) based on deferred rendering approach.

The conventional intermixing pipeline allows on-the-fly intermixing for a scene that includes volume objects. As mentioned, this results in optimization issues considering all the participating objects. Different geometries of the participating volume objects make an optimization technique (e.g., empty-space-skipping, or surface refinement) difficult and inefficient in terms of implementation as well as its performance. Also, this results in over-sampling problem that uses the sampling interval of the global ray-casting for the participating volume objects.

On the contrary, the proposed intermixing pipeline allows the intermixing scene to be performed through image-level intermixing approach, so that the optimization techniques are well preserved due to the rendering independency. This allows easy empty-space-skipping, surface refinement, and essential sampling distance of each ray-casting to the entire intermixing. Moreover, as the prior volume rendering results can be stored in z-thickness buffer, we can optionally choose the on-the-fly intermixing with the prior rendered z-thickness for more accurate result. Usually, to exploit the advantages of the rendering independency, this on-the-fly intermixing is performed during the final volume rendering (referred as 'hybrid-level intermixing'). Fig.5.4 illustrates an example of the rendering independency.

Chapter 6. Case Studies and Results

6.1. Experiment condition

All experiments were performed on an Intel i7 PC with a 3.2 GHz Quad-core processor and 16 GB of main memory. The system was also equipped with a NVIDIA GTX 780 GPU with 2 GB of memory. The proposed algorithms including rendering and intermixing are all accelerated by GPU programming using Direct3D 11 graphics API with HLSL shader model 5.0.

The proposed implementation is composed of two major steps. The first step is to render individual objects while generating their corresponding per-pixel zSlab arrays. The second step is to intermix all the zSlab arrays for all objects. As discussed earlier, for a polygonal object, its zSlab array is not generated during the first step of individual object rendering. Instead, it is made right before entering the second step of intermixing. For the rendering of polygonal datasets, we implemented the raster-based polygon rendering with depth peeling [19]. And for the rendering of volumetric datasets, we implemented the ray-casting algorithm using the pre-integration slab [39] and Phong-Blinn illumination [16]. In addition, for high-quality iso-surface representation [46], all the ray-casting for a single volume adopted the surface refinement method [16, 45] with 10-bisectional intersection. Also, our DVR employed the optimization techniques [32] using block-based empty space skipping and early ray termination.

Considering a constant number of read-back buffers and its performance tradeoff, in this dissertation, we set the maximum number rendering pass of an object as 4 to treat the maximum 4 foremost transparent surfaces for each polygonal object model.

6.2. Smooth visibility transition study

We already described how the effect of the visibility interpolation appears according to two intermixing scenarios: 1. infinitely thin surface intersecting high-opacity zSlabs (see Fig.3.2 and Fig.5.3), 2. two different high-opacity zSlabs overlapping together (see Fig.6.1). As low-opacity zSlab guarantees enough transmittance of its sub-zSlabs and shows little visual-advantage in terms of smooth visibility transition, we focused on the high-opacity zSlab intermixing scenario to show clear advantage of smooth visibility transition based on the proposed method.

6.2.1. High-zoom and high-opacity of MDVR

We conducted experiment to compare the proposed method with conventional multi-volume direct volume rendering (referred as 'MDVR'). MDVR renders multiple objects via a volume ray-casting algorithm while sampling intensities

from all the objects simultaneously and then intermixing those sampled intensities at a given sampling position. This is a straightforward extension of the single volume ray-casting algorithm. For the comparison, the proposed method and the MDVR have the same sampling interval.

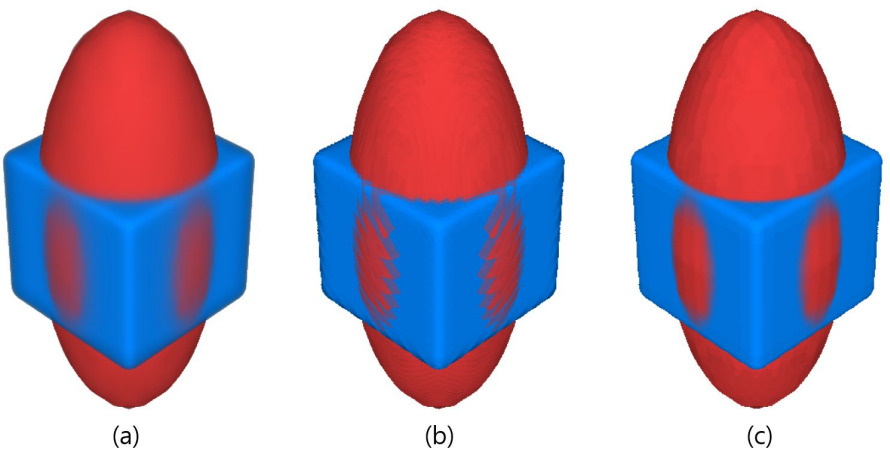


Figure 6.1. Comparison of high zoom-in rendering results of two high-opacity volumetric geometries. Each edge of the cube and the radius of the ellipsoid is represented by 15 voxels. (a) is a conventional multi-volume rendering result with super-sampling x100, (b) is a conventional multi-volume rendering with normal sampling x1 (c) is our intermixing result with normal sampling x1, showing high-quality image (on the overlapping region) as good as the super-sampling result.

It is attributed to the fact that the conventional MDVR does not refine the intersecting position of the objects (i.e., surface refinement) sampling the participating volumes at the same sampling position at each sample step. And, the proposed intermixing handles the partially overlapping ray-slab and makes it to

contribute the final visibility value so that the high-quality visualization of the intersection is achieved without super-sampling.

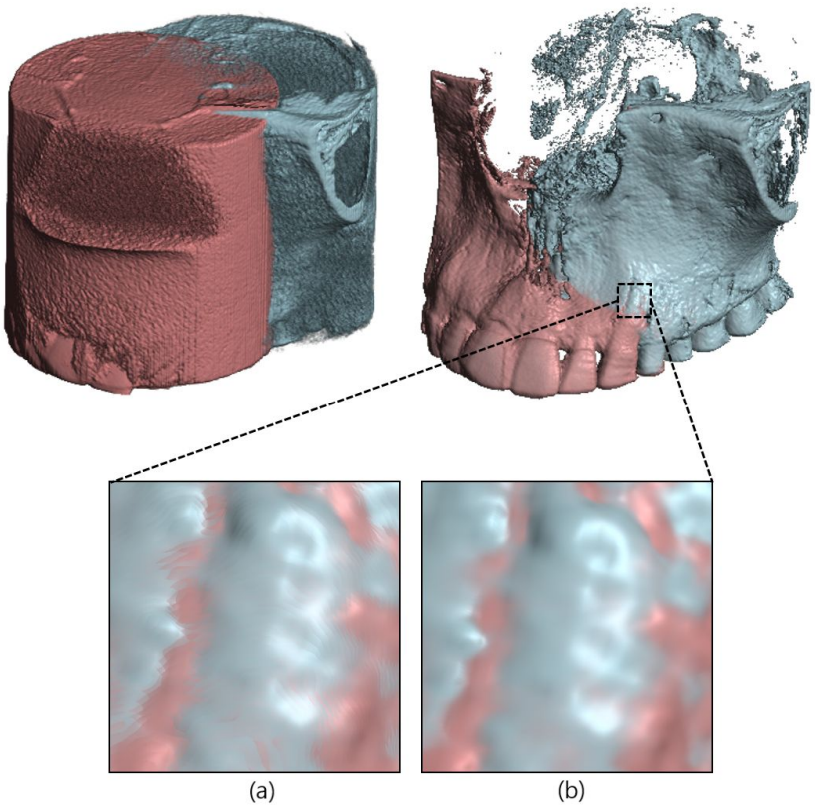


Figure 6.2. Comparison of rendering results of two dental CT scans of each volume is rendered through high-opacity transfer function. (a) is a conventional MDVR result showing the banding artifact on the overlapping area. (b) is the proposed intermixing result with clear and smooth visibility change, which is a superior image.

The visualization effect can be often identified in high zoom-in cases to see

detailed parts of the intersecting structures. For the simple test, we set two of high-opacity volumes with low resolution voxels (see Fig.6.1). Note that our deferred intermixing approach enables various optimization techniques (i.e., surface refinement, empty-space-skipping and early-ray-termination) to be easily applied to each volume rendering in contrast to the conventional MDVR. Fig.6.1 (b) shows severe aliasing artifact over the intersecting region. Fig.6.1 (c) well represents the intersection region as good as the super-sampling result, Fig.6.1 (a).

The renderings of maxilla (i.e., upper jaw) stitched from two dental CT scans in Fig.6.2 also show the similar case that the conventional MDVR exhibits the severe banding artifact at the stitching intersection of the two CT scans, while the proposed method provides much smoother stitching intersection. This shows that the proposed method works well in the practically high-opacity multi-volume rendering case as well.

6.2.2. Opaque surface rendering of stitching volumes

Sometimes, the surface geometry of the volume object (commonly, outer-most structure) is represented as an iso-surface using the ray-casting with surface refinement technique (referred as ‘iso-surface rendering’) [45, 46], so that the detailed (outmost) surface is visualized as a surface rendering. As the rendering image represents a single opaque layer of the foremost surface, N times of rendering is performed for the visualization of the N number of volumes.

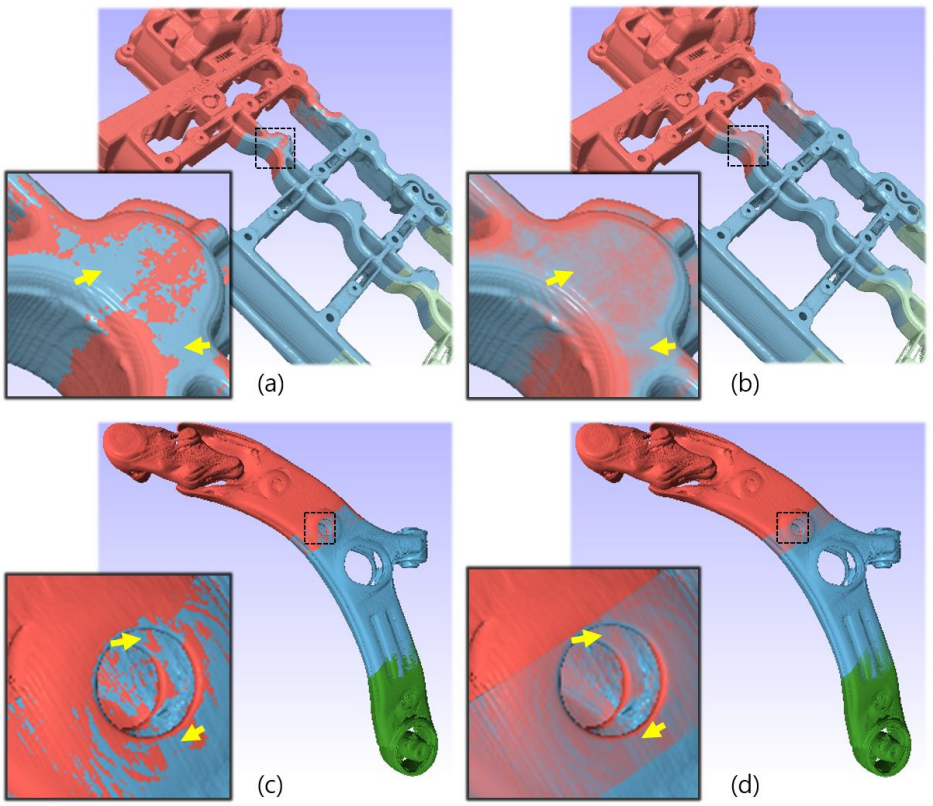


Figure 6.3. Comparison of iso-surface rendering results of two stitching cases. Each dataset consists of three different CT scans; the upper scans an engine part, the lower scans a crankshaft. The left side (a) and (c) are the results based on the traditional z-test showing the visibility in binary way (the foremost surface only). The right side (b) and (d) are the results based on the proposed intermixing method showing not only the foremost surface but also geometry perception of close surfaces (how close they are), using the smooth visibility transition.

In the industrial CT imaging, a component with a big size is usually scanned partly. In this case, a couple of CT scan volumes are obtained, of which each volume comes in a big size. Typically, those big volumes are registered semi-automatically to represent the whole part of the component. Even after registration, users often want to refine the position of one volume (referred as 'floating volume') while fixing that of another volume (referred as 'reference volume').

Fig.6.3 shows the stitched iso-surface renderings of three partially scanned datasets of large mechanical devices. Conventional iso-surface rendering of multiple volumes using single z-test shows random and discrete visibility changes in overlapping regions due to the visibility decision in a binary way (Fig.6.3 (a) and (c)). In contrast, the proposed method provides smooth visibility transition using a ray-sample slab (well suit for zSlab surface based on the thickness boundary) on iso-surface as a zSlab, while delivering more intuitive perception on the overlapping regions.

As shown in Fig.6.3 (b) and (d) which are the rendering result using the proposed intermixing method, they provide better renderings of overlapping regions compared to the conventional iso-surface rendering in terms of the representation of both extra- and intra-structures and the reduction of random visibility changes on the overlapping surfaces. The surface close to blue color means that the blue surface is closer to the camera than the red surface. If the surfaces are close to each other, the fused color is visualized on the surface based on the proposed visibility transition model described in Fig.2.6.

This effect often occurs even in the multiple polygonal-objects that include

some polygons close to each other's polygon. And the proposed method provides naturally smooth and continuous visibility change in such overlapping regions as well shown in Fig.6.4.

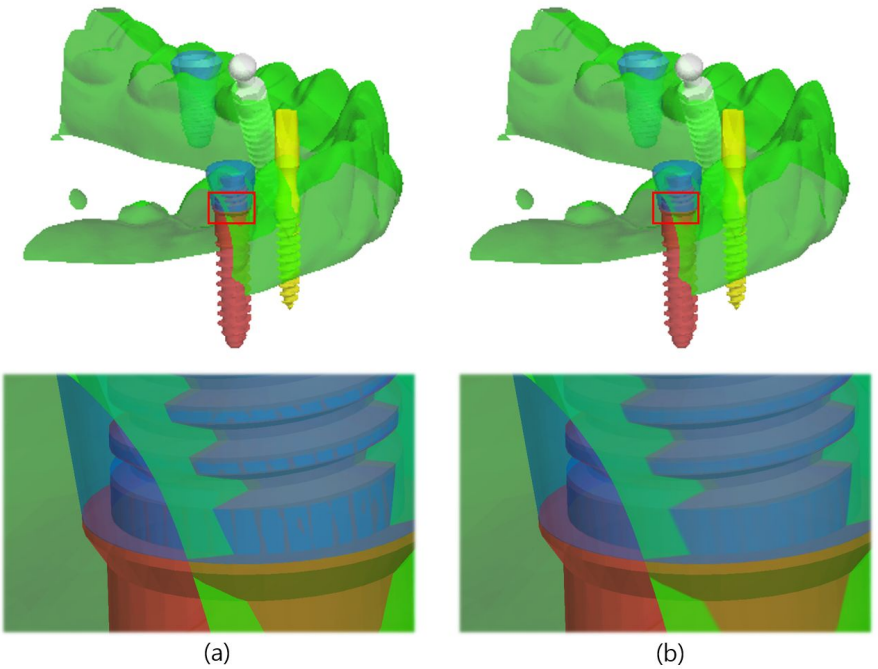


Figure 6.4. Comparison of rendering results of transparent surfaces overlapping each other. (a) is based on a traditional z-test showing the randomly visible change. (b) is based on the proposed zSlab model showing more continuous visible change.

6.2.3. Z-fighting artifact on overlapping surfaces

There are often the different surfaces sharing a same plane in the multi-object scene. The conventional intermixing method (based on a single z-depth test on the overlapping geometry) represents the visible surface in binary way, and machine computation error [48] occurs in projecting the surface during the rendering, thereby resulting in a typical intermixing artifact, called z-fighting [36] that brings about randomly alternate dots shown in Fig.6.5 (a).

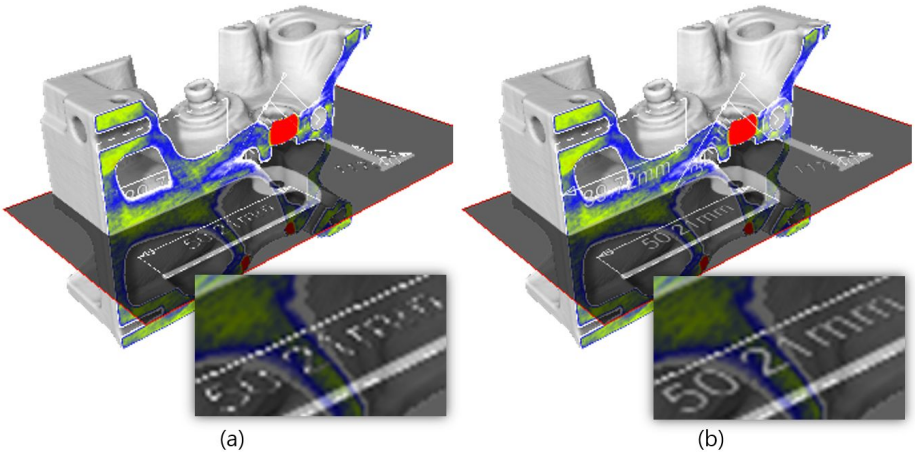


Figure 6.5. Comparison of rendering results of planar objects sharing a same depth; measurement widgets on the superimposed plane or clipping plane of the main volumetric object. (a) is based on a traditional z-test showing randomly dots caused by z-fighting. (b) is based on the proposed method removing the z-fighting artifact.

The proposed method assigns the virtual thickness to such plane surfaces to make the virtual zSlab, thereby providing a better visual smoothness so that the overlapping surfaces is well represented without such problematic artifacts shown in Fig.6.5 (b). Moreover, the proposed method is available to remove the flickering artifact caused by z-fighting during the animation. The z-fighting is also problematic when there is a iso-surface rendering with non-zero opacity clipping plane. As the clipping plane is separately rendered with the clipped iso-surface rendering, the plane is supposed to share a plane with the rendered iso-surface, thereby resulting in the z-fighting problem. Fig.6.6 shows the examples of this problem in the clipping scene.

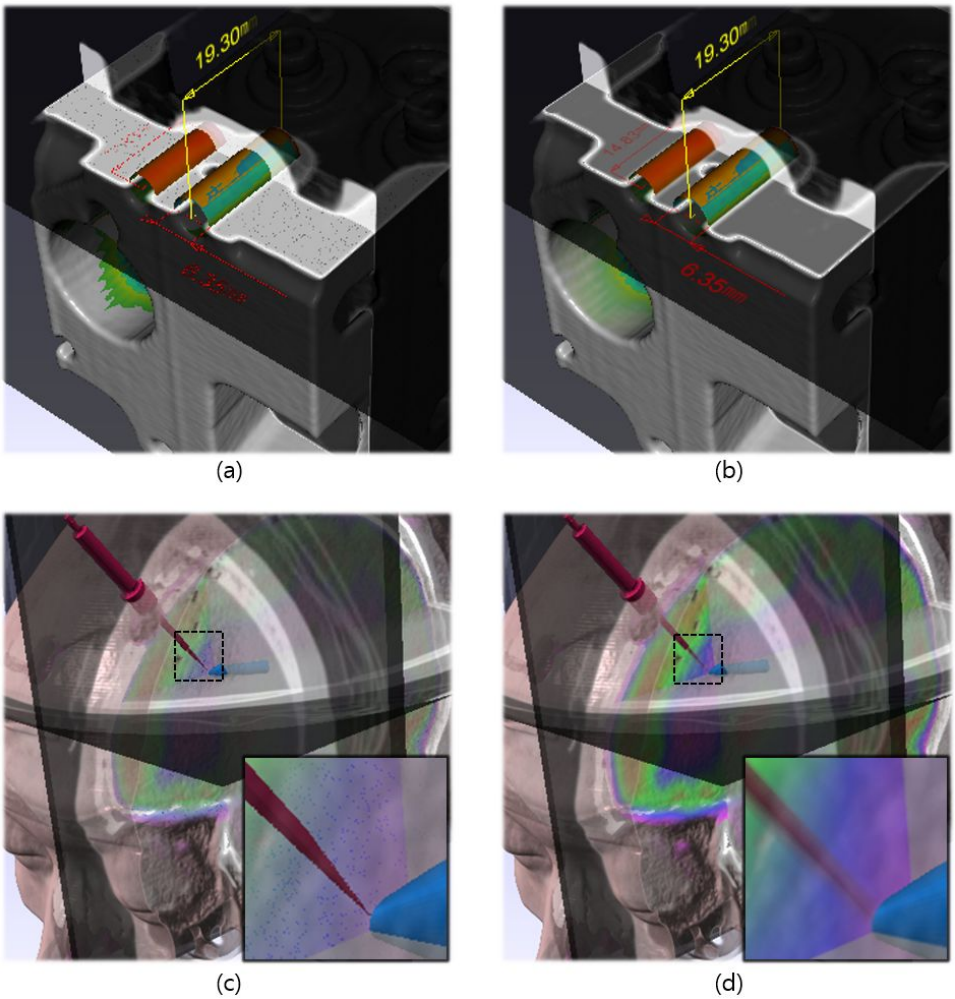


Figure 6.6. Comparison of rendering results visualizing the clipping planes and the artificial objects; the upper is from an industrial application for the coordinate measurement and the lower is from a medical application for the surgical simulation. The left side (a) and (c) show the z-fighting artifact and aliasing artifact in the overlapping and intersecting region, respectively. The right side (b) and (d) resolve such artifacts.

6.3. Transparent MDVR study

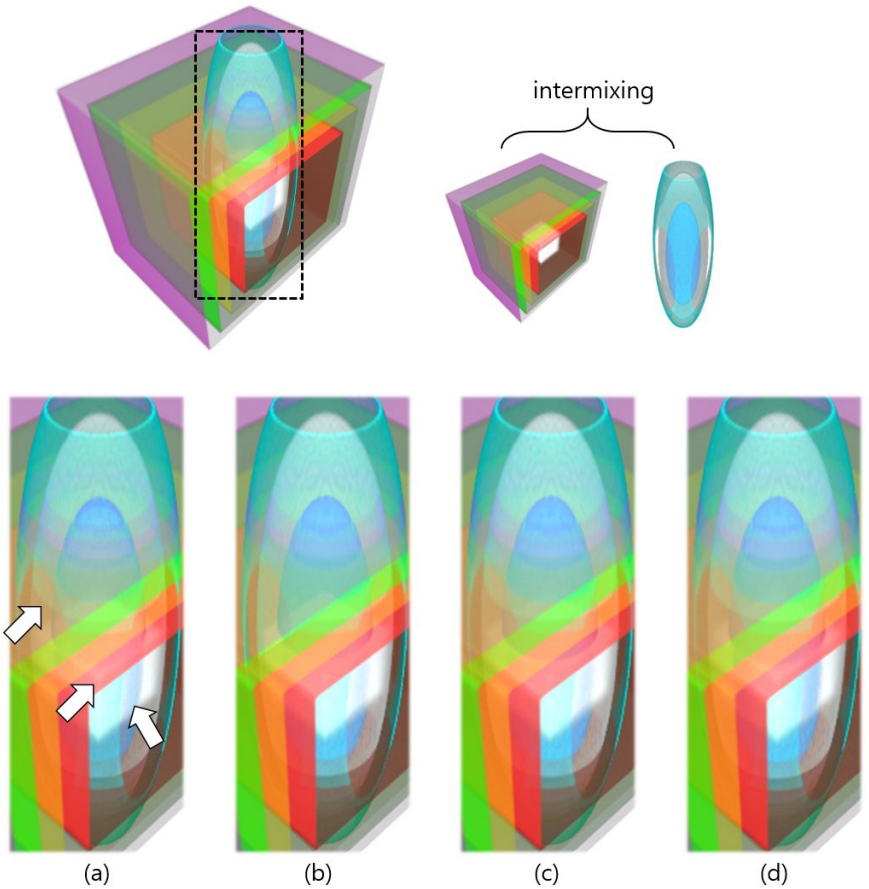


Figure 6.7. Comparison of rendering results of two transparent volumes composed of multiple thin surfaces according to the number of zSlab layers. (a) is a conventional MDVR image as a reference result (which correctly represents the entire transparent geometry). (b) and (c) are based on the full image-level intermixing with 4 and 8 zSlab layers, respectively. (d) is based on the hybrid-level intermixing with 8 zSlab layers. (c) and (d) show represent the entire transparent geometry as well as (a) does, whereas (b) shows some visibly incorrect geometry.

One notable thing is that most of visually distinguishable intersection should have enough opacity to recognize it. And this occurs in the first few structures before reaching such high-opacity that obstructs the visual distinguishability of intersection. This enables our intermixing method based on the proposed zSlab model (with OIT concept) to provide an acceptable rendering result even if there might be some information loss of intersections. Obviously, opaque volumes are well supported in our approach, thereby we have to prove practical multi-volume rendering that represents transparent structures (i.e., transparent multi-volume rendering).

In the proposed intermixing pipeline, there are two options for handling such the transparent multiple-volumes in the volume rendering stage (see. Fig.5.1 and Fig.5.4); full image-level intermixing or hybrid-level intermixing (i.e., intermixing for prior volumes and then on-the-fly intermixing during the ray-casting of the last volume) mentioned in Chapter 5.5. This approach enables faster rendering speed preserving acceptable rendering result compared to the conventional MDVR.

In the hybrid-level intermixing, the prior volumes rendering fulfills the image-level intermixing and the final volume rendering fulfills the on-the-fly intermixing. Therefore, it is necessary to handle a volume that represents complex transparent structures in the final volume rendering so that the more accurate intermixing result is possible compared to the full image-level intermixing preserving the advantages of the participating renderer's independency. Note that the complexity of the representing structures of a volume is determined by the opacity-transfer-function (OTF). Thus, when rendering the multiple volumes based on the hybrid-level

intermixing, it is necessary to consider each OTF state for the determination of the complexity of a volume structure.

Fig.6.7 shows the comparison result of multi-volume rendering. Fig.6.7 (a) is performed based on the conventional MDVR (based on the fully on-the-fly intermixing with super-sampling), which is a reference in this test. And, as the ellipsoid structure has smaller number of transparent layers, we choose the cube volume rendering as the final volume rendering supposed to run the on-the-fly intermixing (see Fig.6.7 (d)). Fig.6.7 (b) and (c) show the result of the full image-level intermixing with different number of zSlab layers in the z-thickness buffer. Of course, more number of zSlab layers enables more accurate representation of overlapping structures (see Fig. 6.7 (b) and (c)). As 8 zSlab layers are able to represent enough transparent and fuzzy surfaces in each test volume rendering, Fig.6.7 (c) well represents the entire mixing structures similar with the reference image, Fig.6.7 (a). Fig.6.7 (d) is the best quality image among our results based on the proposed approach using enough zSlab layers and on-the-fly intermixing during the last volume rendering (i.e., hybrid-level intermixing). This delivers high-fast rendering and acceptable image quality (without any noticeable quality difference) of the transparent multi-volume rendering.

To evaluate further the performance of the proposed OIT-based image-level intermixing of the transparent multi-volume rendering, the full image-level intermixing is performed on a scene that three different (e.g., different resolution, different orientation, or different modality) volumes participate in (see Fig.6.8). For the transparency of the volume rendering, we use the modulation method [49] to visualize the occluded features inside a volume. The right-hand in Fig.6.8 shows

that the full image-level intermixing based on the proposed zSlab model provides visually understandable result compared to the reference image by the conventional method (base on simultaneously multiple samplings along a single ray-casting).

Fig.6.18 shows a rendering speed table of the full image-level intermixing according to the number of zSlab layers that z-thickness buffer has, compared to the conventional MDVR. Obviously, the intermixing performance is strongly dependent on the number of zSlab layers. In both the methods, the early-ray-termination (ERT) is applied, but the empty-space-skipping (ESS) is applied only to the proposed method. As the test rendering is for the transparent volumes (which ERT is hard to occur), this leads to the rendering speed difference between the two test methods. (consider that the proposed method delivers faster rendering speed than the conventional method in the transparent multi-volume rendering in all test cases. See Fig.6.18) The emptier space and the larger size of the volume dataset, the more rendering speed difference between the methods (see Fig.6.18 automobile case).

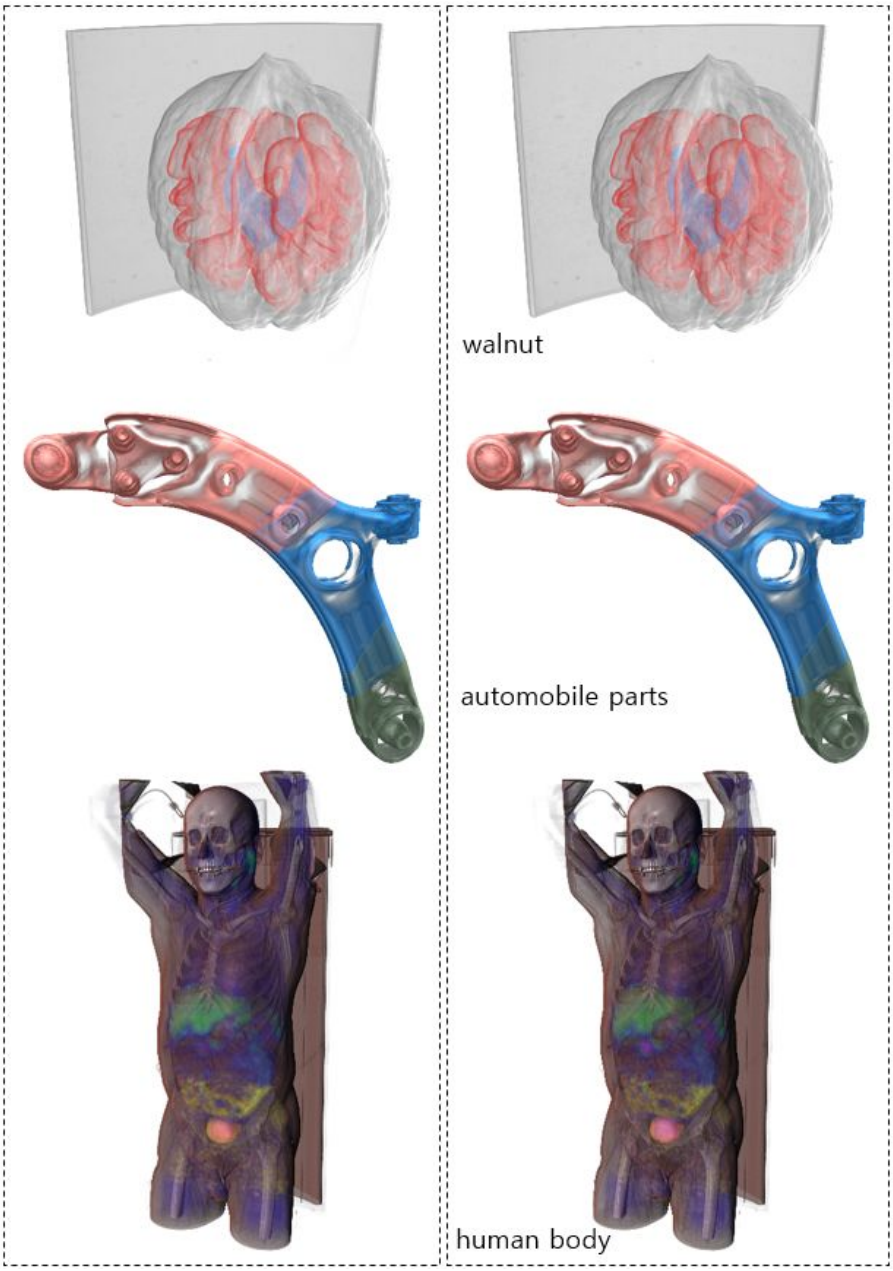


Figure 6.8. Comparison of transparent multi-volume rendering results; scientific, stitching, and multi-modal visualization. Left-hand images (reference) are from based on a conventional MDVR which performs accurate depth comparison. Right-hand images are from the proposed method, providing faster rendering with acceptable rendering accuracy.

6.4. Visibility interpolation study: original vs. modified

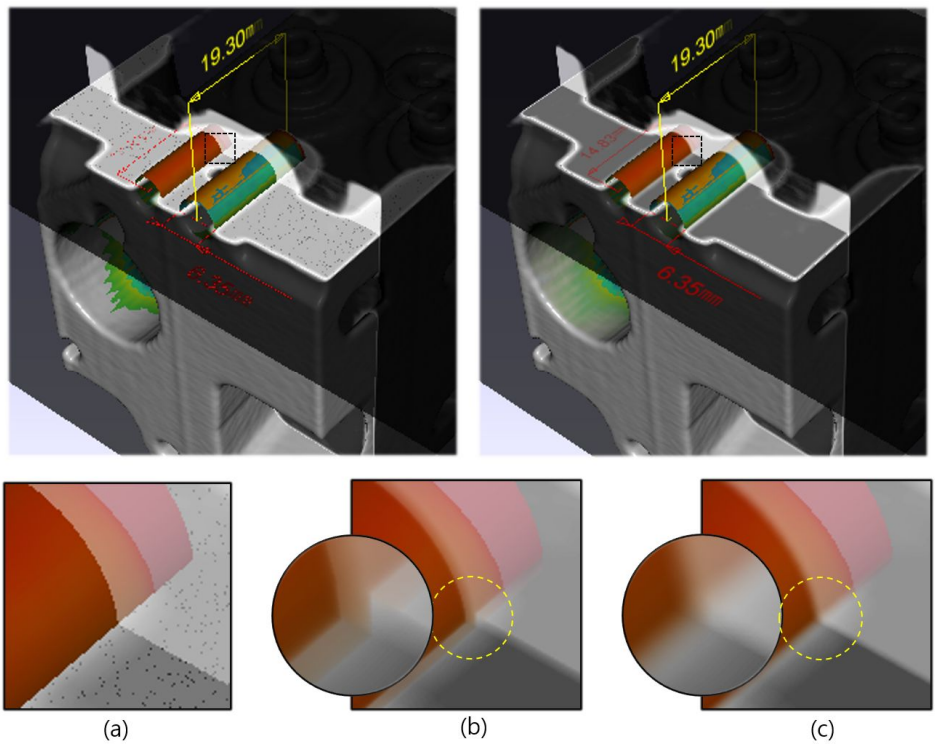


Figure 6.9. Comparison of rendering results of intersecting surfaces along with a clipping plane according to the in-slab visibility interpolation models. (a) is based on a traditional z-test, showing aliasing intersection and z-fighting artifact. (b) and (c) are based on the proposed method. (b) uses the original in-slab visibility interpolation, showing clear and smooth visibility transition but some visible bands. (c) uses the modified in-slab visibility interpolation, showing smoother visibility transition without any visible artifact.

In this dissertation, two versions of in-slab visibility interpolation are introduced to describe zSlab subdivision and its visibility representation. In the original version of in-slab visibility interpolation, the sub-zSlab's opacity gets close to the opaqueness if the zSlab is filled with a high-opacity medium (even if the sub-zSlab is thin enough). This makes the front zSlab blocks the visibility of the back sub-zSlab, thereby, the back sub-zSlabs' visibilities do not contribute the final visibility (even when its thickness is very small). As the visibility occlusion occurs discontinuously over the zSlab subdivision, this brings about discontinuously visible bands (as undesirable visibility error) shown in Fig.6.9 (b).

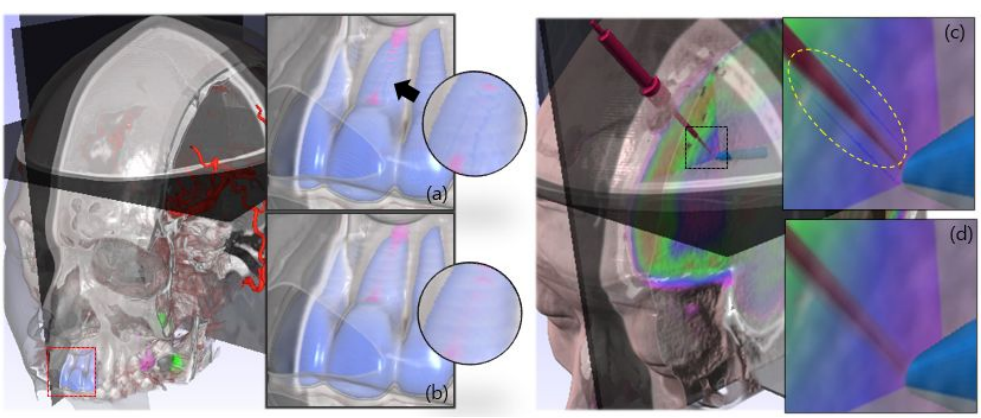


Figure 6.10. Comparison of rendering results showing the visible artifact of intersecting according to the in-slab visibility interpolation models. (a) and (c) are based on the original in-slab visibility interpolation. (b) and (d) are based on the modified in-slab visibility interpolation. The incorrect visible-bands are shown in (a) and (c) caused by zero-divided operation, whereas there is no such artifact in (b) and (d).

The modified version of in-slab visibility interpolation also has advantages in terms of implementation. Eq. (3.4), (3.5) and (3.6) are simple arithmetic operations without expensive exponential operation. And, since every element in a RGBA vector can be independently computed, the parallel operation of the computation is available. Moreover, those operations do not require a branch instruction for the check of zero-dividing, which brings about incorrect visible bands shown in Fig.6.10.

Fig.6.9 and Fig.6.10 are practical examples of Fig.3.2 and Fig.3.3 showing the rationality of the modified version of in-slab visibility interpolation.

6.5. Fusion rendering study for the hidden structures

By exploiting the proposed zSlab model, our intermixing approach can be extended to visualize occluded objects with opacity-derived depth cues as a fusion rendering, breaking general optics rule to provide more visual perception of the entire context.

6.5.1. Implicit attribute volume visualization

In many volume-based segmentation tools, the segmented part is represented

as also a volume, which is visualized with the original volume's context. For example, in industrial CT imaging, metal-foam dataset, encoding the size of porosity, can be derived from a scan dataset. The original CT data and its derived attribute data re-quire different sampling method and optimization strategy. In this case, the conventional MDVR should be implemented with complex branch instructions for integrating the different samplings and optimization techniques in one rendering pipeline.

However, in the proposed method, each volume can be independently rendered with its own optimized visualization technique, and then, two rendering images are intermixed. Fig.6.11 shows the rendering of the industrial CT scan data and its porosity data (as a segmented volume). In this figure, the CT scan (representing the silhouette) is sampled using the gradient magnitude modulation [50] and the porosity data is sampled using the close coverage filter for anti-aliasing [51], respectively, and then both data are intermixed. Note that the transparent structures do not need to be high-frequency feature that represents a detailed shape of surface. Thus, in this experiment, we use the CT scan down-sampled by 64 (4x4x4) and the porosity data in its original resolution, which also manifests that the advantage of our method of the rendering independency makes it easy to render two datasets even with different resolution and different sampling distance, preserving the details of the porosity shape with the surface refinement technique [45].

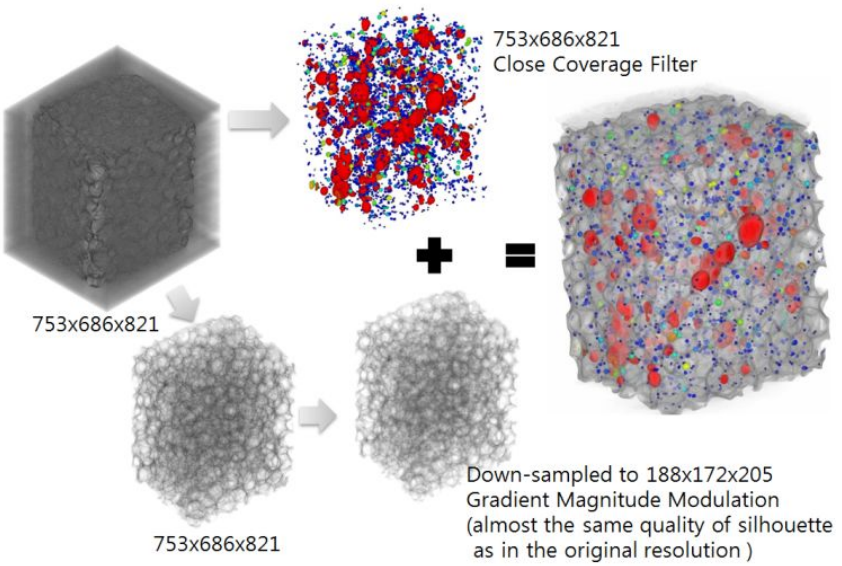


Figure 6.11. An example of the intermixing scene that needs different volume rendering techniques. A volume rendering that shows only the silhouette allows down-sampling to represent the entire context. A volume rendering that shows the segmented attribute volume needs high-precision sampling with a special filtering to show the details.

6.5.2. zSlab blending for visual-cues of hidden structures

When there are several objects in a scene and some important objects are hidden inside the occluding object, it is necessary to visualize the hidden internal-object with the occluding object. Clipping [52] and transparent setting of occlusion structure [49] are the familiar methods to visualize inhomogeneous hidden objects occluded by another object. However, both methods make the object to be

geometrically cut away or to be globally transparent in visualization, showing the hidden internal-objects. To avoid such undesirable missing part of the occluding object, blending operation between the occluding and occluded objects is generally adopted based on a single z-depth test in many applications (consider that the blending is performed only on the visually overlapping region).

By exploiting our virtual thickness and zSlab model, it is possible to visualize the internal object with the occluding object, preserving depth cue sorted in depth-order (referred as 'zSlab blending'). In the scene that several objects participate in and occlude other objects, the proposed method is able to well represent the occluding object as well as the occluded object, providing geometric depth cue and the hidden status of the object. Fig.6.12 shows an example scene that includes an occluding X-mas tree scanned by CT and 7 number of hidden segmented-objects that are a little bit smaller than the occluding part of CT data. With strict depth comparison, all of internal details are hidden (see Fig.6.12 (a)). Fig.6.12 (b) is just an overlay of the hidden structures over the occluding geometry. Fig.6.12 (c) and (d) are optics breaking model using z-depth shift and our proposed zSlab blending, respectively. In Fig.6.12 (d), our zSlab blending enables the local blending over the occluding part representing the occluding part as a thin layer, preserving original rendering of the other parts (even along a viewing direction), which may be important in the scene.

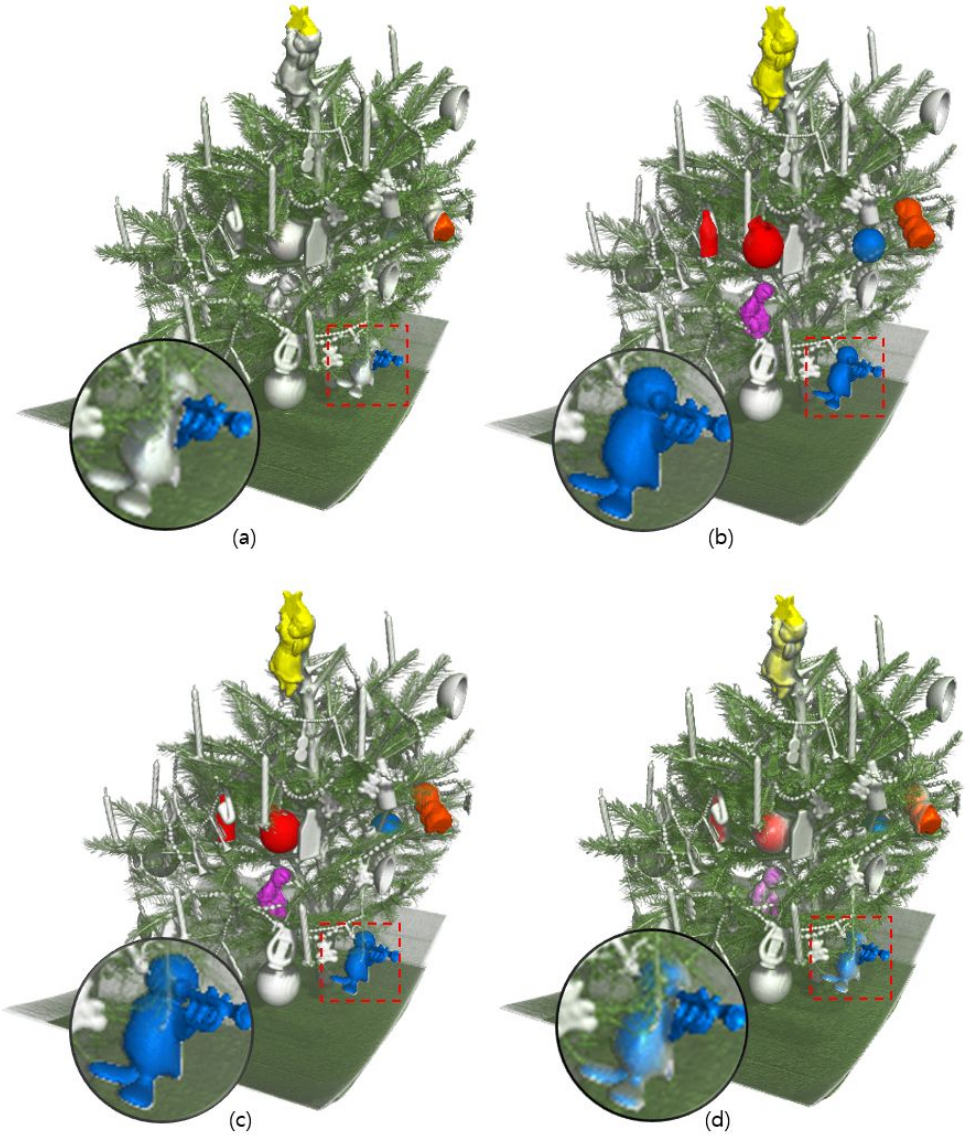


Figure 6.12. Comparison of rendering results for the visualization of the hidden structures according to the representation method of the hidden objects. (a) (b) and (c) are is without virtual thickness. (b) is an overlay image of each individual rendering result. (c) uses a z-depth shift forward to the camera. (d) uses the proposed virtual zSlab with finite length of virtual thickness (3 voxels).

Fig.6.13 shows an original CT object and its segmented objects, which were represented as polygonal objects. To show each segmented objects, which are occluded, Fig.6.13 (a) uses transparent DVR with on-the-fly intermixing method while Fig.6.13 (b) uses high opacity DVR with slab-based intermixing method based on the virtual zSlab with large enough thickness to penetrate the outmost surface as user defined value. This intermixing scheme is well suited for representation of occluded objects while providing the outmost structure without fully traversing of volume on ray-casting which brings severe sampling burden.

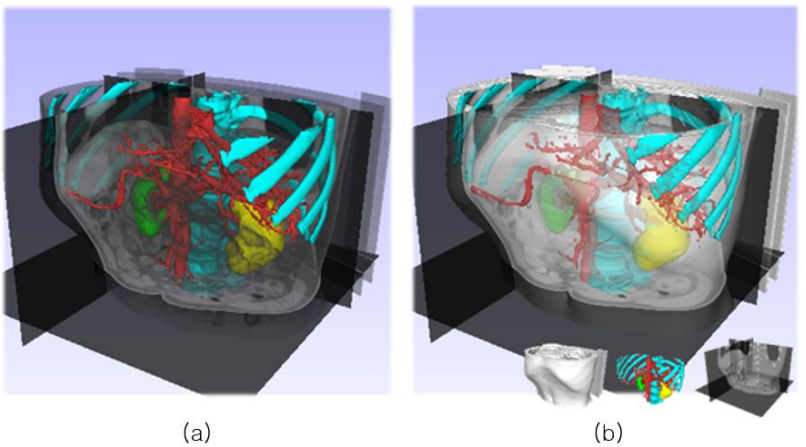


Figure 6.13 An intermixing example of the hidden or intra structures inside a volume. (a) is based on the on-the-fly intermixing during the transparent DVR. (b) is based on the full image-level intermixing with virtual zSlab model using user-defined thickness.

Fig.6.14 shows that our proposed method works well with a variety of transparency and thickness of virtual zSlab including transparent polygon geometry. The proposed virtual zSlab model is able to enhance the visual perception of overlapping surfaces while preserving main structure's depth cues (see Fig.6.12 and Fig.6.14 (b)).

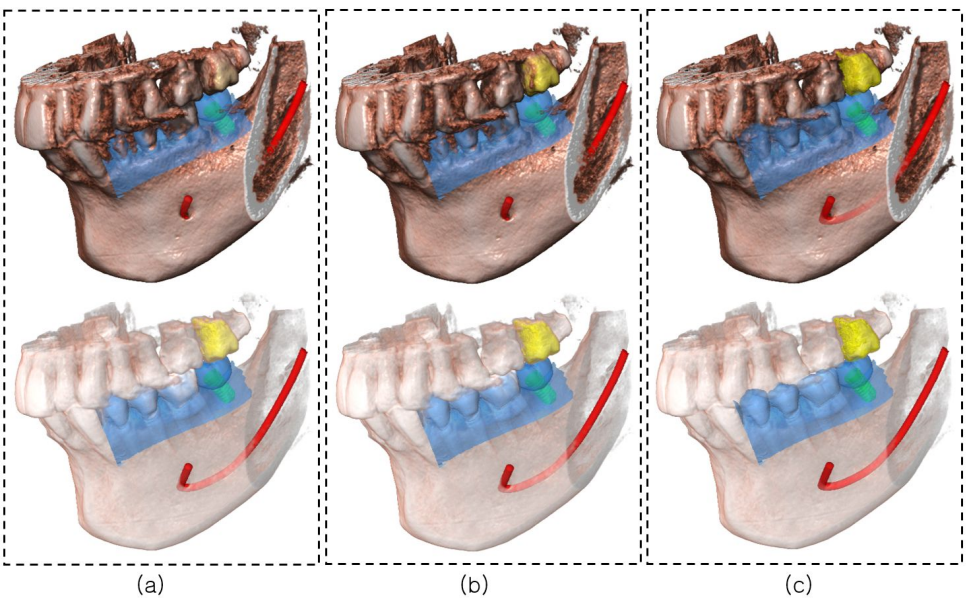


Figure 6.14. Comparison of rendering results of a dental CT and several polygonal objects with a variety of transparency and virtual thickness on the polygon surfaces. The upper shows high-opacity volume rendering and the lower shows transparent volume rendering. The virtual thickness of (a)(b)(c) : 1-voxel, 3-voxel, and 50-voxel, respectively.

6.6. Evaluation of the intermixing speed

For the evaluation of the intermixing efficiency (leads to intermixing speed), we measured the proposed rendering speed focusing on the overhead of the intermixing operations such as in-slab visibility interpolation and slab-based intermixing performed during zSlab traversing, and focusing on the overhead according to the number of zSlab layers. Intuitively, more intermixing operations and more number of zSlab layers, more performance burden in terms of rendering speed. In our intermixing pipeline, the read-back buffer operations and entire setting of meta information are the additional time-consuming work, which reflect the entire rendering speed, mainly dependent on the number of zSlab layers in the z-thickness buffer.

6.6.1. The overhead of the intermixing operations

For the evaluation of the overhead of the proposed intermixing operations, we measured the speed of multi-object rendering based on the hybrid-level intermixing, and the individual objects rendering speed computed by the summation of each individual object rendering time without intermixing. (thus, there is no operations related with the proposed intermixing operations.) In the hybrid-level intermixing, the in-slab visibility interpolation and slab-based intermixing operations occur at every sampling step during the last volume ray-casting, including the read-back

buffer and meta information setting operations.

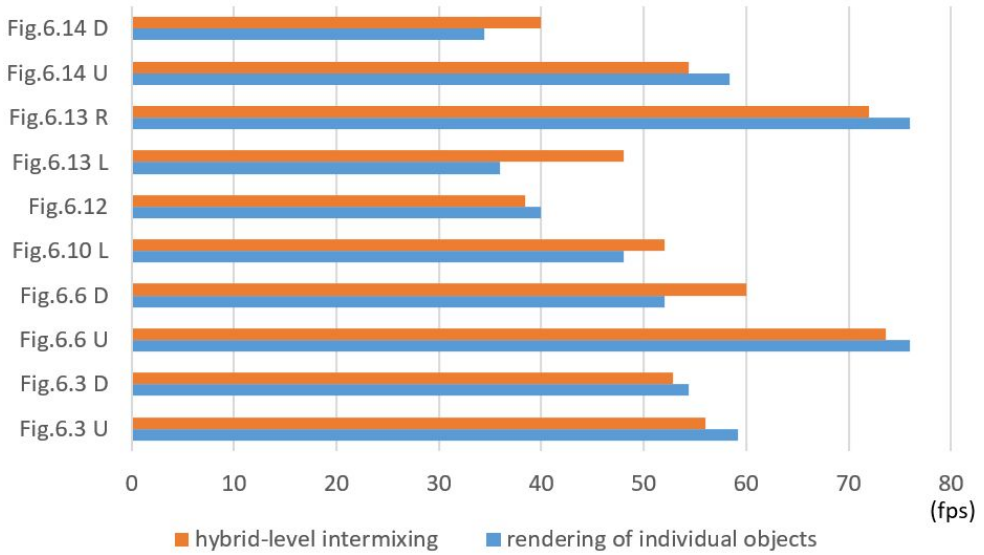


Figure 6.15 A chart depicting the rendering speed of the experimental cases, comparing between the individual objects rendering and the hybrid-level intermixing. The volume size used in each experimental cases is described in Table.6.1.

Fig.6.15 plots each rendering speed of such experiments that evaluate the overhead of the proposed intermixing operations, comparing to that of the individual objects rendering. As there is additional overhead of the intermixing operation overhead is not so critical, the speed of hybrid-level intermixing-based multi-object rendering is a little bit slower than that of individual objects rendering. Even if the intermixing operations cause additional overhead in the entire rendering process, the major time-consuming work is the volume rendering. Considering our experimental results, we can figure out that the proposed intermixing operations do

not bring severe overhead compared to the rendering of individual objects without intermixing operations. Even in most cases, because of ERT made up for our intermixing overhead by skipping shader process, some results (see the rendering speed of Fig.6.6D, 6.10L, 6.13L, 6.14D) show faster performance than that of the rendering of individual objects which has no occlusion by other objects.

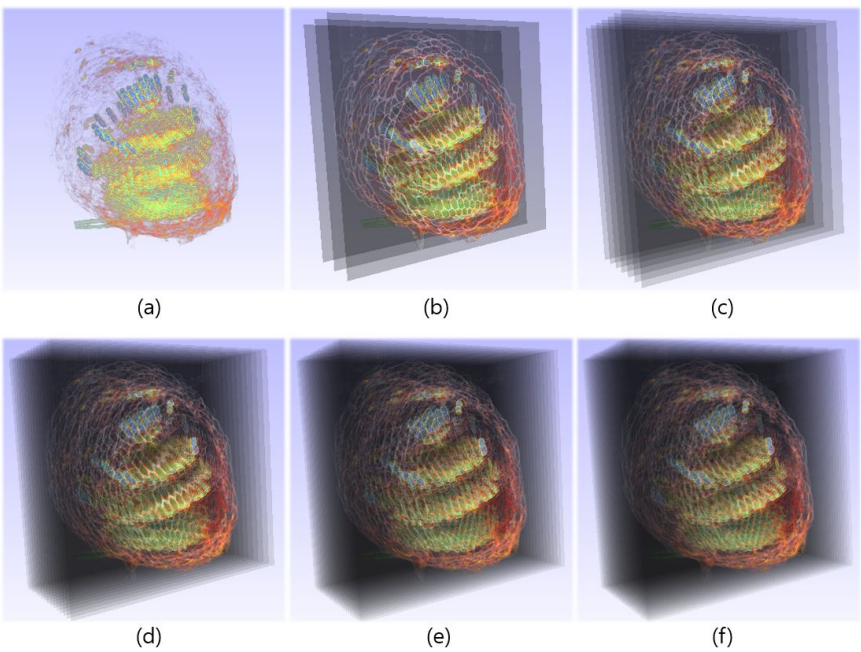


Figure 6.16 Rendering results of the transparent volume embedding superimposed planes, according to the number of the superimposed planes. The volume and planes have enough transparency so that the ERT does not occur during the ray-casting. The volume size is 600x600x600.

To experiment for our slab-based intermixing overhead in details, we modeled a scene including multiple transparent planes and a transparent volume, and a single pass renderer that handles the scene with an input zSlab array that includes the same number of zSlabs as the planes, thereby without read-back buffer and meta-information setting operations. The renderer fulfills the intermixing based on the on-the-fly approach. To do this, we used transparent superimposed MPR planes as an input zSlab arrays and highly transparent DVR for the full traverse of the entire volume, so that no ERT occurs in the intermixing process. Fig.6.16 depicts this experiment. Since there is no ERT, we can easily measure the time of slab-based intermixing operations by subtracting each individual objects rendering time.

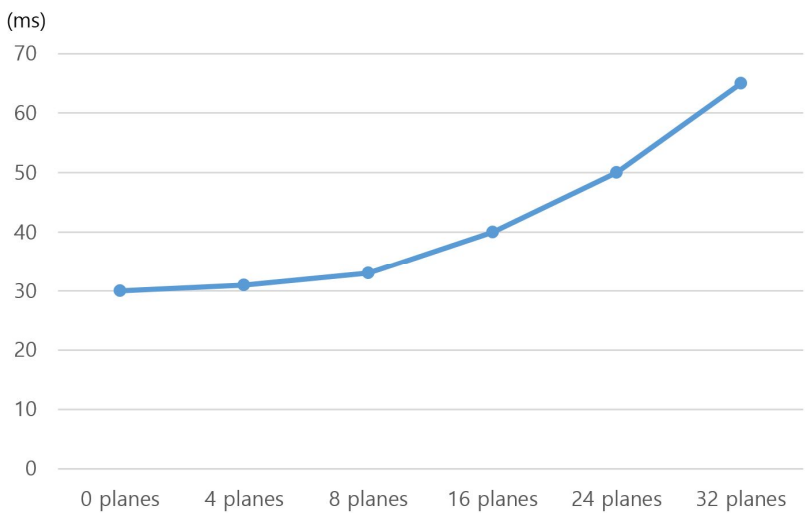


Figure 6.17 A plot showing the rendering time of Fig.6.16 (special version of hybrid-level intermixing) to measure the computational time of the proposed slab-based intermixing kernel function. The rendering time is measured according to the number of the participating planes which is same number as the zSlabs in an input array.

Fig.6.17 shows that the overhead of slab-based intermixing operations increases with the element number of zSlab array (i.e., superimposed MPR planes). When there are small element number of zSlab array (up to 8 planes in our experiment), we can ignore the overhead of slab-based intermixing. However, when there are more than ten elements in the zSlab array, the intermixing operations bring about considerable overhead because the operations work with a large set of texture memory, which causes low cache efficiency.

6.6.2. The overhead according to the number of zSlab layers

For the intermixing speed evaluation according to the number of zSlab layers in the z-thickness buffer, we measured the speed of multi-volume rendering based on the full image-level intermixing described in Fig.6.8 by changing the maximum number of zSlab layers. Fig.6.18 shows that the performance overhead increases with the number of zSlab layers in the z-thickness buffer, which leads to read-back operation overhead [47], more considerable overhead rather than the intermixing instruction overhead. More the maximum number of zSlab layers, more operational burden.

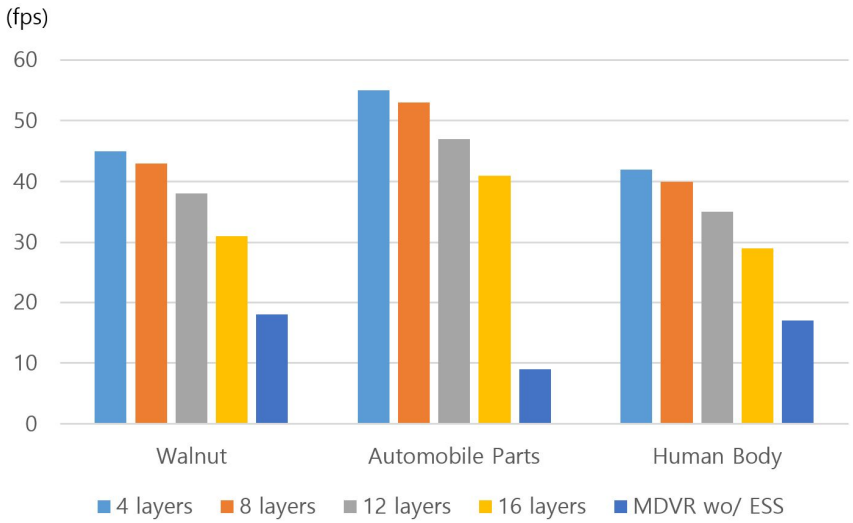


Figure 6.18 Plots for the speed of the transparent multi-volume rendering according to the number of zSlab layers (based on the full image-level intermixing), comparing to the conventional MDVR (without ESS). The rendering resolution is 1024x1024, and the volume size used in each experimental case is described in Table.6.1.

According to our performance test shown in Fig.6.18 (as well as Fig.6.17), 8 number of zSlab layers is the most efficient for the high performance intermixing; quality and speed, which is same result as the number of the OIT with bounded memory environment [18]. Considering the result images in Fig.6.8 and the rendering speed compared to that of the conventional MDVR, the proposed intermixing is still computationally efficient so that that most methods of multi-volume rendering based on our approach can show the interactiveness in terms of the rendering speed.

Table 6.1. Volume size and rendering type information used in the experimental results.

Figure	*Size (voxels)*	*rendering type*
Fig.6.3 upper	3 of 512x512x512	iso-surface rendering *full image-level intermixing
Fig.6.3 lower	2 of 460x460x738 1 of 432x432x589	iso-surface rendering *full image-level intermixing
Fig.6.6 upper	1 of 256x256x256	iso-surface rendering *full image-level intermixing
Fig.6.6 lower	2 of 512x512x512	transparent volume rendering **hybrid-level intermixing
Fig.6.8 upper	3 of 400x296x352	transparent volume rendering *full image-level intermixing
Fig.6.8 middle	same as Fig.6.3 lower	transparent volume rendering *full image-level intermixing
Fig.6.8 lower	1 of 512x512x738 1 of 128x128x243 1 of 168x168x437	transparent volume rendering *full image-level intermixing
Fig.6.10 left	1 of 512x512x512 2 of 300x300x300 3 of 50x50x50	transparent volume rendering **hybrid-level intermixing
Fig.6.12	1 of 512x512x512	normal volume rendering with shadow **hybrid-level intermixing
Fig.6.13	1 of 512x512x512	transparent volume rendering (left) **hybrid-level intermixing (left) normal volume rendering (right) *full image-level intermixing (right)
Fig.6.14	1 of 608x608x352	normal volume rendering (upper) transparent volume rendering (lower) **hybrid-level intermixing

6.7. Evaluation of memory usage

As our implementation is based on GPU, it is important to fit all the intermixing objects inside GPU's dedicated memory to avoid memory swapping, which leads to serious performance loss. By exploiting our deferred rendering approach which treats each intermixing object independently, there could be additional optimization for efficient storage such as uploading of only available volume blocks or volume sampling without inflating of low resolution volume. This keeps memory consumption at acceptable levels while preserving efficient and flexible implementation.

However, this requires additional memory since read-back buffer stores the previously rendered information defined with visibility and depth of each boundary. Each visibility and depth uses 4 bytes respectively (optionally, 20 bytes) per pixel. For a zSlab array, we commonly used 8 number of zSlabs of z-thickness buffer. The z-thickness buffer needs to be dual-allocated for the ping-pong strategy. For the representation of the transparent polygon surfaces, we used 4 number of RT-buffers and dual depth buffers as additional required memory. Therefore, the size of the entire buffers for 512x512 rendering resolution is 62 MB.

Chapter 7. Conclusion

In this dissertation, we proposed a novel intermixing scheme applying the OIT concept (which is typical technique that handles the transparency issues in real-time graphics, generally for complex polygonal-geometry) to the multi-volume rendering as well as the hybrid rendering of polygons and volumes. To do this, we presented an optical model which takes into account the proposed zSlab model that is able to represent fuzzy (or thickness) surface and a ray-segment compressing several boundaries or a wide range of slabs. As the conventional buffer is impossible or difficult to store such geometry, we introduced a special data structure, z-thickness buffer, employing visibility function and slab's geometry information. Along with our intermixing method based on the zSlab model and z-thickness buffer, we carefully designed the in-slab visibility interpolation for more continuous and smoother visibility transition that allows the advantages in visualization. Based on the proposed methods, we constructed an intermixing pipeline that allows the full image-level intermixing (focused on the intermixing speed) and the hybrid-level intermixing (complementing the intermixing accuracy), which are available even in a multi-volume rendering (of course, available in the hybrid rendering of polygon and volume objects).

This novel intermixing approach provides several advantages in terms of visualization, implementation and rendering speed. This improves the intermixing quality on overlapping and intersecting surfaces while maintaining real-time

performance, which can be an efficient solution to overcome aliasing intersection edges and z-fighting as conventional rendering problems of the intermixing scene of multiple objects. And, this enables to easily construct the entire rendering framework maintaining each renderer's optimization techniques, which can be a flexible and efficient framework basis that treats complex scenes including several objects with heterogeneous datasets, which need object-dependent rendering optimization. In addition, this provides special advantages in the case of intermixing with occluded objects by breaking the optics based on the virtual zSlab model.

When the proposed method is adopted into real applications, it exhibits additional outstanding advantages as demonstrated in the case studies. As our method is based on the deferred rendering approach (the intermixing of each rendering result), individual objects can be rendered independently by using their own optimized visualization technique without being forcefully integrated in a single rendering pipeline. Moreover, as our intermixing approach allows to store an object rendering result in a static z-thickness buffer, the zSlab information representing the rendered geometry can be reused as necessary. Those two advantages of the rendering independency and reusability are very profitable when building an interactive multiple-object rendering framework.

Since our proposed slab-based intermixing scheme is basically for two different zSlab arrays, we have to consider priority of rendering objects because the final visibility is not invariant if intermixing order of high opacity objects changes if there are two more intermixing units at the same position. It means higher opacity and prior rendered object contributes more to the final intermixing

visibility than lower and posterior ones do. Even with this limitation, our implementation scheme has a dominant advantage which enables flexible rendering pipeline and its simple implementation, also taking advantages of the image-level intermixing. Moreover, based on the proposed in-slab visibility interpolation, the instructions are efficiently implemented by using parallel instructions.

Obviously, there is additional intermixing overhead. But, our experiment results show that the additional instructions in the intermixing is the small enough to preserve the real-time rendering. Because current graphics hardware has high enough memory bandwidth to efficiently support the read-back operations, the number of rendering passes (with zSlabs up to 8, experimented in this dissertation) does not bring severe overhead to disturb real-time rendering as well.

In the future work, we might be able to develop smarter zSlab compression model for more proper intermixing result providing the acceptable accuracy with reasonable overhead, and for more effective tail-handing in the bounded memory condition. And, the proposed zSlab model and intermixing method are valid not only in the scientific graphics but also in the general real-time graphics for a variety of graphics effects (commonly for the volumetric effect).

Bibliography

[1] Manssour, Isabel, et al. "A framework to visualize and interact with multimodal medical images." Volume Graphics 2001. Springer Vienna, 2001.

[2] Hardenbergh, Jan, et al. "Integrated 3D Visualization of fMRI and DTI tractography." (2005): 94.

[3] Rößler, Friedemann, et al. "GPU-based Multi-Volume Rendering for the Visualization of Functional Brain Images." SimVis. Vol. 2006. 2006.

[4] Rieder, Christian, et al. "High-Quality Multimodal Volume Visualization of Intracerebral Pathological Tissue." VCBM. 2008.

[5] Tiede, Ulf, Thomas Schiemann, and Karl Heinz Höhne. "High quality rendering of attributed volume data." Visualization'98. Proceedings. IEEE, 1998.

[6] Kreeger, Kevin, and Arie Kaufman. "Mixing translucent polygons with volumes." Proceedings of the conference on Visualization'99: celebrating ten years. IEEE Computer Society Press, 1999.

[7] Levoy, Marc. "A hybrid ray tracer for rendering polygon and volume

data." Computer Graphics and Applications, IEEE 10.2 (1990): 33-40.

[8] Kainz, Bernhard, et al. "Ray casting of multiple volumetric datasets with polyhedral boundaries on manycore GPUs." ACM Transactions on Graphics (TOG). Vol. 28. No. 5. ACM, 2009.

[9] Kniss, Joe, et al. "Interactive translucent volume rendering and procedural modeling." Visualization, 2002. VIS 2002. IEEE. IEEE, 2002.

[10] Plate, John, Thorsten Holtkaemper, and Bernd Froehlich. "A flexible multi-volume shader framework for arbitrarily intersecting multi-resolution datasets." Visualization and Computer Graphics, IEEE Transactions on 13.6 (2007): 1584-1591.

[11] Lindholm, Stefan, et al. "Fused Multi-Volume DVR using Binary Space Partitioning." Computer Graphics Forum. Vol. 28. No. 3. Blackwell Publishing Ltd, 2009.

[12] Lux, Christopher, and Bernd Fröhlich. "GPU-based ray casting of multiple multi-resolution volume datasets." Advances in Visual Computing. Springer Berlin Heidelberg, 2009. 104-116.

[13] Grimm, Sören, et al. "Flexible direct multi-volume rendering in dynamic scenes." VMV. 2004.

[14] Rossler, F., Ralf P. Botchen, and Thomas Ertl. "Dynamic shader generation for flexible multi-volume visualization." Visualization

Symposium, 2008. PacificVIS'08. IEEE Pacific. IEEE, 2008.

[15] Max, Nelson. "Optical models for direct volume rendering." Visualization and Computer Graphics, IEEE Transactions on 1.2 (1995): 99-108.

[16] Hadwiger, Markus, et al. "Advanced illumination techniques for GPU volume raycasting." ACM Siggraph Asia 2008 Courses. ACM, 2008.

[17] Barta, Pál, and Balázs Kovács. "Order Independent Transparency with Per-Pixel Linked Lists." Budapest University of Technology and Economics (2011).

[18] Kubisch, C. "Order independent transparency in OpenGL 4. x." Proceedings of the 2014 GPU Technology Conference. Vol. 3. 2014.

[19] Everitt, Cass. "Interactive order-independent transparency." White paper, nVIDIA 2.6 (2001): 7.

[20] Maule, Marilena, et al. "A survey of raster-based transparency techniques." Computers & Graphics 35.6 (2011): 1023-1034.

[21] Zhenzhen, G., N. Luciano, and N. Ulrich. An Interactive Hybrid Renderer using Multi-Layer Depth Maps. technical report. http://www. cs. usc. edu/research/11-922. pdf.

[22] McGuire, Morgan, and Louis Bavoil. "Weighted blended order-independent transparency." Journal of Computer Graphics Techniques (JCGT) 2.2 (2013).

[23] Bernardini, Fausto, and Holly Rushmeier. "The 3D model acquisition

pipeline." Computer graphics forum. Vol. 21. No. 2. Blackwell Publishers Ltd, 2002.

[24] Jacq, Jean-José, and Christian J. Roux. "A direct multi-volume rendering method aiming at comparisons of 3-D images and models." Information Technology in Biomedicine, IEEE Transactions on 1.1 (1997): 30-43.

[25] Ferre, Maria, Anna Puig, and Dani Tost. "Rendering techniques for multimodal data." Proc. SIACG 2002 1st Ibero-American Symposium on Computer Graphics. 2002.

[26] Wilson, Brett, Eric B. Lum, and Kwan-Liu Ma. "Interactive multi-volume visualization." Computational Science—ICCS 2002. Springer Berlin Heidelberg, 2002. 102-110.

[27] Cai, Wenli, and Georgios Sakas. "Data Intermixing and Multi-volume Rendering." Computer Graphics Forum. Vol. 18. No. 3. Blackwell Publishers Ltd, 1999.

[28] Brecheisen, Ralph, et al. "Flexible GPU-Based Multi-Volume Ray-Casting." VMV. 2008.

[29] Salvi, Marco, and Karthik Vaidyanathan. "Multi-layer alpha blending." Proceedings of the 18th meeting of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games. ACM, 2014.

[30] J. Blinn. “Jim blinn’s corner - compositing, part i: Theory”, IEEE

Computer Graphics and Applications, Vol. 14, No. 5, pp. 83-87, 1994.

[31] Wittenbrink, Craig M., Thomas Malzbender, and Michael E. Goss. "Opacity-weighted color interpolation, for volume sampling." Proceedings of the 1998 IEEE symposium on Volume visualization. ACM, 1998.

[32] Kruger, Jens, and Rüdiger Westermann. "Acceleration techniques for GPU-based volume rendering." Proceedings of the 14th IEEE Visualization 2003 (VIS'03). IEEE Computer Society, 2003.

[33] Kindlmann, Gordon, and James W. Durkin. "Semi-automatic generation of transfer functions for direct volume rendering." Proceedings of the 1998 IEEE symposium on Volume visualization. ACM, 1998.

[34] Lokovic, Tom, and Eric Veach. "Deep shadow maps." Proceedings of the 27th annual conference on Computer graphics and interactive techniques. ACM Press/Addison-Wesley Publishing Co., 2000.

[35] Hadwiger, Markus, et al. "GPU-accelerated deep shadow maps for direct volume rendering." SIGGRAPH/EUROGRAPHICS Conference On Graphics Hardware: Proceedings of the 21 st ACM SIGGRAPH/Eurographics symposium on Graphics hardware: Vienna, Austria. Vol. 3. No. 04. 2006.

[36] Wikipedia. "z-fighting" https://en.wikipedia.org/wiki/Z-fighting

[37] Carpenter, Loren. "The A-buffer, an antialiased hidden surface

method." ACM Siggraph Computer Graphics 18.3 (1984): 103-108.

[38] Yang, Jason C., et al. "Real-Time Concurrent Linked List Construction on the GPU." Computer Graphics Forum. Vol. 29. No. 4. Blackwell Publishing Ltd, 2010.

[39] Engel, Klaus, Martin Kraus, and Thomas Ertl. "High-quality pre-integrated volume rendering using hardware-accelerated pixel shading." Proceedings of the ACM SIGGRAPH / EURO GRAPHICS workshop on Graphics hardware. ACM, 2001.

[40] Zhang, Yanci, and Renato Pajarola. "Deferred blending: Image composition for single-pass point rendering." Computers & Graphics 31.2 (2007): 175-189.

[41] Fuchs, Raphael, and Helwig Hauser. "Visualization of Multi-Variate Scientific Data." Computer Graphics Forum. Vol. 28. No. 6. Blackwell Publishing Ltd, 2009.

[42] Vannier, Michael W., Jeffrey L. Marsh, and James O. Warren. "Three dimensional CT reconstruction images for craniofacial surgical planning and evaluation." Radiology 150.1 (1984): 179-184.

[43] Knowles, Pyarelal, Geoff Leach, and Fabio Zambetta. "Efficient layered fragment buffer techniques." (2012): 279-292.

[44] Bavoil, Louis, and Kevin Myers. "Order independent transparency with dual depth peeling." NVIDIA OpenGL SDK (2008): 1-12.

[45] Hadwiger, Markus, et al. "Real-Time Ray-Casting and Advanced

Shading of Discrete Isosurfaces." Computer Graphics Forum. Vol. 24. No. 3. Blackwell Publishing, Inc, 2005.

[46] Bosma, Marco K., Jaap Smit, and Steven Lobregt. "Iso-surface volume rendering." Medical Imaging'98. International Society for Optics and Photonics, 1998.

[47] Liu, Fang, et al. "FreePipe: a programmable parallel rendering architecture for efficient multi-fragment effects." Proceedings of the 2010 ACM SIGGRAPH symposium on Interactive 3D Graphics and Games. ACM, 2010.

[48] Tsao, Nai-Kuan, and Franklin F. Kuo. "On machine precision, computation error and condition number in solving linear algebraic systems." Computers & Electrical Engineering 1.3 (1973): 459-464.

[49] Bruckner, Stefan, et al. "Illustrative context-preserving exploration of volume data." Visualization and Computer Graphics, IEEE Transactions on 12.6 (2006): 1559-1569.

[50] Bruckner, Stefan, and M. Eduard Gröller. Volumeshop: An interactive system for direct volume illustration. IEEE, 2005.

[51] Reeves, William T., David H. Salesin, and Robert L. Cook. "Rendering antialiased shadows with depth maps." ACM Siggraph Computer Graphics. Vol. 21. No. 4. ACM, 1987.

[52] Weiskopf, Daniel, Klaus Engel, and Thomas Ertl. "Interactive clipping techniques for texture-based volume visualization and volume

shading." Visualization and Computer Graphics, IEEE Transactions on 9.3 (2003): 298-312.

[53] Rezk-Salama, Christoph, et al. "Interactive volume on standard PC graphics hardware using multi-textures and multi-stage rasterization." Proceedings of the ACM SIGGRAPH/EUROGRAPHICS workshop on Graphics hardware. ACM, 2000.

[54] Gunther, Johannes, et al. "Realtime ray tracing on GPU with BVH-based packet traversal." 2007 IEEE Symposium on Interactive Ray Tracing. IEEE, 2007.

[55] Rushmeier, Holly, Anthony Hamins, and Mun Young Choi. "Volume rendering of pool fire data." IEEE Computer Graphics and Applications 15.4 (1995): 62-67.

[56] Sanz-Pastor, Nacho, and Luis A. Barcena. "Volumetric three-dimensional fog rendering technique." U.S. Patent No. 6,268,861. 31 Jul. 2001.

[57] Bozorgi, Mohammadmehdi, and Frank Lindseth. "GPU-based multi-volume ray casting within VTK for medical applications." International journal of computer assisted radiology and surgery 10.3 (2015): 293-300..

[58] Lawonn, Kai, et al. "Illustrative multi-volume rendering for PET/CT scans." Proceedings of the Eurographics Workshop on Visual Computing for Biology and Medicine. Eurographics Association,

2015.

[59] Lu, Cai, et al. "Application of Multi-attributes Fused Volume Rendering Techniques in 3D Seismic Interpretation." 2014 SEG Annual Meeting. Society of Exploration Geophysicists, 2014.

[60] Liu, Baoquan, et al. "Octree rasterization: accelerating high-quality out-of-core GPU volume rendering." IEEE transactions on visualization and computer graphics 19.10 (2013): 1732-1745.

초 록

이종의 다중 객체 혼합 가시화에 대한 요구는 최근 여러 가시화 어플리케이션에서 요구되어 왔다. 그러나, 객체를 정의하는 이종의 특성들로 인해, 여러 객체를 혼합하는 고성능 렌더링은 여전히 많은 연산과 구현의 복잡성을 갖는다. 본 논문에서는 다중 볼륨 및 폴리곤 객체의 효율적인 혼합 가시화를 위한 새로운 유형의 다중 객체 렌더링 방법을 제안한다. 폴리곤과 볼륨으로 정의되는 표면들을 다중 레이어로 표현하기 위해, 슬랩 모델을 기반으로 하는 새로운 자료 구조인 zSlab 을 제안한다. zSlab 은 색상과 깊이 그리고 슬랩의 두께 값을 저장하며, 가시화 방향에 대한 확산광 모델을 기반으로 슬랩 내부의 가시화 수치를 보간하여 연산하는 모델을 제안하며, 이를 혼합 기법의 단위 요소로 사용한다.

제안하는 다중 객체의 렌더링에서 zSlab 모델은 순서가 정해져 있지 않은 기하 정보를 다루며, 이것은 비정렬투명도 기술 (Order-Independent-Transparency) 이 다루는 모델과 유사하다. 본 논문에서는 이를 바탕으로 비정렬투명도 기술을 zSlab 모델에 확장 적용하고, 기존의 비정렬투명도 기술이 정의하지 못하는 불명확 경계 표면을 표현할 수 있도록 모델을 설계한다. 이것은 zSlab 기반의 혼합 기법과 함께, 영상 기반의 혼합 가시화를 다양한 다중 객체에 대해 가능하게 하여, 기존의 폴리곤 객체 간 혼합뿐만 아니라, 폴리곤 및 볼륨의 이종 객체 혼합과 다중 볼륨 객체의 혼합에까지 영상 기반의 혼합 가시화를 확장 적용할 수 있도록 한다.

먼저, 본 논문에서는 zSlab 이 불명확 경계를 어떻게 표현할 수 있는지를 기술하고, 제안한 zSlab 을 렌더링 과정에서 저장하는 새로운 자료 구조인 z-thickness 버퍼를 소개한다. 다음으로, zSlab 기반의 혼합

기법을 기술하기 위해, 기하학적 경계 분할로 인한 분할 슬랩의 가시화를 정의해야 하며, 이 기술을 가시성 보간 기법으로 소개한다. 마지막으로, 이러한 기반 기술을 바탕으로, 슬랩 기반의 혼합 방법을 설명하고 이것이 영상 기반 혼합 가시화에 적용되는 방법을 기술한다.

본 논문에서 진행한 실험 결과들은 제안한 방법을 기반으로 하는 영상 기반 혼합 가시화가 이종의 다중 객체 혼합 렌더링에서 고성능으로으로 동작함을 입증하였다. 기존의 고차원 연산 기반 혼합 기법을 통한 영상과 비교를 통해, 유효한 영상이 도출됨을 확인하였으며, 혼합 경계 영역에서 발생하는 계단 현상이나 z-fighting 문제를 효과적으로 처리함을 확인하였다. 또한, 실험 결과는 다중 객체 간의 물리적 혼합 가시화뿐만 아니라, 숨겨진 객체의 가시성을 확보하는 방법론에 잠재적으로 활용될 수 있음을 입증하였다.

주요어: 다중 객체 가시화, zSlab, 비정렬투명도 기술,

가시성 분할 기법, 영상 기반 혼합 렌더링

학번: 2008-30868