



UNIVERSITY OF BREMEN

FACULTY 03: MATHEMATICS AND COMPUTER SCIENCE

INSTITUTE OF COMPUTER GRAPHICS AND VIRTUAL REALITY

Master Thesis

Ray-Marching-Based Volume Rendering of Computed Tomography Data in a Game Engine

Kai-Ching Chang

Matriculation No:	3125983
Primary Examiner:	Prof. Dr. Gabriel Zachmann
Secondary Examiner:	Prof. Dr.-Ing. Udo Frese
Submission Date:	March 2020

Diese Erklärungen sind in jedes Exemplar der Bachelor- bzw. Masterarbeit mit einzubinden.

Name: _____ Matr.Nr.: _____

Urheberrechtliche Erklärung (§10 (11) Allgemeiner Teil der BPO/MPO v. 27.01.2010 (in der jeweils gültigen Fassung))

Hiermit versichere ich, dass ich die vorliegende Arbeit selbstständig verfasst und keine als die angegebenen Quellen und Hilfsmittel verwendet habe.

Alle Stellen, die ich wörtlich oder sinngemäß aus anderen Werken entnommen habe, habe ich unter Angabe der Quellen als solche kenntlich gemacht.

Die Bachelor-/Masterarbeit darf nach Abgabe nicht mehr verändert werden.

Datum

Unterschrift

Erklärung zur Veröffentlichung von Abschlussarbeiten

Die Abschlussarbeit wird zwei Jahre nach Studienabschluss dem Archiv der Universität Bremen zur dauerhaften Archivierung angeboten.

Archiviert werden:

- 1) Masterarbeiten mit lokalem oder regionalem Bezug sowie pro Studienfach und Studienjahr 10 % aller Abschlussarbeiten
- 2) Bachelorarbeiten des jeweils ersten und letzten Bachelorabschlusses pro Studienfach und Jahr.

Bitte auswählen und ankreuzen:

- ☐ Ich bin damit einverstanden, dass meine Abschlussarbeit im Universitätsarchiv für wissenschaftliche Zwecke von Dritten eingesehen werden darf.
- ☐ Ich bin damit einverstanden, dass meine Abschlussarbeit nach 30 Jahren (gem. §7 Abs. 2 BremArchivG) im Universitätsarchiv für wissenschaftliche Zwecke von Dritten eingesehen werden darf.
- ☐ Ich bin nicht damit einverstanden, dass meine Abschlussarbeit im Universitätsarchiv für wissenschaftliche Zwecke von Dritten eingesehen werden darf.

Datum

Unterschrift

Acknowledgments

This thesis is completed with the help of many people.

I want to thank my supervisor, Prof. Gabriel Zachmann, for giving me the opportunity to challenge myself. When I struggled with problems, he kindly arranged the meeting with people who are work in Mevis, and he always patiently gave me feedback during the meeting. And Prof. Udo Frese as the second reader for my thesis, we only met once, but your valuable suggestion solved the doubt that bothered me a long time.

For all the support and patience, during this long-term journey, Roland Fischer gave me a lot of suggestions on researching and writing, invested time to explain everything, encouraged me to join group meetings. I learned a lot. Sincerely thank you.

In the last, special thanks to my friends Meng-Yu Jennifer Kuo, A-Lin, and Emma took their spare time to read and endured my poor English writing.

需要感謝的人太多了，就感謝天罷！

Contents

Acknowledgments	iv
1 Introduction	1
1.1 Motivation	2
1.2 Goals of the Thesis	3
1.3 Thesis Structure	4
2 Fundamentals	5
2.1 Volume Rendering Techniques	5
2.1.1 Ray-Casting & Shear-warp	6
2.1.2 Optical Model	7
2.1.3 Transfer Functions	9
2.1.4 Composition Schemes	10
2.1.5 Ray-Casting Optimization Methods	11
2.2 Standard Illumination and Shadowing	12
2.3 Rendering in Unreal Engine 4	13
2.4 Overview of Digital Imaging Communications in Medicine	14
2.4.1 Analysis of Computed Tomography Data	15
2.4.2 Hounsfield Scale and Windows	16
3 Concept	18
3.1 Challenge	18
3.2 Approach for Obtaining Volumetric Data	19
3.3 Shader and 3D Texture	20

3.4	Algorithmn	21
3.4.1	Ray-Marching-Based Volume Rendering	21
3.4.2	Local Illumination and Shadow	22
3.4.3	Artefacts	23
3.4.4	Sparse Voxel Octrees	25
4	Implementation	27
4.1	Pre-procsssing - Blended Window	27
4.1.1	Pre-computing Transfer Function of Alpha	31
4.2	Custom Ray-Marching	32
4.2.1	Visual Enhancement	36
4.2.2	Branching Prevention	38
4.3	Octree	39
4.3.1	Tree Construction	39
4.3.2	Traversal	40
5	Results	46
5.1	Visualization	46
5.1.1	Ground Truth	46
5.1.2	Volume Rendering for Art versus Scientific Data	51
5.1.3	Different Illumination Result	52
5.1.4	Removed Artifacts	53
5.2	Performance	55
5.2.1	High-quality Illumination versus Performance	55
5.2.2	Octree	57
6	Conclusion & FutureWork	62
6.1	Conclusion	62
6.2	Future Work	63
	List of Figures	65

Contents

List of Tables	67
Bibliography	68

Glossary

BP* Illumination method enables a shadow map and the Blinn-Phong model.. 36, 52, 53, 56, 57, 59, 60

LAO* Illumination method enables a shadow map, the Blinn-Phong model and local ambient occlusion.. 36, 46–53, 56–61, 63

p01_04 Data ID: patient01_04.. 46, 49, 59, 66

p02_360 Data ID: patient02_360.. 46, 48, 51–53, 59, 65, 66

p03_788 Data ID: patient03_788.. 46, 50, 54, 59, 66

p03_78e Data ID: patient03_78e.. 46, 47, 53, 54, 59, 65

p05_f21 Data ID: patient05_f21.. 46, 60, 63

p06_12f9 Data ID: patient06_12f9.. 46, 60

p07_16c1 Data ID: patient07_16c1.. 46, 60, 63

p08_1b14 Data ID: patient08_1b14.. 46, 59

p09_24d Data ID: patient09_24d.. 46, 60

p09_24f Data ID: patient09_24f.. 46, 60

RB* Ray-Marching-Based DVR implementation in UE4 for general volumetric effects by Ryan Brucks [1].. 51, 52, 55–57, 59, 60, 62, 63, 66

Region of Interest A Region of Interest (ROI) is a subset of an image identified for a certain purpose. In medical imaging, ROI, for example, can be the purpose of measuring tumor size.. 1, 2

SM* Illumination method enables a shadow map.. 36, 52, 53, 56, 57, 59, 60

Transfer Function In volume rendering, a Transfer Function (TF) assigns emission and absorption coefficients to each specific scalar value.. 1

Acronyms

BONO Branch-on-Need Octree. 26, 39

CT Computed Tomography. 1, 3, 15–17, 20, 24, 25, 28, 33, 39, 62

DICOM Digital Imaging Communications in Medicine. 14–16, 19, 27, 28, 46, 52, 62, 63, 65

DVR Direct Volume Rendering. 1, 3, 5–7, 9, 18, 19, 21, 31, 55, 62

HLSL High-Level Shading Language. 13, 14

HU Hounsfield Unit. 16, 19, 20, 28, 29, 31, 32, 64

IDVR Iso-surface(Indirect) Volume Rendering. 1, 5, 6

LAO Local Ambient Occlusion. 22, 36, 62, 63

LUT Look-Up Table. 24

MC Marching Cubes. 5, 6

MIP maximum intensity projection. 51, 63

ODE Ordinary Differential Equation. 8, 9

PS Pixel Shader. 14, 19–21, 25, 56

RC Ray Casting. 6, 7, 10, 11, 21

RHI Render Hardware Interface. 13

RM Ray Marching. 18, 21, 22, 24, 25, 34, 37

ROI Region of Interest. ix, 2, 3, 47–50, 53, 62–64

RT Rendering Target. 13, 53

SSAA Supersampling anti-aliasing. 37, 55–62

TF Transfer Function. ix, 1–3, 6, 9, 10, 21, 22, 24, 27, 31, 35, 36, 52, 56, 62–64

UE4 Unreal Engine 4. 3, 14, 18, 19, 21, 25, 31, 40, 43, 52, 59, 62, 63

VRT Volume Rendering Technique. 1–5, 7, 18, 22, 23, 25, 27, 32, 35, 38, 52, 63, 64

VS Vertex Shader. 14, 20

VTK The Visualization Toolkit. 3, 63

1 Introduction

Computed Tomography (CT) is a widely used medical device in clinical practice. It is a technique using X-ray to produce cross-sectional images of piecewise of the human body. The main technique is illuminating the human body through a single axial X-ray rotation. Because different biological tissues have different X-ray absorption(or radiodensity), X-ray will partially transmit through the human body and be captured by the detector. Different absorption coefficients are converted into different grayscale CT images to display [2]. As a non-invasive method, CT scanning can distinguish smaller differences in tissue density than conventional X-ray, which decreases the use of cerebral angiography and pneumography. In 1973, the first images of direct visualization of cerebral infarction were published [3], which revealed the promise of CT scanning. With the evolution of CT(e.g., spiral CT and CT angiography), it benefited the assessment of disease and anatomy [4]. Because human eyes have capable of missing some important information, with the aid of computer science, the Region of Interest can be segmented from the medical image [5], [6].

Though computer science is fundamental for radiology, not until the 1990s, the improved CT scan combined with 3D technology [7] was applied in the realm of treatment planning. With visual capacities expanding, treatment has become intuitive and now is used as the virtual endoscopy [8] before surgery.

3D reconstruction inherently volumetric data are called Volume Rendering Technique (VRT), which is mainly divided into Iso-surface(Indirect) Volume Rendering (IDVR) and Direct Volume Rendering (DVR). The detailed literature review of IDVR and DVR will be compared in the next chapter. DVR observes the data directly from the original data set, which provides the algorithm with the opportunity to modify the Transfer Function (TF) dynamically. Some methods allow the internal structure of the dataset to be visualized in

a semi-transparent way. Ray-casting [9, 10] is most widely used for large-scale volumetric data. The algorithm was done by simulating light transporting through space then projecting on the screen. Transferring the numerical data into optical properties[11], such as color and opacity, can guide the user to focus on a specific subset of the volumetric data.

VRT is also widely applied in the art field to enhance visual effects [12]. Many visual effects in nature are irregular, such as fluids, clouds, and smoke, etc. They are challenging to model with conventional geometric elements, and the simulation with particle systems cannot be perfect.

1.1 Motivation

Although VRT is versatile, especially the traditional ray-cast method which can be applied to the transparent illumination model for the above art effects, the transparent illumination of the model focuses on the display of lighting effects and being artistic. Volume rendering medical data focuses on the intimate details of the material, which requires realistic while researching on volumetric data is a cumbersome task. For example, dealing with neurons and skin data tend to have elaborate structures. The research of TF design has been three decades, but there is no precise modified data to describe which TF is suitable for which type of volumetric data as such.

Furthermore, setting appropriate optical parameters to segment the Region of Interest(ROI) usually involves a process of repeated error attempts [13] and inefficient manual parameter adjustments. The last thing is large-scale medical data, which cannot be calculated in parallel through the same logical structure. The process often relies on logical judgment, which causes branching, and it isn't straightforward to be optimized. Those factors make volume rendering scientific data in the graphics engine unpopular.

With Augmented/Virtual Reality technologies sprung, those technologies have been applied in a critical part of medical applications. Such as Interactive Augmented Reality realizes on the Da Vinci robot, which provided guiding assistance to intraoperative surgeons [14] to gain safety. Although graphic engines become mainstream to implement Augmented/Virtual Reality, the mentioned constraints remain; Game engines are optimized in hardware for mesh

rendering and currently do not support DVR by default. Therefore, doing VRT medical images needs to rely on the third-party open-source library or 3D construction software support. Such as the novel plugin [15] [16] were provided for Unity, which used The Visualization Toolkit (VTK) to support pre-processing medical dataset.

Those limits raise my research interest. Whether there exists an intuitive approach: the rules are used by radiologists and surgeons for analyzing CT images, which can combine with the advantage of graphic engines to efficiently do VRT on medical data. And that means doing VRT on medical data without third-party library support in graphic engines. The implementation must consider specific strategies to design effective TF(visualization) and comprehend the applicable scope of each algorithm(performance); Ultimately, a 3D visualization system for medical images is realized in the graphic engines.

1.2 Goals of the Thesis

The main goal of the thesis is to implement DVR of CT data in the Unreal Engine 4 (UE4). Below are the individual objectives that will be achieved in this work.

1. Investigate the raw(CT) data to understand the point of view from radiologists and doctors when they assess CT images. Design a simple and intuitive method to segment ROI.
2. Investigate and implement techniques to pre-process and transform CT data into a format suited for DVR in the Unreal Engine. Based on cleaned data, TF will be designed for highlighting different ROI.
3. Develop and integrate a shader-based Direct Volume Renderer into the Unreal Engine using state-of-art lighting and shading algorithms focusing on high visual quality.
4. Optimize the DVR for Virtual Reality capable performance. There are two main reasons why doing VRT on medical data in the graphic engines is not popular. One is complicated TF adjustment, which is accounted for in 1 and 2. The other one is programming on the GPU, where branching is costly. Consequently, utilizing massive

parallel programming with minimum branching and acceleration data structures in the shader pipeline can help us maintain performance.

1.3 Thesis Structure

In the following chapter, it will cover the fundamental VRT and assess whether those methods are suitable for the rendering mechanism in the working game engine. The third chapter deals with the upcoming challenges. Based on those obstacles, it gives concrete approaches in detail. The fourth chapter modifies and implements the existing approaches and provides an explanation for the essential implemented-algorithm, which is based on the whole implementation stream. The last two chapters describe current results, including visual and performance comparison, and discuss further possible improvements.

2 Fundamentals

2.1 Volume Rendering Techniques

Volume rendering is a technique used for the visualization of volume data. Volume data mostly consists of voxels. A voxel represents a value encoding color information on a fixed grid in 3D space, and voxels are often stored as a multidimensional array. Volume data is normally split into several slices, and each slice is stored as an image. Therefore, volume rendering often deals with 3D objects visualization via 2D images. Based on the data set and visual result, VRT is classified into two groups: IDVR and DVR.

Indirect Volume Rendering

IDVR displays volume data via the iso-surface method. Iso-surface was extracted with a specific threshold from dataset. There were many ways to construct iso-surface. The most commonly used method was Marching Cubes (MC) [17]. The principle of MC was to treat a series of 2D data as a 3D data field and extracted certain domain values of a substance then connect into triangles in a certain topology. It processed each voxel in the volumetric field and determined the iso-surface construction form inside the voxel according to the value of each vertex. There were two main calculations for the iso-surface construction: the approximation of iso-surface in the voxel and the normal vector of each vertex of the triangle.

For immense data, the iso-surface construction is time-consuming for the first time. Once the iso-surface model is constructed, it is speedy to rotate and zoom the perspective. Therefore, there is no need to get access to the volume data. The model is required to rebuild if the iso-value is adjusted. Besides, you can also use a 2D cross-sectional view to examine the volume data, which can clearly show the details on a specific plane. However, it is impossible to examine the volume data of the entire space at the same time.

Direct Volume Rendering

In contrast to IDVR, the DVR does not require pre-processing, it observes data directly. And TF guides the user to pay attention to a specific subset of volume data, which helps the user to understand volume data effectively. The most software-based approach is the Ray Casting (RC) [10, 9] in DVR.

The basic principle of the RC had a normalized process that was provided by the list, as shown below. A ray was directly drawn from the pixel of the screen, and rays passed through the grid of the volume data. Adjacent points were obtained by interpolation of the color and opacity(TF) during the data classification phase. Colour and opacity were accumulated in sequence until the light passed through the volume, or the opacity reached 1.

- For each pixel
 - Cast a ray into volume
 - Interpolate data values from the voxel
 - Classify data values into optical properties
 - Composite optical properties
 - Return final pixel color

Generally, the memory cost and number of operations of MC is higher than RC [18]. Even if a surface representation can be generated by MC, but the iso-surface extraction requires complicated calculations, which is the reason why it's not possible to change the surface threshold interactively with MC [19]. Therefore, DVR is proved to be more efficient.

2.1.1 Ray-Casting & Shear-warp

The Shear-warp[21] was to decompose the 3D visual transformation into a 3D shear transformation and a 2D deformation transformation. The volumetric data was shear according to the shear transformation matrix then projected into the shear space to form an intermediate image. Finally, the intermediate image was warped to generate the resulting image. The most important feature of the algorithm was to select the slice data and projection data according to the three-axis(x, y, z) of view direction when the view direction changes, the projection

direction did not necessarily change through deformation transformation. The conversion of the sampling process from 3D space into a 2D plane dramatically reduces the amount of calculation.

We have to keep in mind that the goal of applying VRT on medical data is as good as possible. Shear-warp achieves a relatively fast processing speed by sacrificing sampling accuracy. Still, the potential quality of the image generated by this method is worse than that produced by the RC way. Furthermore, a limitation of Shear-warp is that the 3D data transforms into the intermediate coordinate system. When transforming, the view direction must coincide with one of the axes in the 3D coordinate system; otherwise, its advantage no longer exists.

DVR considers data as physical properties, but understanding data is not the only optimal in terms of physical realism. For this reason, various acceleration algorithms have been proposed for the RC method, such as Early-Ray Termination [22] and spatial data structures[23, 24], which reduce the amount of calculation and improve the rendering speed.

2.1.2 Optical Model

RC is a DVR algorithm based on image sequences. From the viewing plane, a line of sight is emitted in a fixed direction and passes through the entire image sequence. It sampled to obtain numerical values which are converted to optical properties such as color and opacity. So clearly, one of the important parts of DVR was determining how light reached the image plane, doing this, we assumed voxels have the contribution to participating medium. We can imagine mediums are particles that can absorb and emit light. To calculate how much light can reach the viewing plane, we can simulate different optical models, and the most common is using absorption and emission model[25] to achieve it.

Absorption Model

The absorption model Fig. 2.1 assumes each particle has an area $A = \pi r^2$, the number of particles per unit volume is ρ , cylindrical slab with the base area E and thickness Δs . I carries the amount of energy. The Equation 2.1 is to solve the function I so that we know at any step of the ray energy, and when ray arrives at the image plane, the function I works as the value

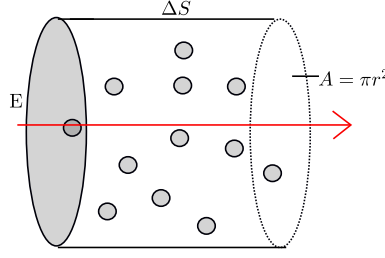


Figure 2.1: The absorption model illustrates the light passing through a cylindrical slab, and the participating medium only absorbs light.

that is a color of the pixel. Knowing function I can be solved by its Ordinary Differential Equation (ODE).

$$\frac{dI(s)}{ds} = \frac{-I(s) * A \Delta \rho(s)}{\Delta s} = -A \rho(s) I(s) \quad (2.1)$$

The Equation 2.2 tells us the amount of light will be allowed to go through, it is equal to the initial energy I_0 and multiplies by the exponential decay, the amount of decay depends on the density of particle ρ with t times of the area of the particle. ρ times A is replaced by a single variable called τ , which is typically called the extinction coefficient and is related to the density of the particle. Because we do not model density and size directly, in this term, it is replaced by a new variable called τ . This e to the power of minus integral of the extinction coefficient can be seen as transparency.

$$I(s) = I_0 * e^{-\int_0^s \rho(t) A dt} = I_0 * e^{-\int_0^s \tau(t) dt} \quad (2.2)$$

Emission Model

Fig. 2.1 is reused, the difference of emission model is that assumes each particle glows lights diffusely and intensity of the glow is C . Then glow per unit area is $CAE\Delta s\rho/E = CA\Delta s\rho$. Similar to the absorption model, now the goal is computing how much light of particle injects into the ray. And same as previous, we can solve Equation 2.3 with ODE.

$$\frac{dI}{ds} = C(s)A\rho(s) = C(s)\tau(s) = g(s) \quad (2.3)$$

In Equation 2.4 I , the energy of light is going to continue to change until it reaches the image plane. The solution for $I(s)$ is the initial energy carried by light plus the contribution from all

the particles along the ray.

$$I(s) = I_0 + \int_0^s g(t)dt = I_0 + \int_0^s C(t)\tau(t)dt \quad (2.4)$$

Add emission and absorption together is the most commonly used optical model in DVR. Equation 2.5 explains how much light is going to change along the ray from back to the image plane. The first term is emission, we have an intensity C multiply extinction coefficient, and the second term is absorption; the negative sign is the particle will absorb the light. We can rewrite it as $g(s) - \tau(s)I(s)$.

$$\frac{dI}{ds} = C(s)\tau(s) - A\rho(s)I(s) = g(s) - \tau(s)I(s) \quad (2.5)$$

Equation 2.5 is solved by ODE to get Equation 2.6. $I(D)$ calculates the final value of the ray when it travels D distance(see Fig. 2.2). The first term is absorption; it includes background energy I_0 goes through the image plane, and energy will be absorbed because of extinction property. The second term is the integral contribution of $g(s)$ function. $\int_0^D g(s)$ goes through g at a different position along the ray. But whenever the particle emits light, it needs to go through again exponential decay caused by the particle in front of it $e^{-\int_s^D \tau(t)dt}$ before the light reaches the image plane.

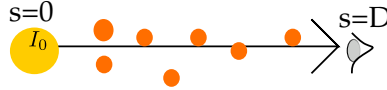


Figure 2.2: The initial light with energy I_0 travels D distance to reach our eye(image plane).

$$I(D) = I_0 * e^{-\int_0^D \tau(t)dt} + \int_0^D g(s)e^{-\int_s^D \tau(t)dt}ds \quad (2.6)$$

2.1.3 Transfer Functions

In DVR, we mapped a sample data value at the fixed position and decided optically-physical properties for each data value. Usually, there is no feasible method to obtain emission and absorption properties from data value directly. Consequently, TF design here was to assign optical attributes to data values for emphasis salient data structure and deemphasis others.

The process of finding TF is essentially classifying voxel, identifying features, and giving different attributes, which are also called classification [26].

From the optical equation, we can map TF in the rendering equation, The α (opacity) is determined based on the coefficient of extinction τ . Instead of modeling it out, we take the whole exponential term as $1 - \alpha$ [27]. Then integrate the energy contribution of current from each sample point, and this represents the color.

Finding TF is still tedious work because sophisticated features do not just involve a single particular range, and some features are not easy to display without a lot of tweaking. We listed existing techniques which were common strategies for designing at a TF:

1. Trial and error[28]: Manually controlled the color and opacity of each sampled value using a graphic user interface. [29] provided intuitive user interfaces for non-expert users.
2. Histogram assisting (1D TF): Data values were divided into finite clusters and counted their frequency. Different clusters can be assigned with different colors and opacity [30]. The drawback of 1D TF is an unambiguous visualization of the target structures and can not be produced. [31] measured intensities of osseous tissues and contrast medium and got mixed due to occurring partial volume effects.
3. Multi-dimension TF: The proposal method [32, 33] tried to plot data value and gradient(First derivative of data value) in an XY plane. It mainly solved boundaries among complex features. [34] demonstrated the benefit for 2D TF and successfully viewed vascular structures in examination data. [35] used multi-dimension TF to classify variants MRI data.

2.1.4 Composition Schemes

Recap RC algorithm. The casting ray sampled and generated optical properties; the process for blending was called composition schemes before getting the final color. There were some commonly employed: MIP [36] that can be generated rapidly but only used a small fraction of data because of lacking depth information; Alpha blending [9, 11] was often implemented

by using the Riemann sum(Equation 2.6) to discretize the continuous function.

$$C_o = \alpha_s * C_s + (1 - \alpha_s)C_d \quad (2.7)$$

Equation 2.7 is the alpha blending equation. α_s represents the transparency of the transparent object. C_s , C_d and C_o respectively represent the original colour of the transparent object, original colour of the target object and the observing colour value. The process of a ray passing through the volume can be sorted from the front to the back:

$$C_i^\Delta = C_{i-1}^\Delta + (1 - \alpha_{i-1})C_i \quad (2.8)$$

$$\alpha_i^\Delta = \alpha_{i-1}^\Delta + (1 - \alpha_{i-1}^\Delta)\alpha_i$$

or reversed from the back to the front:

$$C_i^\Delta = C_i + (1 - \alpha_i^\Delta)C_{i+1}^\Delta \quad (2.9)$$

$$\alpha_i^\Delta = \alpha_i + (1 - \alpha_i^\Delta)\alpha_{i+1}$$

C_i and α_i are the color values and opacity obtained by sampling on the volume texture, which is the data contained in the voxels; C_i^Δ and α_i^Δ represent the accumulated color values and opacity.

2.1.5 Ray-Casting Optimization Methods

The light passes through the cube is the process of light passing through the volume texture. Generally, we pre-computed the travel distance when the ray entered and exits the cube, and also accumulated the step size from ray sampling each time in the volume. Utilizing this to know when we can stop ray to march. Based on it there are many techniques to accelerate RC.

Early ray termination

The process of passing through the volume texture is Riemann integral. For each ray the transparency is set as 0 in the begin, the sampling volume texture coordinates are calculated. Stops compositing of further sample locations until the light is cast out of the cube or the accumulated transparency reaches 1.[37]

Skip blank space

R. Yagel et. al[23] utilized spatial information about the contents of volumetric data to adapt the sampling distance between two sampling locations in order to not compute empty voxels. In this algorithm, the volume can be subdivided into uniform-size blocks [38] or encoded to the Octree [39], the main idea of the Octree recursively divided space into equal 8 sub-space then based on its dividing decision strategy to keep dividing or set as a leaf node. A comprehensive analysis of Octree [40] and storage of voxel data structure was discussed. With the stunning result, Ruijters and Vilanova [41] implemented the Octree to speed up rendering of brick by skipping empty space.

2.2 Standard Illumination and Shadowing

Phong and Blinn-Phong

Various rendering effects in Shader naturally involve the use of various lighting models. Phong and Blinn-Phong [42] are common default lighting rendering methods in computer graphic software. Both local illuminations consider three terms:

$$I = I_{ambient} + I_{diffuse} + I_{specular} \quad (2.10)$$

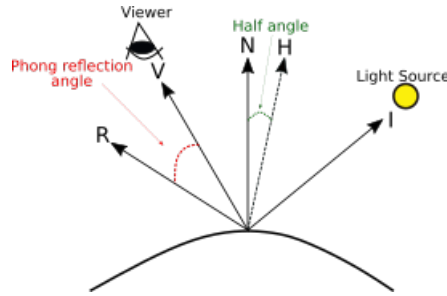


Figure 2.3: A diagram showed the vectors used in calculating Phong and Blinn-Phong shading.

N: the surface normal, I: vector in the direction of a light source, R: vector in the direction of optimal light reflection, V: vector in the direction of viewer, H: "halfway" vector, a vector in the direction of $(V + I)/2$.

The detail of both equations can see Equation 2.11 and Equation 2.12 with diagram Fig. 2.3.

k_a, k_d, k_s are scalar constant between 0 and 1, I_a, I_d, I_s are color and intensity of the global light, and m_a, m_d, m_s are color of the material. In the Phong model, the angle between R and V cannot exceed 90 degrees, the \cos will become negative, and lighting error will occur. Blinn-Phong defines the "halfway" vector to solve the problem, and under the same conditions, the highlight range of Blinn-Phong is larger than that of Phong, and the realistic effect of the Phong lighting model is better. The variables of both equations can be computed.

$$I_{Phong} = k_a M_a I_a + K_d M_d I_d (I \cdot N) + k_s M_s I_s (V \cdot R)^{Shininess} \quad (2.11)$$

$$I_{Blinn-Phong} = k_a M_a I_a + K_d M_d I_d (I \cdot N) + k_s M_s I_s (H \cdot N)^{Shininess} \quad (2.12)$$

Ambient Occlusion

Ambient occlusion [43] is a global shadowing method, which means that light from each point affects other points in the scene. Compare to local shading methods such as Phong / Blinn Phong shading. Ambient light occlusion can add realism because rays are projected "up" from surface points relative to the surface from all directions, which takes into account the brightness of the light due to surface light occlusion relative to the light source.

2.3 Rendering in Unreal Engine 4

This section lists the coverage and division of UE4 from the macro functions of the rendering engine. It does not involve the implementation details of each functional module—reference to the official Graphics Programming Overview [44]. The input of the renderer is the original geometry and material data. The renderer converts the geometry and material data into a rendering API, which is named as Render Hardware Interface (RHI). The RHI is a separate module that includes support data, rendering state, Shader, and parameters of Shader. These data are assembled into a RenderPipeline to execute. Finally, the rendering result is exchanged to the Rendering Target (RT).

The Generation of Shader

Shader generation compiles the node graph in the material editor into High-Level

Shading Language (HLSL) code. The material editor avoids the selection problem of the shading stage. Most of the generated shaders work in the Pixel Shader (PS) stage. In the material editor, MaterialExpression will submit the corresponding HLSL code to FHLSLMaterialTranslator and replace it with the code in the material template. In the end, we will get Shadermap.

Drawing process

The code generated by the material editor is a small part of the Shader. In UE4, there are several Passes in one rendering. The renderer gives the corresponding Shader for PASS based on DrawingPolicy. Take Deferred renderer as an example; during the Runtime phase, it extracts various rendering data in GBuffer for various PASS. Those Passes are drawn with MaterialMeshShader and GlobalShader. Hence, the editing part in the material editor only involves the "material expression." The functional form of the PASS determines what kind of data for each PASS needs to input in Vertex Shader (VS) and PS.

2.4 Overview of Digital Imaging Communications in Medicine

The body absorbs some energy(radiation) when X-ray passes through. The left energy hits the detector behind the body, and then detector records the physical energy of the ray as an electrical signal, which is quantified by a mathematical calculation.

In 1982, The lack of uniform standards became an obstacle to exchange information between various medical imaging equipment. American College of Radiology and National Electrical Manufacturers Association [45] formulated a standard for the digitized transfer, display, and storage of medical images. It is called Digital Imaging Communicaitons in Medicine (DICOM) and as the benchmark for medical image exchange between hospitals and international. The standard DICOM defines imaging, consist of the format of related information and exchange method. Using this standard, people could establish an input/output in medical imaging equipment.

Based on the DICOM 3.0 standard, each image carries a large amount of information,

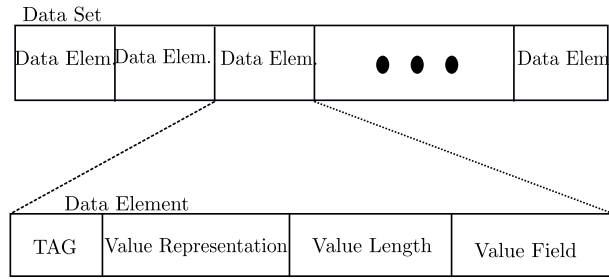


Figure 2.4: DICOM data elements and the content of each basic unit.

which can be divided into four categories: Patient, Study, Series, and Image. A Tag can identify each item of information. The tag is composed of two hexadecimal numbers(Group, Element). Such as (0010, 0010) represents the Patient's name. As Fig. 2.4 shown, each item of information is packaged into basic unit *Data Element* which consists of four parts [46]:

- i Tag: Two hexadecimal numbers, DICOM data element can represent uniquely by tag.
- ii VR: Used in the standard DICOM to describe types of data elements. There are 27 in total.
- iii VL: The length(in bytes) of data in the data field of the data element.
- iv VF: Contains the value of this data element.

2.4.1 Analysis of Computed Tomography Data

Every CT scan contains at least a hundred slices. Among those data elements, the most important tag is the pixel elements. Each pixel is assigned a value that is the average of all attenuation values. However the raw pixel value of DICOM has no practical meaning, so we usually transfer this value into the Hounsfield scale, all the CT values can be computed as follow:

$$HU = pixel_val * rescale_slope + rescale_intercept \quad (2.13)$$

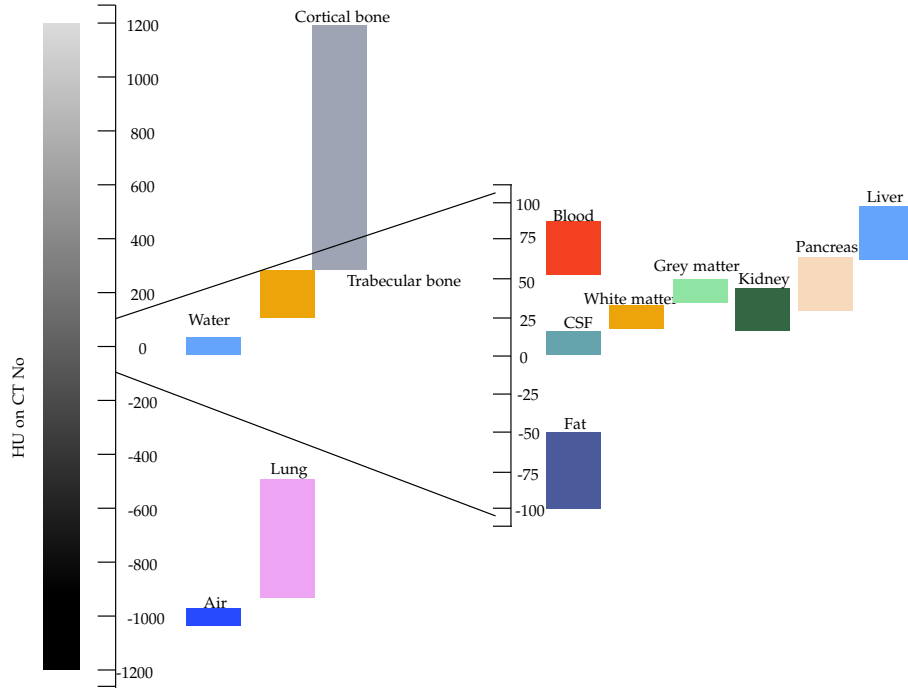


Figure 2.5: Hounsfield Unit Scale

$Pixel_val$ reflects the electron density of the imaged tissue at i^{th} location(pixel). The two tag values $rescale_slope$ and $rescale_intercept$ can be read directly from DICOM header information. Hounsfield Unit (HU) is the CT value of the i^{th} pixel. The range of CT values for specific tissues of the human body is fixed. The Fig. 2.5 illustrates most common HU usage in clinical and radiology [47].

2.4.2 Hounsfield Scale and Windows

Mr. Godfrey Hounsfield [48] compared the assigned values to the water attenuation value, and since then, all data were named in Hounsfield and reflected in the Hounsfield scale table. This Hounsfield gauge assigns the attenuation value of water to zero (HU). The general CT range is -1000 to 2000 HU, but some of modern scanners rise to 4000 HU. HU number is converted into a digital image by assigning a greyscale intensity with +1000 (white) and -1000 (black) at both ends of the spectrum. The eyes of humans can only distinguish 20 different from the grey level. Even CT images

display differences, but the eyes cannot detect it. Therefore, radiologists [49] by adjusting window width and window level to border tissues fall within the greyscale of the image, Table 2.1 provides general used tags information for viewing medical images. Fig. 2.6 explains that the adjusted window and level setting express a range of CT values in grayscale, which can be recognized by the human eyes.

Table 2.1: A description of the used tags table.

Tag	Keyword	Tag Description
(7FE0,0010)	Pixel Data	A data stream of the pixel samples that comprise the Image.
(0018,0050)	Slice Thickness	Nominal slice thickness, in mm.
(0028,0010)	Rows	Number of rows in the image.
(0028,0011)	Columns	Number of columns in the image.
(0028,1052)	Rescale Intercept	The value b in relationship between stored values (SV) and the output units
(0028,1053)	Rescale Slope	m in the equation specified by Rescale Intercept
(0028,1050)	Window Center	Center of the window
(0028,1051)	Window Width	Width of the window

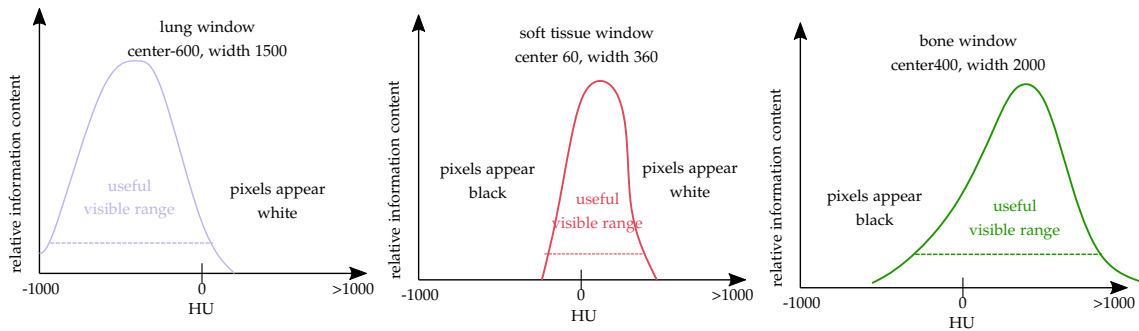


Figure 2.6: Adjusted windows with preset values of window centre and width.

3 Concept

In the previous chapter, well-known techniques in VRT were reviewed. Current work implemented a straight-forward approach called Ray Marching (RM) in VRT. This chapter covers the challenges that we faced and the approaches that were used to tackle each challenge. A state of art graphic engine, UE4, was chosen to realize DVR. Additionally, the current thesis was based on the work of Ryan Brucks [1], which is an implementation of a simple form of volume rendering in UE4 for general volumetric effects.

3.1 Challenge

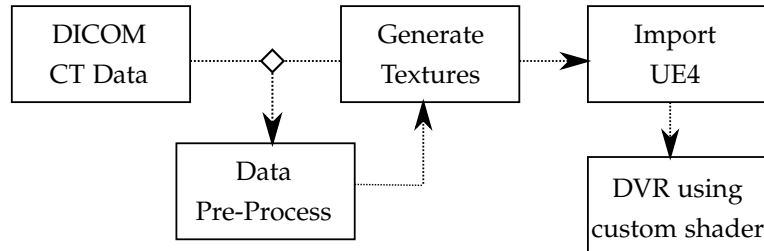


Figure 3.1: My Program Flow

The core of VRT we pursued only showed body details rather than surface details. The differences between the two visual results and algorithms were shown in chapter 2. According to the 3D volumetric data, the technique of displaying all the body details on the 2D image at the same time is called DVR. With DVR, the overall distribution of multiple substances can be displayed in one image, and the control of opacity can reflect the iso-surface. Fig.3.1 shows an overview of the required subtasks for

importing raw medical images into UE4. The most vital steps are discussed in detailed:

1. Data Pre-processing.

It is possible to write our DICOM reader in UE4 without the support of a third-party library. However, to avoid the complications that may come with the large data set, data were first transformed into texture format before importing into UE4. This transformation turned the 16-bit unsigned integer DICOM data into 8-bit unsigned integer texture format data. This allows us to use a lookup table from HU to distinguish tissues. Unfortunately, this format transformation inside the Unreal Engine transfers all the values into floating-point values. This introduced difficulties in feature segmentation and overlapping HU range between tissues. Accordingly, pre-processing and cleaning the data became critical for further segmentation, The goal for data cleaning and pre-processing was to keep as much feature from each image slice as possible, so that the user can easily select the corresponding tissues by manipulating the HU in the UE4.

2. Design an efficient shader for DVR.

In UE4, the texture operations are under the pixel shader pipeline. Therefore, the intuitive way was to utilize pixel shader to sample texture data. Those could be implemented in the material editor. The advantage of realizing DVR in the graphic engine was that it allows customized shader adoption. Furthermore, the ray-marching-based DVR can be implemented in the acceleration hardware. Understanding PS in the UE4 was not only handling different vertex and pixel but also preventing branching and inefficient algorithm.

3.2 Approach for Obtaining Volumetric Data

Before generating textures for reconstructing the medical data set, it was necessary to parse the imported medical images. An important part of the work was to obtain information stored in the images. The previous steps used information tags from

Table 2.1 to acquire meaningful HU values. Although one could generate multi-windows through window adjustment, using multi-windows on the same image was not an ideal method. J. Mandell et al. [50] proposed to use a relative attenuation-dependent image overlay to visualize the full range of CT in one view. This approach facilitated threshold division so that the bones and soft tissues can be successfully separated by the CT values.

3.3 Shader and 3D Texture

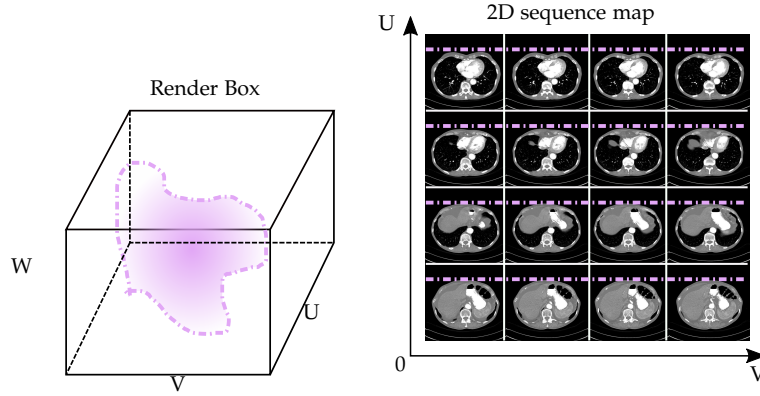


Figure 3.2: Simulate the rendering object through custom UVs sampling sequence map.

With radiological knowledge, cleaned CT data and 2D sequence texture were generated. Accordingly, our independent VS and PS could be designed to render medical images in the material editor. Regarding the data transmission from VS to PS, only custom UVs were required. It transmitted a fixed float2 data type. We, therefore, used a 2D sequence map and then stacked this sequence map into a volume texture [1]. As shown in Fig. 3.2, a 2D sequence map was first generated, and then 3D voxel texture was generated through custom UVs sampling. A 3D unit cube was then used to simulate its UVW. Ultimately a bounding box of the rendered object was obtained and turned into a unit cube.

3.4 Algorithmn

The previous sections explained how the medical data was pre-processed and sampled into voxel in UE4. The remaining approaches were the customized shader, which covered algorithms to reach our goal. This included texture-sampling along the ray, feature-extraction, visual quality enhancement and expected feature of interest, enhance visual quality and empty space skipping.

3.4.1 Ray-Marching-Based Volume Rendering

RM is a variant of RC. Especially, the method we were using was RC for DVR. Traditional RC calculates all intersection points and return the color from the closest intersection point to each pixel on the screen. In RM, ray considers whether to hit the object after movement instead of calculating all intersection points. Step size is a crucial point that determines accuracy. Firstly, there is a 3D volume texture. N rays are emitted from the camera, and each ray has its sampling step size. Secondly, when the ray enters the volume, it takes a sample for every step, computes TF value, and then blends with the value accumulated by the ray. The procedure is the same as mentioned in the subsection 2.1.4; the only difference is taking advantage of the various step size in RM.

Utilizing RM samples 2D sequence texture, we only obtained pixel values and did not know the vertices of the models. To change the structure of the model, we needed to construct the model data in the PS. RM can draw a lot of objects without any vertex data. It creates its own rendering *rule* in the PS. The reason for that RM can render the correct pattern is due to the reversibility of the light path. In other words, the light emitted from the light source scatters and eventually enters the camera with the same effect as coloring and sampling the rays emitted from the camera.

This algorithm is suitable for implementation on the GPU, because the calculation of each ray is independent and parallel, and the GPU has an inherent advantage of parallel computations. The disadvantage is that the amount of each calculation is

large; due to that, it has to consider lighting, shadow and TF at each step. Although hardware acceleration (RM technique) was used, the fact that the micro thread of a Wrap required different branches was still a critical issue. It happens when the condition uses dynamic values, that is, the compiler only knows the values in the runtime, which causes branching on the GPU, which prevents it from being able to make the most out of the simultaneous calculations. This condition is vicious and needs to be avoided.

3.4.2 Local Illumination and Shadow

Local Illumination

Although absorption with an optical emission model can create decent simulation, this doesn't account for the light reflection of surrounding objects. An object which does not emit light by itself can be seen because it reflects light from elsewhere. Creating a "perfect" lighting model can be very complicated. In literature, people usually use the approximation of actual lighting methods for VRT, such as Phong or Blinn-Phong section 2.2. In volume shading, there is no vertex normal information; instead, we computed the gradient of the scalar field as the normal vector. In a scalar field $f(x)$, normal vector can acquire form Gradient $\nabla f(x)$ that is perpendicular to isosurface. Central differences are one of the numerical computations of the gradient:

$$\begin{aligned} G_x &= V_{x+1,y,z} - V_{x-1,y,z} \\ G_y &= V_{x,y+1,z} - V_{x,y-1,z} \\ G_z &= V_{x,y,z+1} - V_{x,y,z-1} \end{aligned} \tag{3.1}$$

Through Equation 3.1, the further gradient needs to be normalized, then can be applied as normal vector. Frida H. et al [51] proposed Local Ambient Occlusion (LAO) for direct volume rendering method, instead of taking conventional approach to compute gradient $\nabla f(x)$ for normal vector, it considers integral absorption coefficient(α). The advantage with LAO can avoid the fully shadowed area that may hide relevant

information. *Shadow Strategy*

Although the above illumination model is straightforward to implement, few disadvantages exist. The critical disadvantage is that it cannot reflect the shadows between objects. Shadows can make the scene look much more realistic and allow the observer to obtain the position relationship between objects. Because the shadow is the result of the light being blocked, the conventional method is casting a secondary ray to check whether a light source can "see" this intersection. When the light of a light source cannot reach the surface of an object due to the blocking of other objects, then the object is in the shadow [52].

Regarding semi-transparent volumes, they give rise to soft shadows due to light absorption. The self-shadow for opacity volume has been proposed [53], which calculates a histogram of the density field to generate cluster bins and construct 1D visibility function along the viewing ray. The visibility function [54] has also been used in depth shadow map paper, which is known as transmittance function. Through this method, the cluster bins have different shadow contributions.

$$E = L_0 \exp[-\alpha(s)d] \quad (3.2)$$

The transmittance function can be modeled with Beer-Lambert law. The Equation 3.2 expresses transmitted radiance E as that of incident light L_0 , which exponentially decays as a function of distance it travels, with α being the absorption coefficient.

3.4.3 Artefacts

VRT usually has a trade-off between quality and performance. Hence, there are more or less noticeable artifacts in the resulting image based on variant stages of VRT [37]. Identifying the sources of artifacts can help to find the method which can produce a high-quality image. Aliasing artifacts are visible at the edges of the slice polygons, and the viewport-aligned slice can remove it. Additionally, color-bleeding artifacts occur during interpolation, and pre-multiply colors by their corresponding opacity value before interpolation can suppress it. The most tricky artifact is the wood-grain

artifacts, it happens if the insufficient sampling rate is used, and numerical integration does not hold for high opacity. This can be solved by increasing the sampling rate. However, since our RM approach already demanding on the computation effort and the CT data are usually large, we should avoid expensive readback of it from the GPU memory. In order to achieve the best in both quality and performance, we adopted

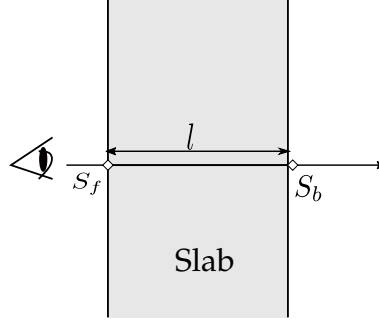


Figure 3.3: Pre-Integral Slab: A slab of the volume between two slices.

work from Pre-integral [55] to reduce wood-grain effects. The conventional method takes the front sample value(s_f) and back sample value(s_b) directly via TF that easily introduces high frequency during ray integration. Pre-integration calculates the exact line integral between pair samples on the slab(see Fig. 3.3) to generate continuous scalar field: $s_l(x) = s_f + \frac{x}{l}(s_b - s_f)$. The pre-integrated Look-Up Table (LUT) can be analytically integrated to incorporate all features of the TF between s_f and s_b in order to avoid problematic product of Nyquist frequencies [56]. The integral equation can be rewritten as Equation 3.3.

$$I(D) = I_0 * e^{-\int_0^D \tau(s_l(t))dt} + \int_0^D g(s_l(s))e^{-\int_s^D \tau(s_l(t))dt}ds \quad (3.3)$$

However, CT scan provides a longitudinal section, this kind of section usually contains plenty of empty pixels, and the adjusted sampling length does not have extra sampling between pair samples. Hence, the pre-integral cannot work for all cases. Stochastic Jitter can hide wood-grain artifacts without adding sampling and is the best approach for all data. The coherence between pixels that becomes manifest in artifacts can be hidden by noise. Jitter randomly offsets the ray sampling position.

3.4.4 Sparse Voxel Octrees

The step size can be adjusted in RM technique when ray visiting empty sub-space needs to take a big stride forward to skip redundant computing. Hierarchical partition on three-dimensional space is a standard method to handle complicated geometry scene and VRT data. Here, the Octree is applied as our hierarchical spatial scheme. After the pre-processing data stage, we saved the CT values as an 8 bits greyscale. And through sampling 2D sequence map, the voxels distributed in the unit cube. Therefore, the taken subdivision strategy was based on the average density of voxels in the sub-space. Then the store Octree was used as texture type because ultimately, the tree is traversed directly in the material editor. That also means the tree should be full octave, which can be stored as a linear array without using the pointer. For the traversal strategy, a top-down arithmetic method [57] was used, which selected child sub-space from the current one. The selection was based on the parameter of the ray and comprised simple comparison, which reduced numbers of instructions in PS. The only note is that the UE4 coordinate system is different from the general system, which will be explained in the next chapter in detail.

Branch-on-Need Octree Construction

The ideal condition is constructing a full Octree, then store the tree as an array(texture), which can be identified without using any pointers. Unfortunately, most of the data volume doesn't have dimensionally equal size, and additionally, the size of the data is at the scale of a power of 2. As a result, a fully created Octree was not possible. Take volumetric data with 16x8x4 resolution as example, Table 3.1 display non-full Octree subdivision in each level. We can follow Equation 3.4 to calculate the total number of tree nodes for the volume with resolution S in each dimension. Therefore, this example has $1 + 8 + 8 * 8 + 8 * 8 * 4 + 8 * 8 * 4 * 2 = 814$ nodes and node data ratio derived from Equation 3.4 optimal node ratio is $(\frac{S^3-1}{7})/S^3 \approx 0.1428(optimal)$.

Level(No. nodes)	Subdivided
Level 1(8)	8x4x2
Level 2(8)	4x2x1
Level 3(4)	2x1x1
Level 4(2)	1x1x1

Table 3.1: No. of nodes in each level for non-full Octree with 16x8x4 resolution

In final, non-full Octree ratio is $812/512 \geq 0.1428(optimal)$.

$$8^0 + 8^1 + \dots + 8^{(\log S - 1)^2} = \frac{S^3 - 1}{7} \quad (3.4)$$

Larger node data ratio has more considerable overhead because each node does not always have eight children. It is thus difficult to predict the position of a child node, which stores pointers that represent additional space overhead. Wilhelms and Van Gelder [58] proposed Branch-on-Need Octree (BONO) method, which allows zero-padding imaginary overlay (power of 2) volume to perform a partition with imaginary volume. This way, the ratio of nodes can be minimized to reduce space overhead.

4 Implementation

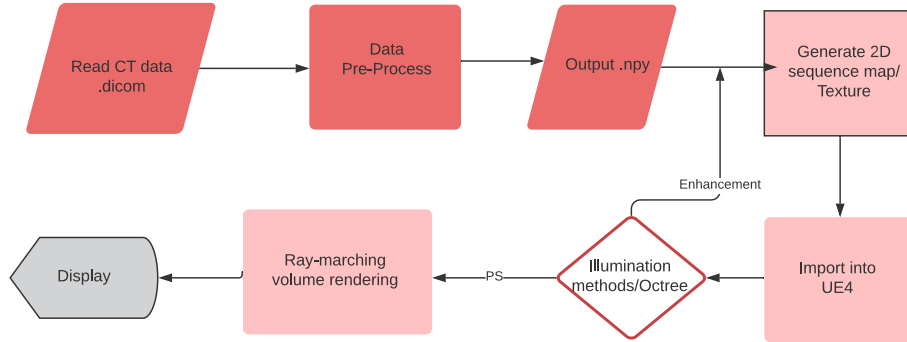


Figure 4.1: Implementation pipeline.

This chapter describes adjustment and of our method, as shown in Fig. 4.1. The data pre-process block. Here will give elaboration based on Fig. 4.1 in order. The data pre-process block was running under the python3 environment and saved as .npz format. The cleaned data was saved as a 2D sequence map for Alpha TF purpose, gradient map for different illuminating methods, and texture for construction of octree. The main implementation of ray-marching-based VRT on medical images used the Unreal Engine 4.22.3 version. We eliminated the aliasing artifact and acquired the current sampling frame separately based on the viewing-aligned plane approach and "1DTo2D" function introduced by Rayan B [1].

4.1 Pre-procsssing - Blended Window

With the rapid growth of DICOM analysis in clinical medicine, we now have greater needs as well as the ability to develop frameworks that support the DICOM API.

In this work, we employed PYDICOM modules supported in Python to read files, since it is mainly used to manipulate various data elements of DICOM files. Usually,

Keyword	Values
Slice Thickness	3.0
Rows	512
Columns	512
Rescale Intercept	-1024
Rescale Slope	1
Window Center	[40, 700]
Window Width	[350, 1800]

Table 4.1: The Patient ID: 01_13 used tag values table.

one CT scan of the patient is a folder, there are a sequence of DICOM files under the folder, and each file is called a slice. In the pre-processing block contains several steps. Those steps were merged as *Initilization* function(see algorithm 1). First of all, scan a patient's catalog and load all the slices; Secondly, sort all slices based on the switched z-direction and extract all used tags values(Table 4.1); In the third step some scanners have a cylindrical scan range, but their images are rectangular. Values that fall outside the boundaries have a fixed value of -2000, and those values are reset to 0, which is the same as the HU value of Air. Then, return to HU by multiplying the rescaled slope and adding the intercept (Through reading tag header can extract those values); The last step, use the blended window (algorithm 2) to produce images.

Take patient ID: 01_13 as an example, Fig.4.2 displays the histogram about the value and image after passing HU on the first slice. The data under the first slice is ranging from -1024 to 1372, which indicates that the bone value is included. In order to segment bone and keep tissues at the same time, we computed the bone window, soft-tissue window, and lung window by algorithm 2. As illustrated in Fig.4.3, after creating all windows, we set a soft-tissue window as the main window. We blended it with the bone window where the attenuation is low and the lung window where the attenuation is high together into a single gray-scale image. The advantage of setting a

Algorithm 1: Initialization

Input: dir_path

Output: A set of tag values

Result: Save cleaned data as .npy

```

1 load_files = dicom.read_file(dir_path);
2 load_files.sort(key = lambda x: int(x.InstanceNumber));
3 try:
4     extract tag values;
5 except:
6     print(exception occurs);
7 for idx ← 0; i < Num_of_Slices; idx+ = 1 do
8     load_files[idx][load_files[idx] == -2000] = 0;
9     if slope != 1 then
10         load_files[idx] ← slope*load_files[idx].astype(float64);
11         load_files[idx] ← load_files[idx].astype(int16);
12     end
13     load_files[idx] += (intercept);
14 end
15 save(load_files);
16 return tag values set

```

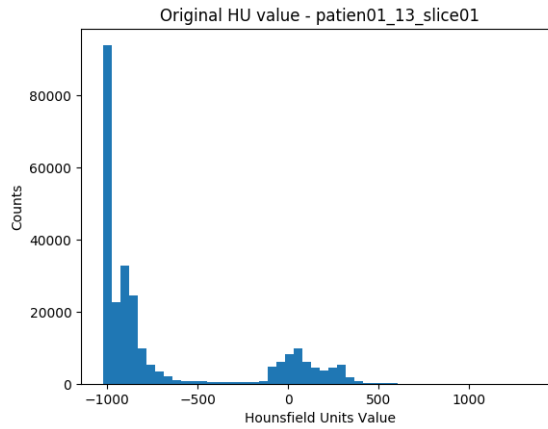


Figure 4.2: Histogram of the first slice P01_13 HU data. Data range[-1024, 1372] distributes into 50 bins.

Algorithm 2: Blended window

Input: Window Center $WC = [c_1, c_2]$, Window Width $WW = [w_1, w_2]$, $numpy_path$

Output: Blended window array

```

1  $numpy\_file = read(numpy\_path);$ 
2  $Bone\_Window = Windows(WC[1], WW[1], numpy\_file);$ 
3  $SoftTissue\_Window = Windows(WC[0], WW[0], numpy\_file);$ 
4  $Lung\_Window = Windows(WC[1], WW[0], numpy\_file);$ 
5 for Each pixel do
6   if retained low attenuation then
7      $SoftTissue\_Window += Lung\_Window ;$ 
8   else if retained high attenuation then
9      $SoftTissue\_Window = Bone\_Window ;$ 
10 end
11 return  $SoftTissue\_Window;$ 

```

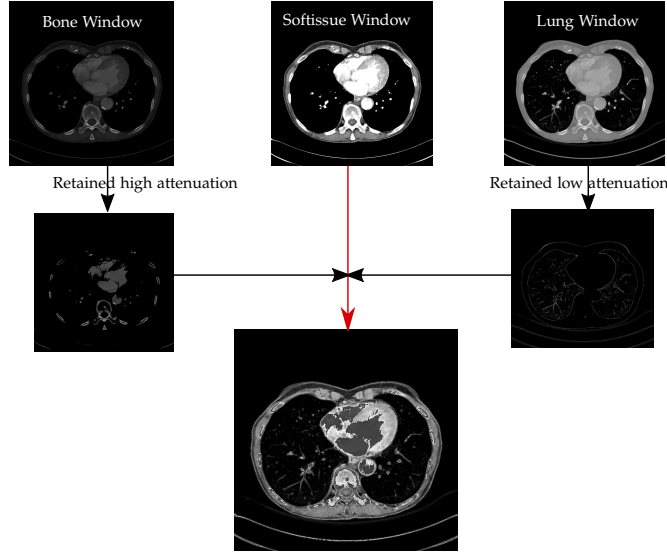


Figure 4.3: The diagram demonstrates the window blending algorithm, which fixes the soft-tissue window as the primary layer. Only relatively high attenuation information is retained from the bone window, which replaces the corresponding pixel locations in the soft-tissue window. On the lung window, only relatively low attenuation is summed with the underlying soft-tissue window.

soft-tissue window as the main window is that it allows the information of essential organs(Fig. 2.5), including kidney, liver, brain matters, etc. remained after blending as its HU locates between -150 to 150.

4.1.1 Pre-computing Transfer Function of Alpha

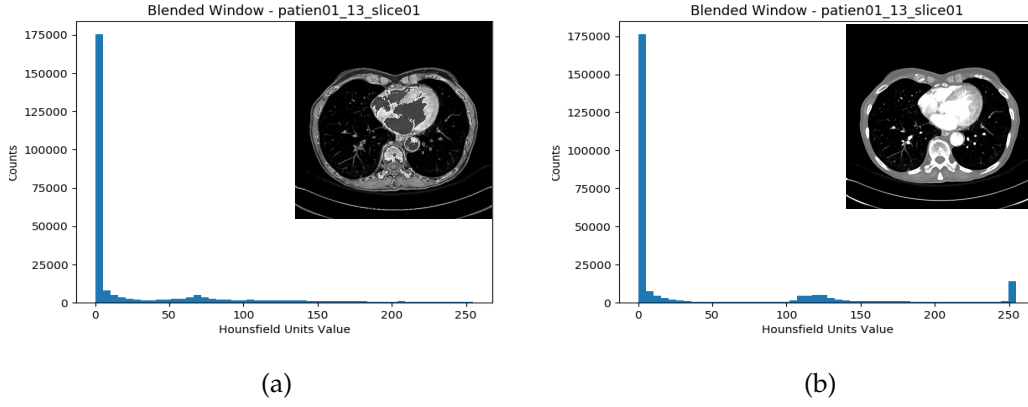


Figure 4.4: Histogram comparison of different blended windows approach. (a) was implemented with [50] approach: the soft-tissue window is man window and blend with lung and bone windows (b) considered soft-tissue and lung windows only, in order to generate Alpha TF in advance.

Although Jacob M. et al. [50] proposed method can get the visual segmentation, blending values of bone are close to and overlap with values of tissue. The ideal case for reducing the consumption of computing is to set the scalar values to be as alpha TF directly when importing the texture into UE4. Because of DVR, to render semi-transparent results, the ideal distribution from dark to light in grayscale is lung, tissues, and bone in order. The above ≈ 390 of the soft-tissue window was set to near white, which covers the range of bone(Refer to Fig. 2.5). The proposal classification of bone should be near white, lung near black, and soft tissues locate in the middle. The default setting of the soft-tissue window already segmented two categories: soft-tissue and bone. The remaining is to replace the same pixel location of the lung window to the soft-tissue window.

Therefore, based on algorithm 2, only soft-tissue window, and lung window are considered and comparing pixel location to replace empty values to the HU of lung ultimately. Fig. 4.4 compares histograms between the three-blended window and two-blended window. The two-blended window has clear clusters than the three-blended window.

4.2 Custom Ray-Marching

This section describes "ray-marching-based VRT" block in Fig. 4.1 diagram. After creating the medical images, we can import them into the material editor. Due to the material editor is graphical and node-based shader, code can be added in "custom node"; However, it doesn't support "LOOP" node to read hundreds of image, the intuitive way is to save images as sequence map(flipbook) or import the video file as a media texture. In Fig. 4.1, the "2D sequence map" block aims to produce such kind of texture maps, and the concept of the stack can do it. The most important thing is, "HOW" to sample it after importing to the editor.

Determine raymarching fragments

Under the custom node, each viewing ray needs to sample its corresponding UV coordinate in the sequence map. In order to shift UV on the 2D sequence map and estimate the corresponding voxel in 3D space at the same time, a proxy geometry is required. That is, to determine which pixel on the 2D screen belongs to which volume texture. It is reminiscent of a bounding box. The proxy geometry can use the node "BoundingBoxBased_0-1_UVW" to simulate local UVW, the slice-thickness tag value is considered to adjust W length:

$$W_length = slicethickness * Num_of_Slices \quad (4.1)$$

$$\begin{aligned} W_min &= W_length/2 \\ W_max &= (1 - W_length/2) \end{aligned} \quad (4.2)$$

Algorithm 3: Raymarching Fragments

Input: $W_{min}, W_{max}, PlaneAligned, LocalCamPos, CamVector$

Output: $entrypos, boxthickness$

```

1 intersection1 = front intersection align box( $W_{min}$ , LocalCamPos, CamVector);
2 intersection2 = back intersection align box( $W_{max}$ , LocalCamPos, CamVector);
3  $t0 = \min(intersection1)$ ;
4  $t1 = \max(intersection2)$ ;
5 //viewing-align plane [1];
6  $planeoffset = 1 - \text{frac} ( t0 - \text{length}(LocalCamPos - .5) ) * MaxSteps$ ;
7  $t0 += (planeoffset / MaxSteps) * PlaneAligned$ ;
8  $t0 = \max(0, t0)$ ;
9  $boxthickness = \max(0, t1 - t0)$ ;
10  $entrypos = localcampos + (\max(0, t0) * CamVector)$ ;
11 return  $float4(entrypos, boxthickness)$ ;

```

Equation 4.2 calculates the minimum and maximum W position, respectively (Note the CT slice is square; therefore, we only need to beware of Z -axis). As long as a correct bounding box size is obtained, the next step is to compute the first/last sample point base on the viewing-align plane; this part adopted [1] approach directly. In the end, only the fragments are covered by the bounding box on the screen, which needs to be raymarching. The function displayed in algorithm 3.

Raymarching direction and step length

Given the sampling position, each step length and the maximum number of sampling steps were pre-computed for each viewing ray. Equation 4.3 depict sampling step size, the XY stores the number of rows and columns in the 2D sequence map, XY product is divided by W_{length} to acquire local slice thickness. To get adjusting step length need to multiply local slice thickness with normalized local camera vector.

$$\begin{aligned}
 StepSize &= W_{length} / (XY.x * XY.y), \\
 StepSize* &= localcameraDir.
 \end{aligned}
 \tag{4.3}$$

And we can divide $boxthickness$ by local slice thickness to get the maximum amount of sampling. Through both pre-computed values can fix the sampling rate as low as

possible. That suppresses redundant sampling between slices.

Sample Transfer Function

Until now, the essential parameters for the custom node were possessed. The next is performing UV movement to extract stored *alpha*. algorithm 4 depicts from sampling texture values to interpolation. The $\frac{inPos.xy}{XY}$ can get the unit UV for sampling 2D sequence map. Sampling which slice is decided by the z-axis. $ceil((inPos.z - W_min) / StepSize)$ can have current sample slice index in sequence map. "Convert1dto2d" [1] converts locating z frame that has been laid out in a 2D sequence map into a UV position. Both front slice UV and back slice UV are ready; we can apply it to a sampling function for interpolation. In the end, algorithm 4 returns interpolated α value between the front slice and back slice. Note that Tex (2D sequence map) is set as bilinear filter when the sample position does not hit the pixel center. Therefore, interpolation is tri-linear.

Algorithm 4: GetAlphaTF

Input: Texture Tex, XY, inPos, MaxSteps, StepSize

Output: α

```

1 zframe  $\leftarrow$  ceil((inPos.z - W_min) / StepSize);
2 zphase  $\leftarrow$  frac((inPos.z - W_min) / StepSize);
3 uv = frac(inPos.xy) / XY;
   /* Convert1dto2d provided by [1] tutorial */
4 curframe = Convert1dto2d(xy.x, zframe) / XY;
5 nextframe = Convert1dto2d(xy.x, zframe + 1) / XY;
6  $S_f$  = SampleLevel(Tex, uv + curframe);
7  $S_b$  = SampleLevel(Tex, uv + nextframe);
8  $\alpha$  = lerp( $S_f$ ,  $S_b$ , zphase);
9 return  $\alpha$ 

```

algorithm 5 sums up previous steps and provides the fundamental RM, the transmittance(1-opacity) is the index of early ray termination. The main loop included the illumination approach, which will be explained in the next subsection.

Algorithm 5: Ray-marching-based VRT

Input: Texture *Tex*, *XY*, *CurPos*, *MaxSteps*, *W_length*

```

1 StepSize = (W_length/XY.x*XY.y); StepSize *= CamVector;
2 transmittance  $\leftarrow$  1;
3 for  $i \leftarrow 0$  to MaxSteps do
4   if CurPos not in bounding_box or transmittance == 0 then
5     break;
6    $\alpha$  = GetAlphaTF(Tex, XY, CurPos, MaxSteps, StepSize);
7   if  $\alpha > air$  then
8     rgb = GetRGBTF( $\alpha$ );
9     if shadow then
10      for  $s \leftarrow 0$  to  $i$  do
11         $acc\_a$  += GetShadow(Tex, XY, ShadowPos, MaxSteps, StepSize);
12        ShadowPos += ShadowDir*ShadowStep;
13      end
14      if  $\alpha == bone$  then
15        color +=  $\exp(-acc\_a) * rgb * \alpha * transmittance$ ;
16        transmittance *=  $\alpha$ ;
17      else if  $\alpha == tissues$  then
18         $\alpha$  *= Density;
19        color +=  $\exp(-acc\_a) * rgb * \alpha * transmittance$ ;
20        transmittance *=  $\alpha$ ;
21      else if  $\alpha == lung$  then
22         $\alpha$  *= Density;
23        color +=  $\exp(-acc\_a) * rgb * \alpha * transmittance$ ;
24        transmittance *=  $\alpha$ ;
25      CurPos += StepSize;
26 end
27 opacity  $\leftarrow$  1 - transmittance ;
28 return color, opacity

```

4.2.1 Visual Enhancement

Deep Shadow Map

In algorithm 5, "Getshadow()" applied Equation 3.2 to accumulate amount of attenuation from the sampling position to the light source. The accumulated value was limited, which also took the current hitting sampling medium and transmittance into account to set the threshold.

Illumination models

Current work provided three different illumination methods: SM*, BP* and LAO*. SM* has been introduced in the above paragraph. BP* enables Blinn-Phong reflection, which is a standard approach in computer graphics as described in section 2.2.

To compute LAO, Equation 2.6 is revisited again, the formulation of g at closer look can be Equation 2.12. LAO is a way to enhance the ambient term A . Therefore, Equation 2.6 can be rewritten as:

$$A_x(x) = \frac{1}{M} \sum_{m=1}^M g_m \prod_{i=1}^{m-1} (1 - \alpha_i). \quad (4.4)$$

The ambient term A considers a local neighborhood, a sphere centered at voxel location x , $A_k(x)$ represents incident light ray with combined LAO $A(x)$. g_m represents the light contribution at sample point m along the ray. α_i is current TF opacity at sample position i . The $\frac{1}{M} \sum_{m=1}^M g_m$ is equal to 1 because of $1/M$ scaling. The whole Equation 4.4 can be simplified to $\prod_{i=1}^{m-1} (1 - \alpha_i)$, then all incident light ray are taken into account, the equation can be rewritten:

$$A(x) = \frac{1}{K} \sum_{k=1}^K \omega_k A_k(x), \quad (4.5)$$

where ω_k is the weighting term for directional ambient light. Ultimately, the $A(x)$ is associated with α TF and directional weighting. The directional light vectors can be randomly generated by "Rejection Sampling" described in algorithm 6.

Anti Artefacts

Three different methods were provided to suppress artifacts according to the needs,

Algorithm 6: Random directions with Reject Sampling

```

1 while TRUE do
2   x = RandomFloat(-1, 1);
3   y = RandomFloat(-1, 1);
4   z = RandomFloat(-1, 1);
5   if  $x*x + y*y + z*z > 1$  then
6     /* if(Dot(float3(x,y,z), N) < 0);
       down direction will be adjusted during sampling.          */
7     return normalized(float3(x,y,z));
8 end

```

the cause for the formation of artifacts was described in subsection 3.4.3. Supersampling anti-aliasing (SSAA) renders a draw with a higher resolution, which is used in the anti-aliasing border. SSAA splits each pixel into several sub-pixels and uses samples from each sub-pixel center to produce the output. In the implementation, a simple 2x2 grid algorithm was used. In the RM, each casting ray corresponds to each pixel of the screen. The only need is firing more than one ray for each pixel. algorithm 7 that computes image gradient in both horizontal and vertical directions cannot be applied inside dynamic branching and loops because dynamic loops are already included in our RM approach. Instead of that, the derivative was computed outside the loops and generate four ray entry positions. The pre-computed derivative positions can be used inside dynamic loops and marched as RM approach.

The Pre-integral approach is to reduce wood-grain artifacts; however, the valid proportion is under linear scalar function between two sampling points. Precisely our sampling is tri-linear. So, there are remaining wood-grain artifacts. Through adding a small offset to the sampling position of rays in the view direction can hide wood-grain artifacts, this technique is called Stochastic Jitter. The offset is determined by *Sv_Position* where the pixel position (x,y) is in normalized vertex coordinates.

Oversampling is the alternative to remove wood-grain artifacts. Oversampling in

Algorithm 7: 2x2 Ordered grid supersampling

```

/* per pixel screen space partial derivatives */
1 dx ← ddx(TexCoord.xy) * 0.25; /* horizontal offset */
2 dy ← ddy(TexCoord.xy) * 0.25; /* vertical offset */
/* supersampled 2x2 ordered grid */
3 inPos1 ← (inPos.xy + dx + dy, inPos.z);
4 inPos2 ← (inPos.xy - dx + dy, inPos.z);
5 inPos3 ← (inPos.xy + dx - dy, inPos.z);
6 inPos4 ← (inPos.xy - dx - dy, inPos.z);
7 return sets of inPos;

```

VRT means sampling at a higher rate than the intended rate, due to the adjusted sampling step size and amount that fix the sampling rate into the minimum. An incremental sampling rate between two sample slices can reduce the sudden change in the depth between neighboring opaque fragments belonging to the same pixel.

4.2.2 Branching Prevention

We assumed custom nodes had been minimized constants and less number of instructions so we can explore branch problem only. Branch statements(*for, if-else, while*) involve instruction jumps and cause the pipeline to stall on the GPU. The branches in the shader can be divided into static branches and dynamic branches. The difference is the value of condition statement takes static uniform variable or dynamic value, that is, the compiler knows the value only at run-time. Dynamic branching causes the GPU is done in batches; the GPU fails to maximize the use of simultaneous calculations. Instead, the calculations are performed twice with the overhead of creating "warp" and copying data, which should be avoided.

Although the branch statement is required in many scenarios, there are several ideas used to improve it:

1. The trick method skips the *if*: *step* replaces the *if* statement.

2. Optimized by compiler: The compiler can optimize branches under some condition. The statement contains only constants or uniform parameters. The static data and uniform data will not change, the compiler can judge and compile optimization.
3. Unroll code: For example, after testing, *if* can generate two branches; it may be faster to execute the instructions of both branches than to use *if*.

Also, note that unless the branch is skipping over significantly more lines of code than a single texture read will probably reduce performance and not improve it.

4.3 Octree

Given dataset, each CT slice contains at least 40% empty space. Using a hierarchical structure to perform leaping empty space have been proved that can improve performance. The concept of the Octree has been introduced in subsection 3.4.4. This section includes approaches of the Octree construction and shifting UV coordinates to traverse the tree.

4.3.1 Tree Construction

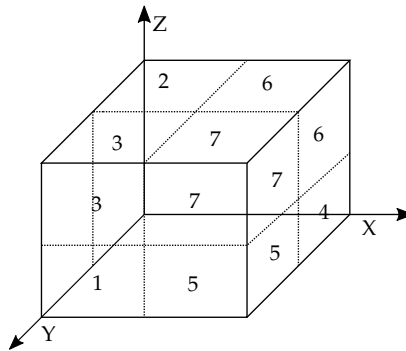


Figure 4.5: Labeled nodes of octree in Z-up left-handed coordinates

The strategy implements BONO on constructing and saves tree as texture type:

1. Saving structure: Each pixel contains RGB with uint8 data-type. RG stores the child index of the current node, and B stores the average density of sub-space. Extra memory for traversal is allocated to record the number of slices in sub-space. Therefore, each child holds three pixels, each row stored eight children, and the column contains the total number of nodes.
2. Z-up left-handed coordinates: The construction should be base on UE4 coordinate system. Fig. 4.5 depicts labeled nodes in Z-up left-handed coordinates.
3. Termination criteria: Maximum Octree depth or minimum/maximum density matches threshold in sub-space.

Condition	[R, G, B]	[R, G, B]	[R, G, B]
Leaf node	[0,0,0]	[slices,0,0]	[0, 0, density]
Can be subdivided	[0,0,0]	[0,0,0]	[ptr, idx, density]

Table 4.2: Each child contains three pixels, the memory usage is based on two conditions.

Leaf nodes: the average of density below or higher thresholds, or tree reaches the maximum depth. Node gives the number of slices and average of density in current sub-space; Nodes can be subdivided: Node provides pointer of the row which is the location of the current node's children.

algorithm 8 explains whole Octree construction. The basic idea was that if a node can be subdivided, then AABB values of the current sub-space was stored in its child(label 0), then allocated new "children row" and concatenated into constructing tree. The AABB values were extracted for further subdivision when column_idx and row_idx visited it. Until row_idx visited all rows(child nodes), the construction terminated. The saving policy of nodes can refer to Table 4.2.

4.3.2 Traversal

The traversal is based on [57] top-down arithmetic. There are a few things need to beware: To extract accurate RGB value, need to load texture as linear and compute

Algorithm 8: BONO Construction Procedure

```

1 Begin
2   struct {
3     |   float r,g,b;
4   } Node;
5   Function Calculate_density(vmin, vmax, imgs_arr, child_idx):
6     |   avg  $\leftarrow$  0 ;
7     |   /* Get the accurate voxels occupying space */
8     |   front_idx  $\leftarrow$  floor((vmin[2] - W_min) / thickness);
9     |   back_idx  $\leftarrow$  floor((vmax[2] - W_min) / thickness);
10    |   space  $\leftarrow$  (back_idx - front_idx) * abs(vmax[1] - vmin[1]) * abs(vmax[0] - vmin[0]);
11
12    |   r1, r2, c1, c2 = computeRowColumn(vmin, vmax, child_idx); /* Compute column
13    |   and row of start and finish on pixel in sub-space. */
14    |   for depth  $\leftarrow$  front_idx to back_idx do
15    |   |   img  $\leftarrow$  imgs_arr[depth];
16    |   |   for row  $\leftarrow$  r1 to r2 do
17    |   |   |   rows  $\leftarrow$  img[row];
18    |   |   |   avg + = sum(rows[c1 : c2])
19    |   |   end
20    |   end
21    |   avg / = space;
22    |   return avg, slices;
23
24   Function InitOctree(img_arr):
25     |   child  $\leftarrow$  TRUE;
26     |   Octree  $\leftarrow$  {Node}i=125;
27     |   row_idx  $\leftarrow$  0; column_idx  $\leftarrow$  0;
28     |   /* Initial root UVW box size */
29     |   Octree[0][0] = [0,0, W_min]  $\leftarrow$ ; Octree[0][1] = [512,512, W_max];
30     |   idx  $\leftarrow$  -1;

```

```
26
27 while child do
28     xyz  $\leftarrow$  Octree[0][0:1];
29     for row_idx  $\leftarrow$  0; column_idx < 24; column_idx += 3 do
        /* compute labelled child AABB box and label, the partition fills
           W to 512 */
30     sub_xyz, idx = computesubbox(xyz, row_idx);
31     subdivide  $\leftarrow$  abs(sub_xyz[0][0] - sub_xyz[1][0]);
32     avg_density, slices = Calculate_density(sub_xyz[0], sub_xyz[1], img_arr, idx);
33     if subdivide  $\leq$  subdivide_threshold || ceil(avg_density)  $\leq$ 
        minDensity || ceil(avg_density)  $\geq$  maxDensity then
34         Octree[row_idx][column_idx + 1] = [slices, 0, 0];
35         Octree[row_idx][column_idx + 2] = [0, 0, avg_density];
36     else
        /* Create next child index, concatenate into parent node */
37     next_child  $\leftarrow$  {Octree}i=125;
38     child_ptr = (Octree.shape[0] + 1)%255;
39     child_idx = int((Octree.shape[0])/255);
40     next_child[0][0 : 1]  $\leftarrow$  sub_xyz;
41     Octree[row_idx][column_idx + 2] = [child_ptr, child_idx, avg_density];
42     Octree  $\leftarrow$  concatenate((Octree, next_child), axis = 0)
43     row_idx ++;
44     if row_idx  $\geq$  Octree.shape[0] then
45         child  $\leftarrow$  FALSE
46     saveImage(Octree);
47 End
```

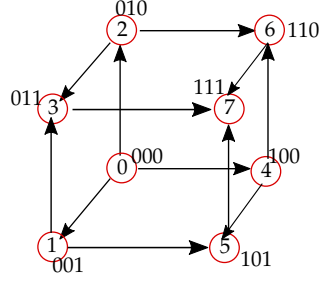


Figure 4.6: The diagram shows the relationship between the initial node and sub-node in z-up left-hand coordinates. The digits from the first to the last in order indicates X-axis, Y-axis and Z-axis, and 0 and 1 indicate the negative and positive direction respectively.

unit UV; otherwise, the stored values of the pointer will be incorrect; Revelles J. et al. [57] used left-handed Cartesian coordinates. The method adopted was to assign the binary bits 000-111 from 0-7 labeled nodes and through an associated bit to find the next intersection node. The order of the child nodes is fixed, Fig. 4.6 depicts the relationship between sub-node and initial node after converting into UE4 coordinate system.

The Octree traversal is described in Fig. 9, the entire procedure is similar to the original proposal, except different coordinates and returning leaf node condition. As long as shader successfully traversed the tree and sampled accurate pixel values, the last step was to apply the conditional policy(Fig. 4.2). When RG values in the third pixel of the child are 0, that means leaf node; however, B represents average density, which determines whether sub-space can be skipped. If B is nearly empty, the function returns the No. jumping slices, and -1 tells the custom node to skip the current sub-space. On the other hand, the non-empty space condition still returns the number of slices to the ray, which marches until it visits the next sub-space.

Algorithm 9: Octree traversal

```
1 Begin
2   Function First_node( $tx0, ty0, tz0, txm, tym, tzm, rayDir$ ):
3     node_idx  $\leftarrow$  0;
4     fareset  $\leftarrow$  max( $tx0, \max(ty0, tz0)$ );
5     if YZ plane then
6       if  $tym < tx0$  then
7         | node_idx | = 2;
8       if  $tzm < tx0$  then
9         | node_idx | = 1;
10      if  $rayDir.x < 0$  then
11        | node_idx | = 4;
12    else if XZ plane then
13      if  $txm < tz0$  then
14        | node_idx | = 4;
15      if  $tym < tz0$  then
16        | node_idx | = 2;
17      if  $rayDir.y < 0$  then
18        | node_idx | = 1;
19    end
20    else if XY plane then
21      if  $txm < ty0$  then
22        | node_idx | = 4;
23      if  $tzm < ty0$  then
24        | node_idx | = 1;
25      if  $rayDir.z < 0$  then
26        | node_idx | = 2;
27    end
28    return node_idx;
```

```
26
27 Function OctreeProcess(OctreeTex, rayO, rayD, UVWbox0, UVWbox1, uv):
28     curV  $\leftarrow$  1, kid  $\leftarrow$  -1;
29     IDX  $\leftarrow$  0, uv_  $\leftarrow$  0 ;
30     for depth  $\leftarrow$  0 to maxDepth do
31         /* compute entry/exit plane and t1, t0 */
32         if t1  $\leq$  t0 || t1  $\leq$  0 then
33             break
34         uv_.y  $\leftarrow$  uv.y*curV;
35         /* shift to next node */
36         kid = First_node(entry_plane.xyz, middle_plane, rayD);
37         if kid then
38             uv_.x = WhichKid(uv.x, kid);
39             node  $\leftarrow$  SampleLevel(OctreeTex, uv_, 0);
40             if node.r == 0 and node.g == 0 then
41                 /* leaf node */
42                 uv_.x = WhichKid(uv.x, kid);
43                 /* movement stores in second pixel */
44                 movements = SampleLevel(OctreeTex, uv_, 0);
45                 if node.b == empty then
46                     IDX  $\leftarrow$  float2(-1, movement);
47                 else
48                     IDX  $\leftarrow$  float2(1, movement);
49                 break;
50             else
51                 boxUVW0 = compute sub-space UVW(kid, UVWbox0, UVWbox1);
52                 curV  $\leftarrow$  node.r * 255 + node.g * 2552;
53     return IDX
```

5 Results

5.1 Visualization

Data_ID	No. Slices	Device	Source	Data_ID	No. Slices	Device	Source
p08_1b14	47	CT	Anonymous	p01_04	150	CT	Anonymous
p03_78e	81	CT	Anonymous	p09_24f	165	CT	Anonymous
p02_360	100	CT	Anonymous	p03_788	256	CT	Anonymous
p09_24d	119	CT	Anonymous	p05_f21	279	CT	Anonymous
p06_12f9	134	CT	Anonymous	p07_16c1	317	CT	Anonymous

Table 5.1: Used DICOM Files covered nine variant patients, the Data_ID was sorted by the number of slices from low to high. The data from patient 01-03 were used for section 5.1, and the entire dataset were used for evaluation part.

The visual results were compared to two open-source software systems: RadiAnt DICOM Viewer and VTK(Visualization Toolkit). VTK is mainly used for 3D computer graphics, image processing, and scientific visualization. It provides technical support for researchers to engage in visual application development. Table 5.1 presents eight obtained anonymous patient DICOM data. Each folder contains thoracic cavity scans of the body with different sections, amount, and thickness.

5.1.1 Ground Truth

Four different data-sets comparison results with RadiAnt and VTK are shown in: Fig. 5.1, Fig. 5.2, Fig. 5.3 and Fig. 5.4. The order is sorted from low to high according to the number of slices. The LAO* had clear contour and depth perception with different

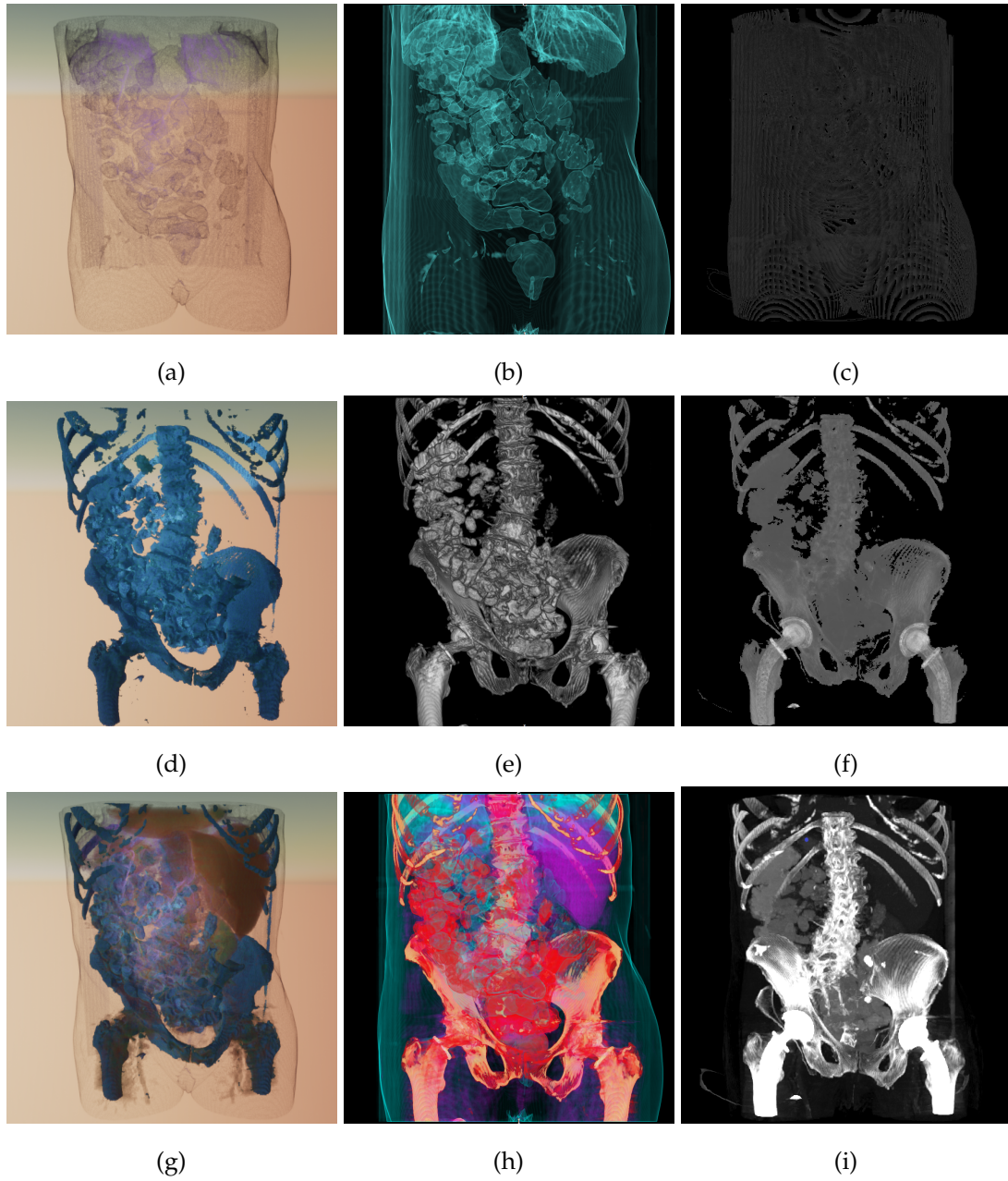


Figure 5.1: The ground-truth comparison with RadiAnt and VTK used the data ID: p03_78e. The columns show different implementation results: Current work approach LAO*((a), (d), (g)), RadiAnt((b), (e), (h)) and VTK((c), (f), (i)). The rows represent a different ROI: lung and skin((a) - (c)), bone((d) - (f)) and all((g) - (i)) in order.

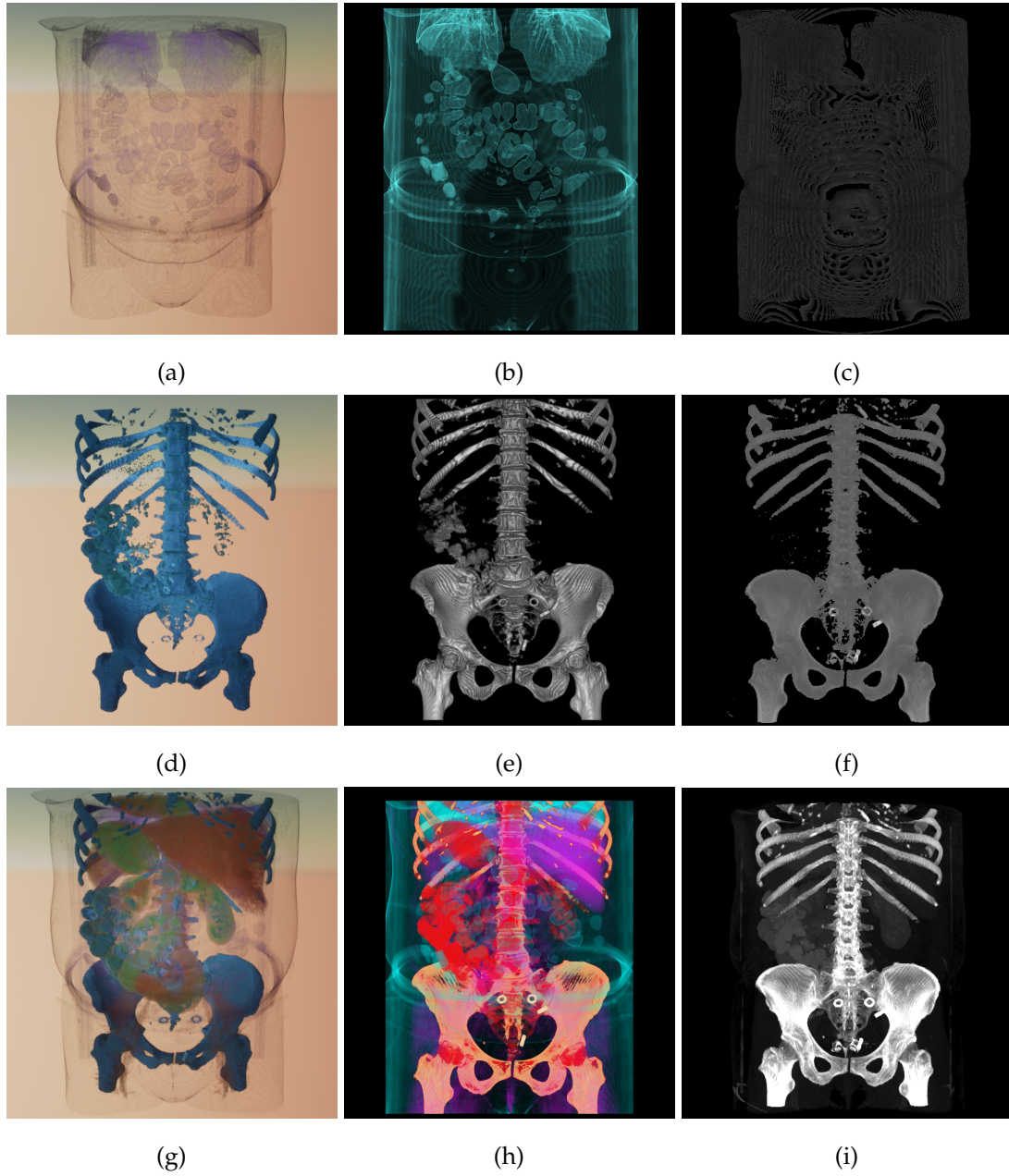


Figure 5.2: The ground-truth comparison with RadiAnt and VTK used the data ID: p02_360. The columns show different implementation results: Current work approach LAO*((a), (d), (g)), Radiant((b), (e), (h)) and VTK((c), (f), (i)). The rows represent a different ROI: lung and skin((a) - (c)), bone((d) - (f)) and all((g) - (i)) in order.

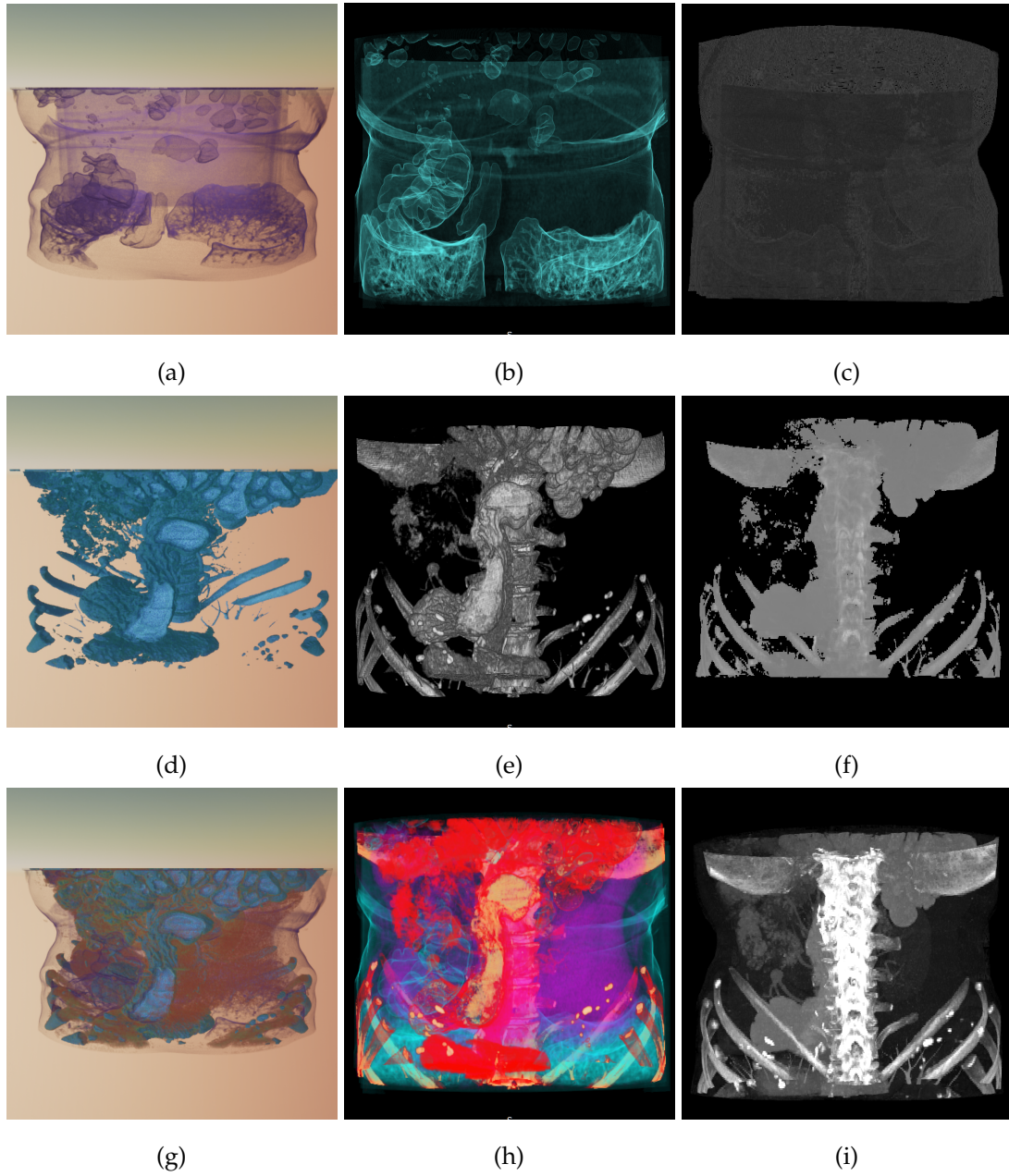


Figure 5.3: The ground-truth comparison with RadiAnt and VTK used the data ID: p01_04.

The columns show different implementation results: Current work approach LAO*((a), (d), (g)), Radiant((b), (e), (h)) and VTK((c), (f), (i)). The rows represent a different ROI: lung and skin((a) - (c)), bone((d) - (f)) and all((g) - (i)) in order.

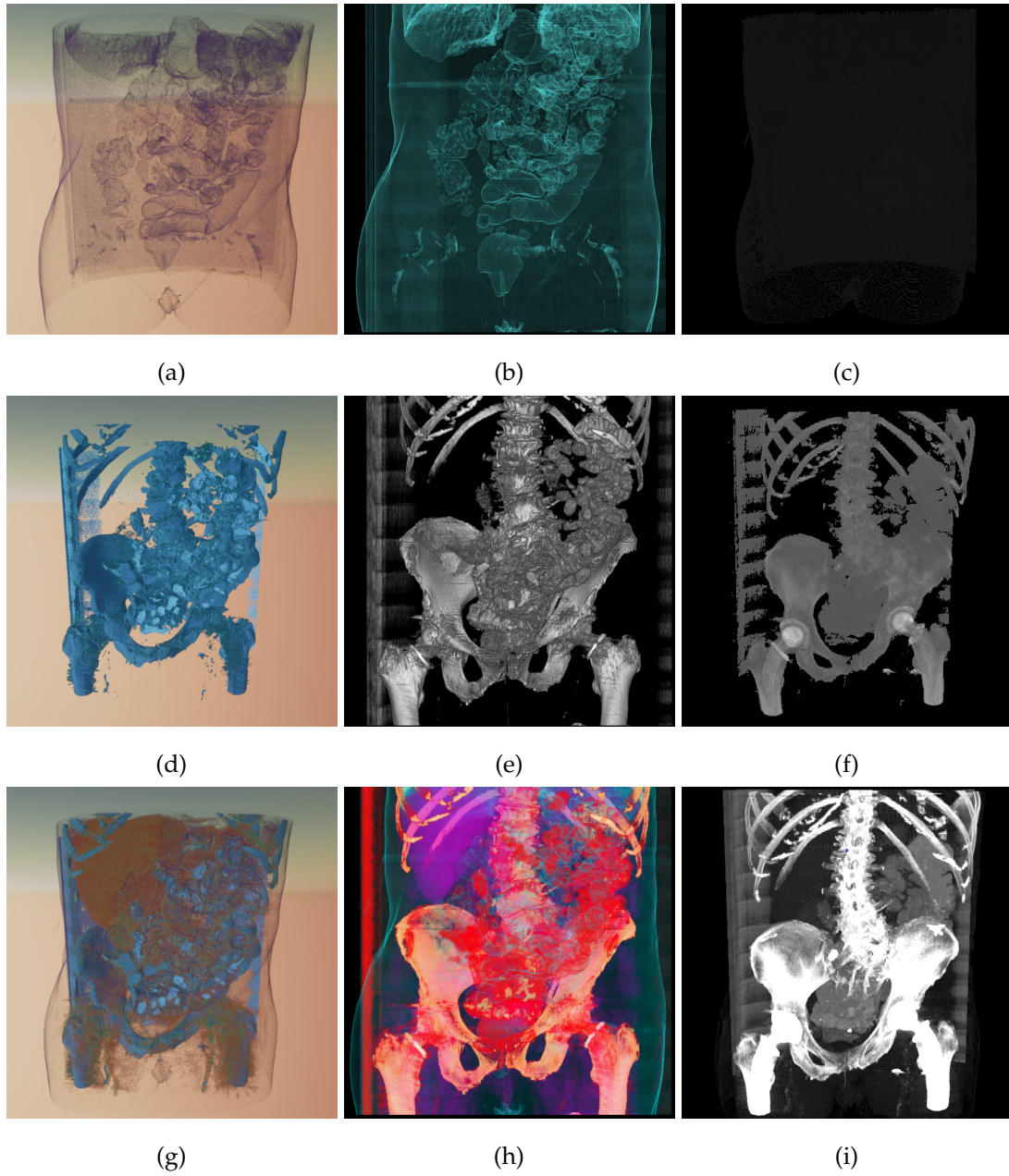


Figure 5.4: The ground-truth comparison with RadiAnt and VTK used the data ID: p03_788.

The columns show different implementation results: Current work approach LAO*((a), (d), (g)), Radiant((b), (e), (h)) and VTK((c), (f), (i)). The rows represent a different ROI: lung and skin((a) - (c)), bone((d) - (f)) and all((g) - (i)) in order.

visual modes. As Fig. 5.2(h)(i), Fig. 5.3(h)(i) and Fig. 5.4(h)(i)) shown, the visual results of RadiAnt and VTK displayed overlapped organs without comprehensive relative position for bone and tissues mode.

The rendering algorithm of VTK is maximum intensity projection (MIP) [36], which only collects maximum emissions along with casting rays. The advantage of MIP is all contours are visible equally but are independent of their depth order.

5.1.2 Volume Rendering for Art versus Scientific Data

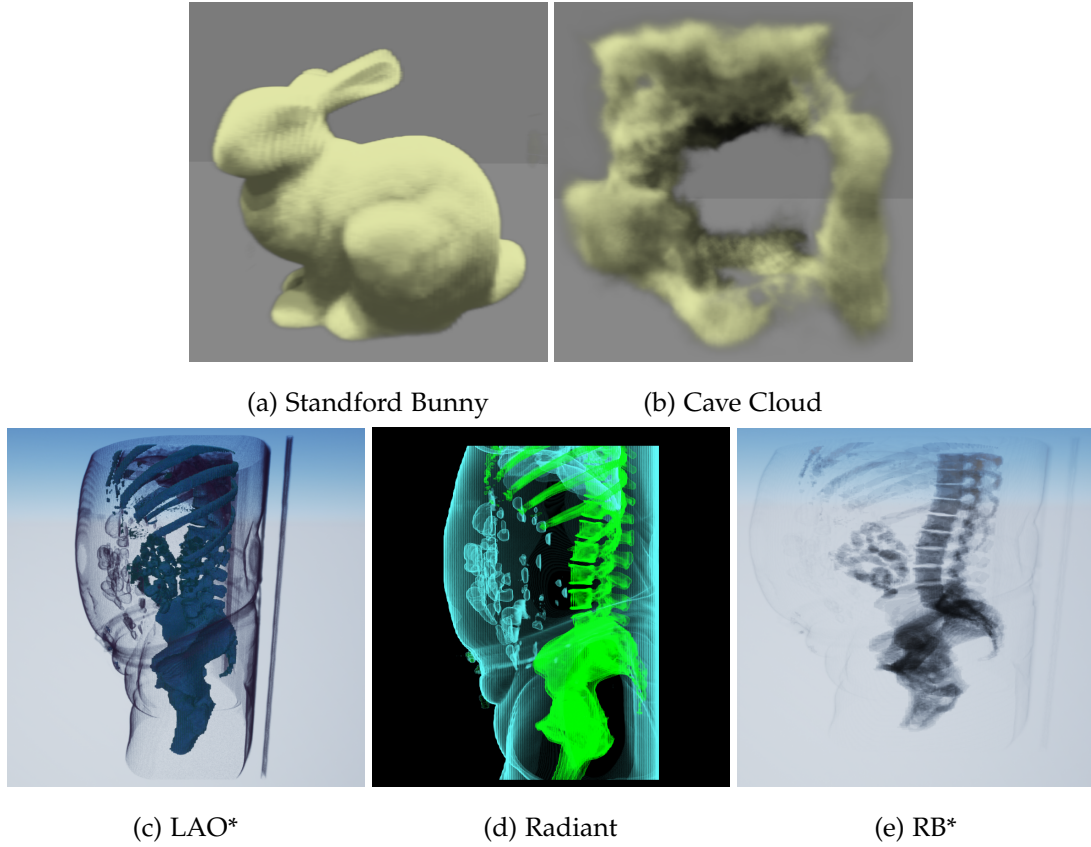


Figure 5.5: (a),(b) use RB* to perform volume rendering art effects. Medical dataset p02_360 is used for visual comparison (c)-(e). (c) enables LAO*, (d) enables RadiAnt as ground-truth. (e) enables RB*.

We have mentioned that the graphic engine usually utilizes volume rendering

technique to implement complex art effects(e.g., cloud, fog and fire). Here, the tutorial by Ryan Brucks [1](RB*) was followed to perform volumetric rendering in UE4 and it was compared with Radiant and our approach(LAO*).

Volume rendering Stanford bunny and cloud (Fig. 5.5(a)(b)) display decent visual effect. When the same algorithm was applied on the p02_360 as an example, the Fig. 5.5(e) shows the inappropriate blending color and incorrect organ positions, that was compared with ground-truth Fig. 5.5(d) from Radiant. After considering tag values from DICOM and TF design, Fig. 5.5(c) illustrates better representation of medical data.

5.1.3 Different Illumination Result

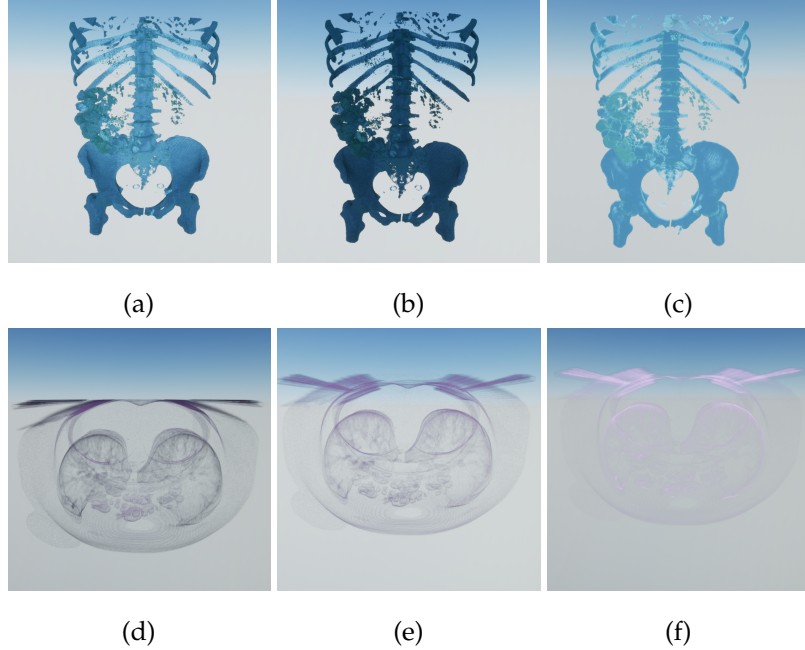


Figure 5.6: Ray-marching-based VRT uses p02_360 with different illumination methods. (a)(d) LAO* (b)(e) BP* (c)(f) SM*. The figures only consider one light source. Position of light source is behind the rendering target.

The current work implemented various illumination approaches (LAO*, BP* and SM*) to enhance visualization. The Fig. 5.6 displays the visual results with different

ROI. SM* can have proper visualization with high opacity medium (Fig. 5.6(c)). When it encountered low opacity tissues(e.g. skin and lung) the boundary disappeared (Fig. 5.6(f)). The Fig. 5.6(e) shows that BP* solved missing boundary. However, when light position is behind the RT, as Fig. 5.6(b) display, RT became totally dark. The Fig. 5.6(a) displays that the bone was visible in detail with LAO*. The LAO* prevented fully occlusion, also kept the advantage of BP*.

5.1.4 Removed Artifacts

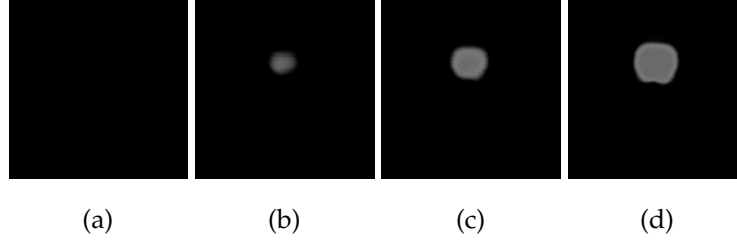


Figure 5.7: The first 4 slices take from p02_360. Due to the longitudinal section, pre-integration can easily get empty pair.

The current thesis provided two methods to remove wood-grain artifacts. Due to each sample, step length and maximum steps have been adjusted base on the scan thickness and ray direction. In the beginning, this adjustment saved performance and simulates the correct organ position. Because of the low sampling rate, the Pre-integration cannot take its advantage to remove artifacts. Especially implementing with longitudinal section scan dataset(e.g., p03_78e, p02_360). In Fig. 5.7(a)-(d), there exists 0 scalar value(air) in the pair sample points with adjusted step length. That means no extra samples in the slab to neutralize the empty scalar values.

Therefore, we suggested Stochastic Jitter to remove wood-grain artifacts without increasing the sampling rate. The example explains that different sections and slices-density caused the remaining artifacts. Fig. 5.8 (a)-(d) were rendered with lower amount of longitudinal-section slices. Fig. 5.8 (e)-(h) were implemented with higher number of cross-section slices. (a) and (e) were rendered without any approach to

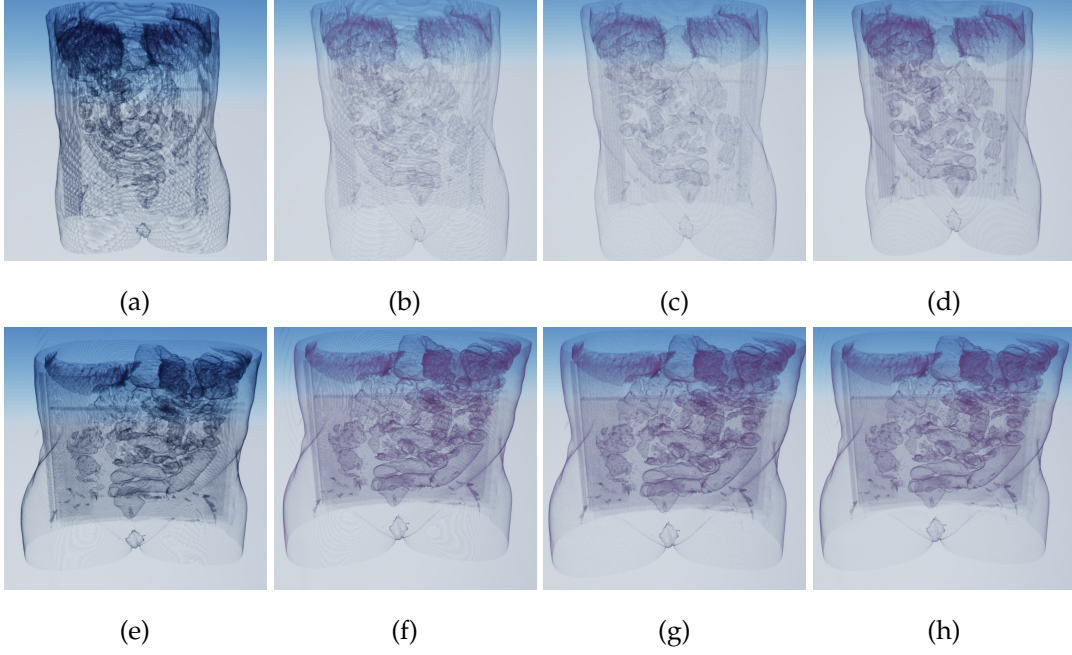


Figure 5.8: The visual comparison of handling artifacts in an efficient way. (a)-(d) use p03_78e with longitudinal-section 81 slices and 3 mm slice thickness. (e)-(h) use p03_788 with cross-section 256 slices and 1.5 mm slice thickness. (a)(e) Without any visual optimization. (b)(f) Pre-Integral. (c)(g) Pre-Integral + Jitter. (d) Pre-Integral + 3 times sampling rate. (h) Pre-Integral + 2 * sampling rate.

remove wood grain, but (a) displays visible artifacts. (b) and (f) both had remaining artifacts after using Pre-integral. Without adding a sampling rate, Stochastic jitter can suppress woos grain, as seen in (c) and (g). On the other hand, without using jitter, the lower amount of longitudinal-section slices (d) needed a higher number sampling rate than (h) to remove wood-grain artifacts.

Regarding of anti-aliasing, the Fig. 5.9(a) had smoother edge resolution from 2x2 sub-pixel pattern in comparison to Fig. 5.9(b), which had notable edge gradient resulting.

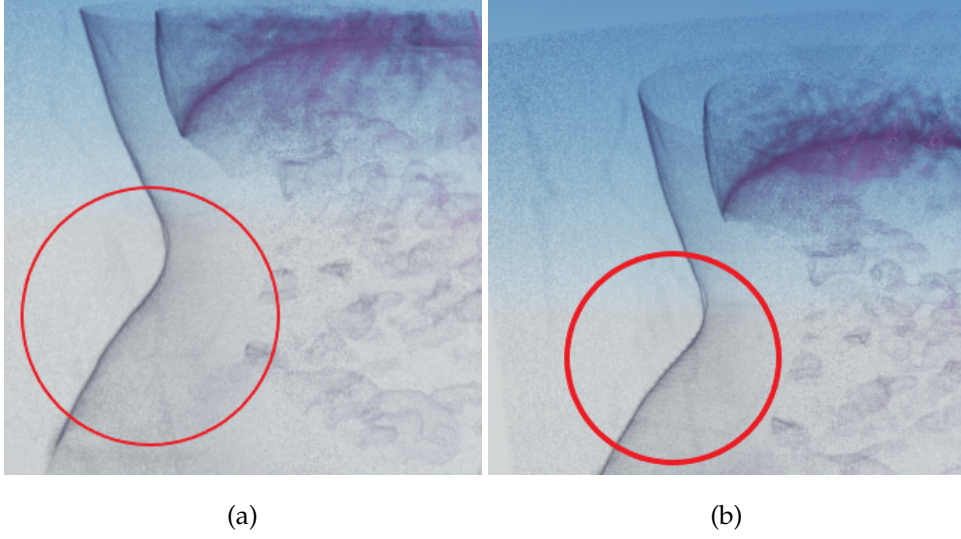


Figure 5.9: The visual comparison of removing anti-aliasing. (a) 2x2 ordered grid SSAA, (b) without SSAA.

5.2 Performance

The performance evaluation was done under Windows 10 using an Intel Core i7 4790, 32GB RAM, and a Nvidia Titan V. The Unreal Engine Version is 4.22.3. The testing used eight anonymous patients with ten different amount of slices (See Table 5.1). The amount of slices ranged from 47 to 317. Due to the restriction of the maximum size of importing texture in Unreal Engine, we resized two datasets below or equal to 8192x8192.

5.2.1 High-quality Illumination versus Performance

In the previous visual comparison, RB* approach cannot reach high-quality visualization with medical images. Current work provides decent visual quality. This section analyses whether our illumination methods need to trade performance off. The theoretical complexity of our DVR algorithm (for all illumination methods) is $O(m * n)$ where m is the fixed resolution of the viewport and n is the number of samples, per ray, which is the dynamic and is mainly influenced by the number of

slices, sampling rate, and Octree. Without increasing sampling rate, n is correlated to the number of slices. The three illumination methods unfolded the loop and pre-computed shadow values inside the 2D sequence map; therefore, complexity regarding their Big-O notation is the same. PS executes under the GPU, and sampling is the process by which the GPU reads the texture information from memory. That typically needs between 100 and 1000 clock cycles, which is hundreds of times more than usual instruction execution. Algorithm utilized tri-linear interpolation to sample value. Each sampling slice used a bilinear filter to extract value and then interpolated with the next sampling slice. Hence, sampling is crucial in runtime. On the other hand, RB* does not use TF; its shadow requires large sampling that has a big impact on performance.

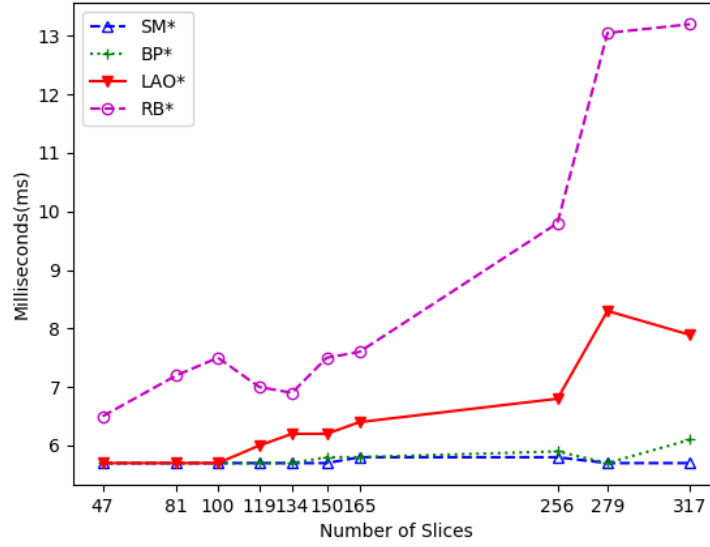


Figure 5.10: Performance comparison with various amount of slices: Three illumination methods(LAO*, SM*, BP*) and RB* without SSAA.

As Fig. 5.10 illustrates, as the number of slices increasing, the performance dropped dramatically using RB*. During this evaluation, the 279 and 317 slices were resized to 8192x5734 and 8192x6553, respectively. Both have a smaller resolution than 256 slices.

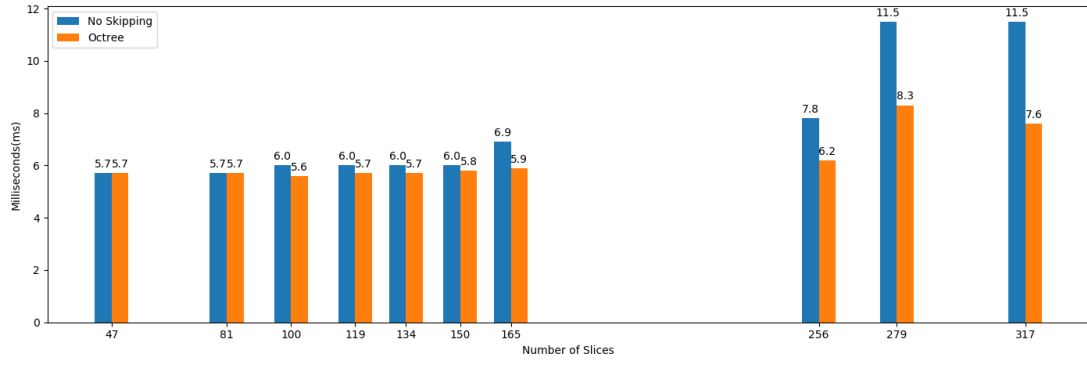
There was no obvious difference between the three illumination methods when the number of slices was below 100. The LAO* dropped slightly from 100 to 256 number of slices. When rendering 279 slices, RB* slumped up to 13 ms, BP* and LAO* slightly decreased but still kept acceptable performance, and SM* remained.

5.2.2 Octree

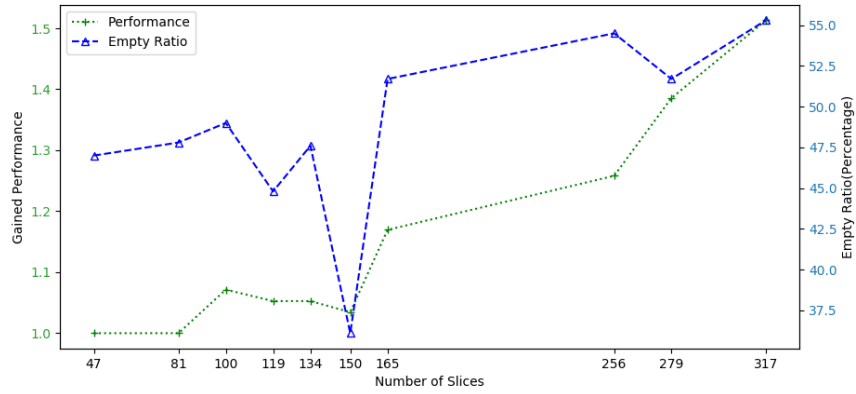
In general, the performance had visibly dropped when rendering with a high amount of slices and complicated illumination methods. Instead of only performing early ray termination, we also utilized hierarchical data structure(Octree) to leap blank space. LAO*, when rendering below 100 amount of slices, had high performance. To observe the advantage of the Octree, we implemented SSAA for evaluation.

The Fig. 5.11(a) analyzes performance improvement with Octree under SSAA condition. The gained performance had a slight correlation with the number of slices when the slices were higher than 150. The main purpose of Octree in the current thesis was to skip empty space. Therefore, for better understanding, Fig. 5.11 (b) shows the correlation between improved performance and empty ratio. The gained performance of 279 slices was higher than 256 slices even though its empty ratio was lower. The overall improved performance cannot have a positive correlation with the empty ratio, which was also influenced by the amount of slices/image size. This phenomenon can be observed in Fig. 5.11(b), the 279 slices had a smaller empty ratio than 256 slices, but its gained performance was higher than 256. Because the image size of the amount 256 was far greater than that of amount 279. The statics information for evaluation can be seen in Table 5.2.

Although the performance can be improved via Octree, the scalar value of skin was nearly empty and average density as subdivision standard caused low opacity medium to be skipped under certain viewing angle as shown in Fig. 5.12(b).



(a)



(b)

Figure 5.11: (a)Performance comparison: Under 2x2 SSAA and LAO* compared various amount of slices with/without skipping empty space (b) The improved performance and empty space ration from (a).

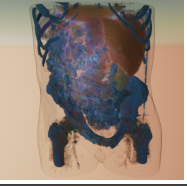
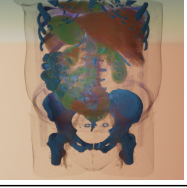
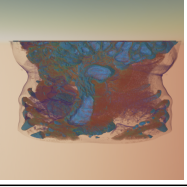
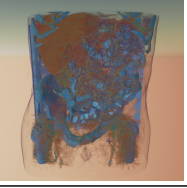

DataSet	Description	Datasize and type	Empty ratio	Millisecond(ERT)					
				SSAA+LAO*		No SSAA			
				no skipping	octree	SM*	BP*	LAO*	RB*
	p03_78e 512x512x81 ST: 3mm	Images: 3.9MB(8bit) Octree: 30.5KB(8bit)	76.1%	5.7	5.7	5.7	5.7	5.7	7.2
	p02_360 512x512x100 ST: 3mm	Images: 4.2MB(8bit) Octree: 60.9KB(8bit)	60.9%	6	5.6	5.7	5.7	5.7	7.5
	p01_04 512x512x150 ST: 1.5mm	Images: 4.5 MB(8bit) Octree: 60.9KB(8bit)	40.3%	6	5.8	5.7	5.8	6.2	7.6
	p03_788 512x512x256 ST: 1.5mm	Images: 6.8MB(8bit) Octree: 60.9KB(8bit)	57.1%	7.8	6.2	5.8	5.9	6.8	9.8
	p08_1b14 512x512x47 ST: 2mm	Images: 1.3 MB(8bit) Octree: 76.1KB(8bit)	49%	5.7	5.7	5.7	5.7	5.7	6.5

Table 5.2: Dataset statistics for the volumes used the evaluation of methods. All the evaluations are under early ray termination(ERT). This table contains whether leaping blank space with SSAA octree, and novel three illumination methods(SM*, BP*, LAO*) compare with RB* provided by UE4

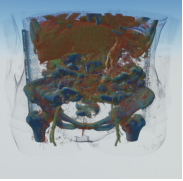
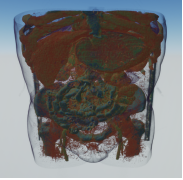
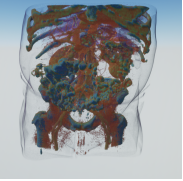
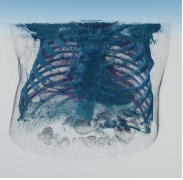
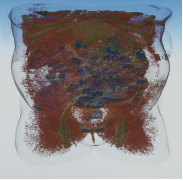
DataSet	Description	Dataseize and type	Empty ratio	Millisecond(ERT)					
				SSAA+LAO*		No SSAA			
				no skipping	octree	SM*	BP*	LAO*	RB*
	p06_12f9 512x512x134 ST: 3mm	Images: 3MB(8bit) Octree: 60.9KB(8bit)	51.1%	6.0	5.7	5.7	5.7	6	6.9
	p07_16c1 512x512x317 ST: 1.5mm	Images: 5.3MB(8bit) Octree: 76.1KB(8bit)	58.5%	11.5	7.6	5.75	6.1	7.9	13.2
	p09_24f 512x512x165 ST: 3mm	Images: 3.4MB(8bit) Octree: 60.9KB(8bit)	54.7%	6.9	5.9	5.8	5.8	6.3	7.6
	p09_24d 512x512x119 ST: 3mm	Images: 7.7 MB(8bit) Octree: 30.5KB(8bit)	48.9%	6	5.7	5.7	5.7	6	7
	p05_f21 512x512x279 ST: 1.5mm	Images: 5.4 MB(8bit) Octree: 38.1KB(8bit)	54.2%	11.5	8.3	5.7	5.7	8.3	13.05

Table 5.3: Dataset statistics for the volumes used the evaluation of methods. Continue from previous page

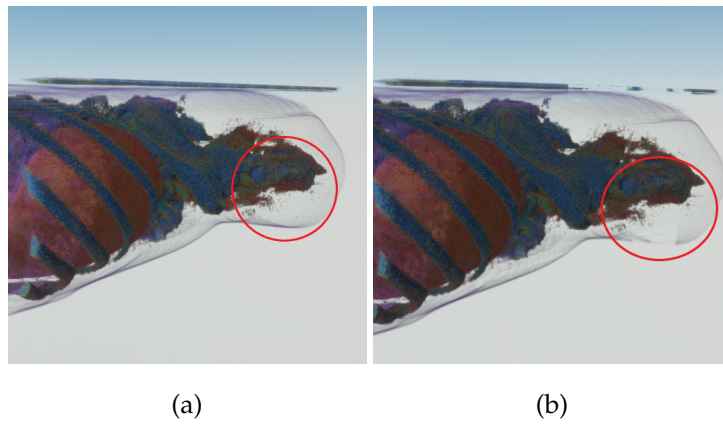


Figure 5.12: Visual comparison with non-Octree and Octree (a) enable SSAA and LAO* (b) enable SSAA, LAO* and Octree.

6 Conclusion & FutureWork

6.1 Conclusion

The current thesis successfully implements DVR into UE4, which is fast enough for real-time viewing and Virtual Reality, and at the same time, it provides high visual quality.

From the blended window algorithm, knowledge from radiology to segment ROI was utilized. The most important tissues, such as organs and bones, were successfully segmented out. Similarly, the scalar value can be extracted and used for opacity TF and shadow directly.

Different illumination methods were deployed to enhance visual quality. Through branching prevention and pre-computed opacity TF and under high-level LAO rendering even performs fast.

About performance and artifacts, both jitter and increasing sampling rate can remove wood-grain artifacts. Stochastic jitter only needs to be applied before starting the sampling loop, which not only saves a lot of computing but also suits for any CT section. Although the SSAA suppressed aliasing, most of the time, the aliasing artifacts were not notable. SSAA can be an optional function for visual enhancement. Octree was constructed successfully, which rendered faster than without deploying octree under SSAA condition.

Through well understanding DICOM information, current work can pre-compute step length and the maximum number of steps for each viewing ray to prevent redundant sampling and keep the correct model shape. In performance comparison, RB*[1]

simulated realistic shadow and implemented with plenty of sampling instruction due to which the number of slices impacts the performance. The overall LAO* can have better performance and visual precision than RB* for rendering the variant number of medical data.

Regarding visual comparison with the open-source library and DICOM application, high-quality screenshots were provided in section 5.1. In the visual comparison with VTK, VTK used different blending method; the result only can prove that the MIP cannot segment high complex structure. RadiAnt provided different color for tissues; However, there were no proper shadow mapping on them (only bone mode has shadow), and it was hard to tell depth, on glowing organs. LAO did not have the above shortcomings and gave a clear internal structure in semi-transparency.

Therefore, as shown in current works, both visualization and performance tell, it is possible doing VRT under the game engines without relying on the open-source library.

6.2 Future Work

There are numerous opportunities for future work.

The raw data through pre-processed then was saved as images. Because of texture format, the intuitive way was to use the material editor, but this method can also be a restriction. The maximum number of importing size for texture is 8190x8190 in Unreal Engine. Via resizing 2D sequence map may lose resolution, which we did for p07_16c1 and p05_f21, the alternative way is adjusting custom node, that can read multiple 2D sequence textures to solve it temporarily. The suggestion can be store cleaned data as a text file then interpreted as a texture inside UE4.

Regarding TF for medical diagnosis needs further improvement. The current approach can acquire essential ROI, and the viewer cannot see a particular organ or a complicated structure. That can be improved by using 2D/multi-dimension TF. The data we possessed only scanned chest to limbs because the advantage of considering

higher dimensional TF can handle a variant range of HU such as the head. The other option to optimize ROI is performing supervised machine learning [59] to classify multi-material. The material topology of objects [60] allowed viewers to select render objects and depict adjacent materials in a volume. For further photo-realistic illumination, the cinematic rendering technique can also be applied in VRT.

The current Octree aimed to skip empty space. The subdivision standard caused a low-opacity medium to be skipped. We can see empty space as an irregular shape and apply irregular grid algorithm [61] to build the spatial structure. This method not only skipped empty space accurately but also merged the same medium(tissue), which can reduce the cost of ray sampling.

List of Figures

2.1	Absorption model	8
2.2	Traveling Light	9
2.3	Blinn-Phong model	12
2.4	DICOM data elements	15
2.5	Hounsfield Unit Scale	16
2.6	Adjusted windows with preset values	17
3.1	My Program Flow	18
3.2	Custom UVs	20
3.3	Pre-Integral Slab	24
4.1	Implementation pipeline	27
4.2	Histogram of the first slice	29
4.3	Native blended window algorithm	30
4.4	Histogram comparison of different blended windows approach.	31
4.5	Z-up left-handed coordinates	39
4.6	The relationship between the initial node and sub-node in z-up left-hand coordinates.	43
5.1	The ground-truth comparison with RadiAnt and VTK used the data ID: p03_78e.	47
5.2	The ground-truth comparison with RadiAnt and VTK used the data ID: p02_360.	48

5.3	The ground-truth comparison with RadiAnt and VTK used the data ID: p01_04.	49
5.4	The ground-truth comparison with RadiAnt and VTK used the data ID: p03_788.	50
5.5	The visual comparison with RB*[1] and our approaches.	51
5.6	Visual comparison with different illumination methods.	52
5.7	The first 4 slices take from p02_360. Due to the longitudinal section, pre-integration can easily get empty pair.	53
5.8	Using different methods to remove wood-grain artifacts.	54
5.9	Comparison of anti-aliasing.	55
5.10	Performance comparison with various amount of slices	56
5.11	Performance comparison with Octree and non-Octree	58
5.12	Visual comparison with non-Octree and Octree	61

List of Tables

2.1	An description for used tags table	17
3.1	Non-full Octree	26
4.1	Used Tag Table Values which were extracted from patient ID: 01_13 .	28
4.2	Stored nodes policy	40
5.1	Used DICOM Files	46
5.2	Dataset statistics-1	59
5.3	Dataset statistics-2	60

Bibliography

- [1] RyanB. *Authoring Pseudo Volume Textures*. Oct. 2016. URL: <https://shaderbits.com/blog/authoring-pseudo-volume-textures>.
- [2] A. Cantatore and P. Müller. "Introduction to computed tomography". In: 2011.
- [3] G. D. Rubin. "Computed Tomography: Revolutionizing the Practice of Medicine for 40 Years". In: *Radiology* 273.25 (Oct. 2014). DOI: <https://doi.org/10.1148/radiol.14141356>.
- [4] E. H. Dillon, M. S. van Leeuwen, M. A. Fernandez, B. C. Eikelboom, and W. P. Mali. "CT angiography: application to the evaluation of carotid artery stenosis." In: *Radiology* 189.1 (Oct. 1993). DOI: <https://doi.org/10.1148/radiology.189.1.8372196>.
- [5] J. M. Goo, T. Tongdee, R. Tongdee, K. Yeo, C. F. Hildebolt, and K. T. Bae. "Volumetric Measurement of Synthetic Lung Nodules with Multi-Detector Row CT: Effect of Various Image Reconstruction Parameters and Segmentation Thresholds on Measurement Accuracy". In: *Radiology* 235.3 (June 2005). DOI: <https://doi.org/10.1148/radiol.2353040737>.
- [6] T. Kobayashi, X. W. Xu, H. MacMahon, C. E. Metz, and K. Doi. "Effect of a computer-aided diagnosis scheme on radiologists' performance in detection of lung nodules on radiographs." In: *Radiology* 199.3 (June 1996).
- [7] E. K. Fishman, D. Magid, D. R. Ney, E. L. Chaney, S. M. Pizer, J. G. Rosenman, D. N. Levin, M. W. Vannier, J. E. Kuhlman, and D. D. Robertson. "Three-dimensional imaging." In: *Radiology* 181.2 (Nov. 1991). DOI: <https://doi.org/10.1148/radiology.181.2.1789832>.

- [8] G. D. Rubin, C. F. Beaulieu, V. Argiro, H. Ringl, A. M. Norbash, J. F. Feller, M. D. Dake, R. B. Jeffrey, and S. Napel. "Perspective volume rendering of CT and MR images: applications for endoscopic imaging." In: *Radiology* 199.2 (May 1996). doi: <https://doi.org/10.1148/radiology.199.2.8668772>.
- [9] M. Levoy. "Display of Surfaces from Volume Data". In: *IEEE Comput. Graph. Appl.* 8.3 (May 1988), pp. 29–37. issn: 0272-1716. doi: 10.1109/38.511. url: <https://doi.org/10.1109/38.511>.
- [10] H. T. UY and L. T. UY. "IEEE Computer Graphics and Applications". In: 4 (1984).
- [11] N. Max. "Optical Models for Direct Volume Rendering". In: *IEEE Transactions on Visualization and Computer Graphics* 1.2 (June 1995), pp. 99–108. issn: 1077-2626. doi: 10.1109/2945.468400. url: <https://doi.org/10.1109/2945.468400>.
- [12] J. Fong, M. Wrenninge, C. Kulla, and R. Habel. "Production Volume Rendering: SIGGRAPH 2017 Course". In: *ACM SIGGRAPH 2017 Courses*. SIGGRAPH '17. Los Angeles, California: Association for Computing Machinery, 2017. isbn: 9781450350143. doi: 10.1145/3084873.3084907. url: <https://doi.org/10.1145/3084873.3084907>.
- [13] H. Pfister, W. Lorensen, C. Bajaj, G. Kindlmann, W. Schroeder, L. Avila, K. Raghu, R. Machiraju, and L. Jinho. "The Transfer Function Bake-Off." In: *Computer Graphics and Applications, IEEE* 21 (June 2001), pp. 16–22. doi: 10.1109/38.920623.
- [14] L. Soler, S. Nicolau, P. Pessaux, D. Mutter, and J. Marescaux. "Real-time 3D image reconstruction guidance in liver resection surgery". In: *Hepatobiliary Surgery and Nutrition* 3.2 (2014).
- [15] R. Bruggmann. *Unity® Volume Rendering – Plug-in zum Rendern von medizinischen Daten*. Oct. 2016. doi: 10.13140/RG.2.2.19248.05124.
- [16] G. Wheeler, S. Deng, N. Toussaint, K. Pushparajah, J. A. Schnabel, J. M. Simpson, and A. Gomez1. "Virtual interaction and visualisation of 3D medical imaging data with VTK and Unity". In: *Healthc Technol Lett.* 5.5 (Oct. 2018), pp. 148–153. doi: 10.1049/htl.2018.5064.

- [17] W. Lorensen and H. Cline. "Marching Cubes: A High Resolution 3D Surface Construction Algorithm". In: *ACM SIGGRAPH Computer Graphics* 21 (Aug. 1987), pp. 163–. doi: 10.1145/37401.37422.
- [18] D. Bartz and M. Meiner. "Voxels versus Polygons: A Comparative Approach for Volume Graphics". In: (Jan. 2000). doi: 10.1007/978-1-4471-0737-8_10.
- [19] S. Parker, P. Shirley, Y. Livnat, C. Hansen, and P. .-. Sloan. "Interactive ray tracing for isosurface rendering". In: (Oct. 1998), pp. 233–238. issn: 1070-2385. doi: 10.1109/VISUAL.1998.745713.
- [20] M. Meiundefinedner, J. Huang, D. Bartz, K. Mueller, and R. Crawfis. "A Practical Evaluation of Popular Volume Rendering Algorithms". In: *Proceedings of the 2000 IEEE Symposium on Volume Visualization. VVS '00*. Salt Lake City, Utah, USA: Association for Computing Machinery, 2000, pp. 81–90. isbn: 1581133081. doi: 10.1145/353888.353903. url: <https://doi.org/10.1145/353888.353903>.
- [21] P. Lacroute and M. Levoy. "Fast Volume Rendering Using a Shear-Warp Factorization of the Viewing Transformation". In: *Proceedings of the 21st Annual Conference on Computer Graphics and Interactive Techniques. SIGGRAPH '94*. New York, NY, USA: Association for Computing Machinery, 1994, pp. 451–458. isbn: 0897916670. doi: 10.1145/192161.192283. url: <https://doi.org/10.1145/192161.192283>.
- [22] H. Pfister, J. Hardenbergh, J. Knittel, H. Lauer, and L. Seiler. "The VolumePro Real-Time Ray-Casting System". In: *Proceedings of the 26th Annual Conference on Computer Graphics and Interactive Techniques. SIGGRAPH '99*. USA: ACM Press/Addison-Wesley Publishing Co., 1999, pp. 251–260. isbn: 0201485605. doi: 10.1145/311535.311563. url: <https://doi.org/10.1145/311535.311563>.
- [23] R. Yagel and Z. Shi. "Accelerating volume animation by space-leaping". In: *Proceedings Visualization '93*. Oct. 1993, pp. 62–69. doi: 10.1109/VISUAL.1993.398852.
- [24] J. Danskin and P. Hanrahan. "Fast Algorithms for Volume Ray Tracing". In: *Proceedings of the 1992 Workshop on Volume Visualization. VVS '92*. Boston, Massachusetts, USA: Association for Computing Machinery, 1992, pp. 91–98. isbn: 0897915275. doi: 10.1145/147130.147155. url: <https://doi.org/10.1145/147130.147155>.

- [25] N. Max. "Optical models for direct volume rendering". In: *IEEE Transactions on Visualization and Computer Graphics* 1.2 (June 1995), pp. 99–108. issn: 2160-9306. doi: 10.1109/2945.468400.
- [26] M. Meißner, H. Pfister, R. Westermann, and C. Wittenbrink. "Volume visualization and volume rendering techniques". In: (Jan. 2000).
- [27] P. Ljung, J. Krüger, E. Groller, M. Hadwiger, C. D. Hansen, and A. Ynnerman. "State of the Art in Transfer Functions for Direct Volume Rendering". In: *Computer Graphics Forum* 35.3 (2016), pp. 669–691. doi: 10.1111/cgf.12934. eprint: <https://onlinelibrary.wiley.com/doi/pdf/10.1111/cgf.12934>. URL: <https://onlinelibrary.wiley.com/doi/abs/10.1111/cgf.12934>.
- [28] H. Pfister, B. Lorensen, C. Bajaj, G. Kindlmann, W. Schroeder, L. S. Avila, K. M. Raghu, R. Machiraju, and Jinho Lee. "The transfer function bake-off". In: *IEEE Computer Graphics and Applications* 21.3 (May 2001), pp. 16–22. issn: 1558-1756. doi: 10.1109/38.920623.
- [29] C. Rezk Salama, M. Keller, and P. Kohlmann. "High-Level User Interfaces for Transfer Function Design with Semantics". In: *IEEE transactions on visualization and computer graphics* 12 (Oct. 2006), pp. 1021–8. doi: 10.1109/TVCG.2006.148.
- [30] G. Kindlmann and J. W. Durkin. "Semi-automatic generation of transfer functions for direct volume rendering". In: *IEEE Symposium on Volume Visualization (Cat. No.989EX300)*. Oct. 1998, pp. 79–86.
- [31] N. Young, N. W. C. DorschR, J. Kingston, G. Markson, and J. McMahon. "Intracranial aneurysms: evaluation in 200 patients with spiral CT angiography". In: *Eur Radiol* 11.123 (2001). issn: 0938-7994. doi: <https://doi.org/10.1007/s0033000000523>.
- [32] J. Kniss, G. Kindlmann, and C. Hansen. "Interactive volume rendering using multi-dimensional transfer functions and direct manipulation widgets". In: *Proceedings Visualization, 2001. VIS '01*. Oct. 2001, pp. 255–562. doi: 10.1109/VISUAL.2001.964519.
- [33] J. Kniss, G. Kindlmann, and C. Hansen. "Multidimensional transfer functions for interactive volume rendering". In: *IEEE Transactions on Visualization and Computer Graphics* 8.3 (July 2002), pp. 270–285. issn: 2160-9306. doi: 10.1109/TVCG.2002.1021579.

- [34] F. V. Higuera, N. Sauber, B. Tomandl, C. Nimsy, G. Greiner, and P. Hastreiter. "Automatic adjustment of bidimensional transfer functions for direct volume visualization of intracranial aneurysms". In: *Medical Imaging 2004: Visualization, Image-Guided Procedures, and Display*. Ed. by R. L. G. Jr. Vol. 5367. International Society for Optics and Photonics. SPIE, 2004, pp. 275–284. DOI: 10.1117/12.535534. URL: <https://doi.org/10.1117/12.535534>.
- [35] J. Kniss, J. P. Schulze, U. Wössner, P. Winkler, U. Lang, and C. Hansen. "Medical Applications of Multi-Field Volume Rendering and VR Techniques". In: *Eurographics / IEEE VGTC Symposium on Visualization*. Ed. by O. Deussen, C. Hansen, D. Keim, and D. Saupe. The Eurographics Association, 2004. ISBN: 3-905673-07-X. DOI: 10.2312/VisSym/VisSym04/249-254.
- [36] L. Mroz, H. Hauser, and E. Gröller. "Interactive High-Quality Maximum Intensity Projection". In: *Computer Graphics Forum* 19.3 (2000), pp. 341–350. DOI: 10.1111/1467-8659.00426. eprint: <https://onlinelibrary.wiley.com/doi/pdf/10.1111/1467-8659.00426>. URL: <https://onlinelibrary.wiley.com/doi/abs/10.1111/1467-8659.00426>.
- [37] K. Engel, M. Hadwiger, J. M. Kniss, A. E. Lefohn, C. R. Salama, and D. Weiskopf. "Real-Time Volume Graphics". In: *ACM SIGGRAPH 2004 Course Notes*. SIGGRAPH '04. Los Angeles, CA: Association for Computing Machinery, 2004, 29–es. ISBN: 9781450378017. DOI: 10.1145/1103900.1103929. URL: <https://doi.org/10.1145/1103900.1103929>.
- [38] K. Xie, J. Yang, and Y. Zhu. "Real-time visualization of large volume datasets on standard PC hardware". In: *Computer Methods and Programs in Biomedicine* 90.2 (2008), pp. 117–123. ISSN: 0169-2607. DOI: <https://doi.org/10.1016/j.cmpb.2007.12.006>. URL: <http://www.sciencedirect.com/science/article/pii/S0169260707003112>.
- [39] J. Kruger and R. Westermann. "Acceleration Techniques for GPU-Based Volume Rendering". In: *Proceedings of the 14th IEEE Visualization 2003 (VIS'03)*. VIS '03. USA: IEEE Computer Society, 2003, p. 38. ISBN: 0769520308.
- [40] S. Laine and T. Karras. "Efficient Sparse Voxel Octrees". In: *IEEE transactions on visualization and computer graphics* 17 (Oct. 2010), pp. 1048–59. DOI: 10.1109/TVCG.2010.240.

- [41] D. Ruijters and A. Vilanova. "Optimizing GPU volume rendering". In: *Journal of WSCG* 14.1-3 (2006), pp. 9–16.
- [42] J. F. Blinn. "Models of Light Reflection for Computer Synthesized Pictures". In: *Proceedings of the 4th Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH '77. San Jose, California: Association for Computing Machinery, 1977, pp. 192–198. ISBN: 9781450373555. DOI: 10.1145/563858.563893. URL: <https://doi.org/10.1145/563858.563893>.
- [43] K. Suffern. *Ray Tracing from the Ground Up*. USA: A. K. Peters, Ltd., 2007. ISBN: 1568812728.
- [44] I. Epic Games. *Graphics Programming Overview*. 2004-2020. URL: <https://docs.unrealengine.com/en-US/Programming/Rendering/Overview/index.html>.
- [45] R. Bibb, D. Eggbeer, and A. Paterson. "2 - Medical imaging". In: *Medical Modelling (Second Edition)*. Ed. by R. Bibb, D. Eggbeer, and A. Paterson. Second Edition. Oxford: Woodhead Publishing, 2015, pp. 7–34. ISBN: 978-1-78242-300-3. DOI: <https://doi.org/10.1016/B978-1-78242-300-3.00002-0>. URL: <http://www.sciencedirect.com/science/article/pii/B9781782423003000020>.
- [46] N. E. M. A. (NEMA) and D. S. Committee. *Digital Imaging and Communications in Medicine*. URL: <https://www.dicomstandard.org/>.
- [47] A. Kalra. "Developing FE Human Models From Medical Images". In: Jan. 2018, pp. 389–415. ISBN: 9780128098318. DOI: 10.1016/B978-0-12-809831-8.00009-X.
- [48] M. Lev and R. Gonzalez. "17 - CT Angiography and CT Perfusion Imaging". In: *Brain Mapping: The Methods (Second Edition)*. Ed. by A. W. Toga and J. C. Mazziotta. Second Edition. San Diego: Academic Press, 2002, pp. 427–484. ISBN: 978-0-12-693019-1. DOI: <https://doi.org/10.1016/B978-012693019-1/50019-8>. URL: <http://www.sciencedirect.com/science/article/pii/B9780126930191500198>.
- [49] L. MH, F. J, G. JJ, H. ST, H. GJ, K. WJ, and G. RG. "Acute stroke: improved nonenhanced CT detection–benefits of soft-copy interpretation by using variable window width and

- center level settings.” In: *Radiology* 213.1 (Oct. 1999). DOI: 10.1148/radiology.213.1.r99oc10150.
- [50] J. Mandell, B. Khurana, L. Folio, H. Hyun, S. Smith, R. Dunne, and K. Andriole. “Clinical Applications of a CT Window Blending Algorithm: RADIO (Relative Attenuation-Dependent Image Overlay)”. In: *Journal of Digital Imaging* 30 (Jan. 2017). DOI: 10.1007/s10278-017-9941-1.
- [51] F. Hernell, P. Ljung, and A. Ynnerman. “Local Ambient Occlusion in Direct Volume Rendering”. In: *IEEE Transactions on Visualization and Computer Graphics* 16.4 (July 2010), pp. 548–559. ISSN: 2160-9306. DOI: 10.1109/TVCG.2009.45.
- [52] T. Ritschel, C. Dachsbacher, T. Grosch, and J. Kautz. “The State of the Art in Interactive Global Illumination”. In: *Comput. Graph. Forum* 31.1 (Feb. 2012), pp. 160–188. ISSN: 0167-7055. DOI: 10.1111/j.1467-8659.2012.02093.x. URL: <https://doi.org/10.1111/j.1467-8659.2012.02093.x>.
- [53] T. Lokovic and E. Veach. “Deep Shadow Maps”. In: *Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH ’00. USA: ACM Press/Addison-Wesley Publishing Co., 2000, pp. 385–392. ISBN: 1581132085. DOI: 10.1145/344779.344958. URL: <https://doi.org/10.1145/344779.344958>.
- [54] T. Mertens, J. Kautz, P. Bekaert, and F. Van Reeth. “A Self-Shadow Algorithm for Dynamic Hair using Density Clustering.” In: Jan. 2004, pp. 173–178. DOI: 10.1145/1186223.1186278.
- [55] K. Engel, M. Kraus, and T. Ertl. “High-Quality Pre-Integrated Volume Rendering Using Hardware-Accelerated Pixel Shading”. In: *Proceedings of the ACM SIGGRAPH Conference on Computer Graphics* (June 2001). DOI: 10.1145/383507.383515.
- [56] H. Nyquist. “Certain Topics in Telegraph Transmission Theory”. In: *Transactions of the American Institute of Electrical Engineers* 47.2 (Apr. 1928), pp. 617–644. ISSN: 2330-9431. DOI: 10.1109/T-AIEE.1928.5055024.
- [57] J. Revelles, C. Ureña, M. Lastra, D. Lenguajes, S. Informaticos, and E. Informatica. “An Efficient Parametric Algorithm for Octree Traversal”. In: (May 2000).

- [58] J. Wilhelms and A. Van Gelder. “Octrees for Faster Isosurface Generation”. In: *ACM Trans. Graph.* 11.3 (July 1992), pp. 201–227. ISSN: 0730-0301. DOI: 10.1145/130881.130882. URL: <https://doi.org/10.1145/130881.130882>.
- [59] K. P. Soundararajan and T. Schultz. “Learning Probabilistic Transfer Functions: A Comparative Study of Classifiers”. In: *Computer Graphics Forum* 34.3 (2015), pp. 111–120. DOI: 10.1111/cgf.12623. eprint: <https://onlinelibrary.wiley.com/doi/pdf/10.1111/cgf.12623>. URL: <https://onlinelibrary.wiley.com/doi/abs/10.1111/cgf.12623>.
- [60] O. Sharma, T. Arora, and A. Khattar. “Graph-Based Transfer Function for Volume Rendering”. In: *Computer Graphics Forum* (May 2019). DOI: 10.1111/cgf.13663.
- [61] A. Pérard-Gayot, J. Kalojanov, and P. Slusallek. “GPU Ray Tracing Using Irregular Grids”. In: *Comput. Graph. Forum* 36.2 (May 2017), pp. 477–486. ISSN: 0167-7055. DOI: 10.1111/cgf.13142. URL: <https://doi.org/10.1111/cgf.13142>.