PEOPLE'S DECMOCRATIC REPUBLIC OF ALGERIA
Ministery of higher Education and Scientific Research
University Mohamed Khider – BISKRA
Faculty of Exact Sciences, Natural and Life Sciences

Computer science departement

Order N° : IVA09/M2/2023

Dissertation

Presented to obtain the academic master's degree in

Computer Science

Option : **Image and Artificial Life(IAL)**

Distance maps bases ray casting of volumetric images

By :

ZINEDDINE MAGHARBI

Defended in 19/06/2023 before the members of the jury composed of :

CHIGHOUB Rabiaa	**MCB**	President
BAHI Naima	**MCB**	Supervisor
BENTRAH Ahlem	**MCB**	Examiner

Academic year 2022-2023

Contents

List of Figures

List of Tables

Résumé

Le rendu volumique est une technique essentielle en infographie qui permet la visualisation de jeux de données tridimensionnels (3D). Cet article présente un aperçu du rendu volumique et du raycasting, mettant en évidence l'intégration des cartes de distance de Chebyshev dans le processus.

Le rendu volumique consiste à générer des images en 2D à partir de données volumétriques, telles que des scans médicaux ou des simulations scientifiques. Le raycasting est un algorithme couramment utilisé dans le rendu volumique, qui trace des rayons à travers le volume pour déterminer la contribution de chaque voxel à l'image finale. En combinant cette technique avec les cartes de distance de Chebyshev, nous pouvons obtenir une précision accrue dans la visualisation volumique.

Les cartes de distance de Chebyshev fournissent une mesure de la distance basée sur la différence maximale dans n'importe quelle dimension entre deux points de l'espace. Dans le rendu volumique, les cartes de distance de Chebyshev sont utilisées pour calculer les fonctions de transfert d'opacité et de couleur, permettant une représentation plus précise des structures complexes et des frontières à l'intérieur du volume. Cette approche permet une meilleure visualisation de caractéristiques telles que les frontières d'organes, les structures vasculaires et les détails anatomiques.

Abstract

Volume rendering is a vital technique in computer graphics that allows the visualization of three-dimensional (3D) data sets. This article presents an overview of volume rendering and raycasting, highlighting the integration of Chebyshev distance maps in the process.

Volume rendering involves generating 2D images from volumetric data, such as medical scans or scientific simulations. Raycasting is a commonly used algorithm in volume rendering that traces rays through the volume to determine the contribution of each voxel to the final image. By combining this technique with Chebyshev distance maps, we can achieve enhanced accuracy in volume visualization.

Chebyshev distance maps provide a measure of distance based on the maximum difference in any dimension between two points in space. In volume rendering, Chebyshev distance maps are used to calculate the opacity and color transfer functions, enabling a more accurate representation of complex structures and boundaries within the volume. This approach allows for better visualization of features like organ boundaries, vascular structures, and anatomical details.

General Introduction

Volume rendering plays [19] a critical role in various scientific domains, enabling the visualization and analysis of three-dimensional data. It has proven indispensable in fields such as medical imaging, computational fluid dynamics, and materials science, where understanding volumetric structures is essential for accurate diagnosis, simulation, and research. Two primary methods used in volume rendering are direct and indirect rendering [[5], [58]]. Indirect volume rendering is a technique for generating geometric representations from volumetric data. Like in the Marching Cubes algorithm, we extract polygonal surfaces from the volume, enabling the visualization of complex structures. Direct volume rendering techniques, such as raycasting, involve casting rays through the volume and computing the color and opacity of each ray based on the properties of the underlying data.

However, the efficient rendering of volumetric datasets poses several challenges that need to be addressed. One significant challenge is sampling, as the volume may contain a vast number of data points, making it impractical to render every point. Various sampling strategies[[37], [13]], including regular sampling, Regular sampling can lead to aliasing artifacts and may not capture fine details accurately. Irregular sampling techniques, such as adaptive and hierarchical sampling, aim to improve efficiency by selectively sampling regions of interest, but they can introduce complexity in determining the optimal sampling rate and may still miss important features. These strategies require careful parameter tuning and can be computationally expensive.

So, the second critical aspect is acceleration, as rendering large volumes in real time requires efficient algorithms and data structures. Acceleration techniques [[69],[45]], such as spatial partitioning structures like KD-trees and octrees, enable faster ray traversal by organizing the volume data in a hierarchical manner. These structures facilitate efficient intersection tests and enable faster empty space skipping, reducing the computational overhead.

The main idea behind distance maps is to represent the distance from each point in a volumetric dataset to the nearest surface. Distance maps provide valuable information about the spatial relationship between the volume and the surfaces within it. By calculating and utilizing distance maps, it becomes possible to efficiently skip empty spaces during rendering, thereby reducing the computational cost. This approach enables faster and more efficient volume rendering by focusing only on relevant regions and bypassing empty or non-contributing regions. Distance maps are commonly used in techniques such as empty space skipping and can significantly enhance the rendering process.

In the context of this project, the focus is on accelerating direct volume rendering using a specific acceleration technique called Chebyshev Distance empty space skipping. This technique leverages the Chebyshev Distance, which represents the distance between a ray and the nearest occupied block in the volume. By efficiently estimating this distance and utilizing it to skip multiple empty blocks along a ray, unnecessary computations can be avoided, leading to significant performance improvements.

The main objective of this project is to develop and evaluate an efficient approach for accelerating volumetric raycasting using Chebyshev Distance empty space skipping. By incorporating this technique into the rendering pipeline, the goal is to enhance the rendering performance for large-scale volumetric datasets while maintaining high-quality visual results.

The manuscript is structured to provide a comprehensive understanding of the research conducted. It consists of different chapters covering concepts and foundations, including an introduction to volume rendering, volumetric data representation, various rendering techniques, and volume rendering tools. The subsequent chapters delve into the details of volumetric raycasting, including sampling strategies, empty space skipping techniques, such as KD-trees and octrees, also the used Chebyshev distance empty space skipping approach. The implementation details, experimental setup, dataset selection, and results are described in the chapter on implementation and results. Finally, the general conclusion summarizes the findings discusses their implications, and highlights potential future directions for enhancing the efficiency of volumetric raycasting using Chebyshev Distance empty space skipping.

Chapter 1

Volume Rendering

1.1 Introduction

This chapter aims to provide readers with a comprehensive introduction to volume rendering[19], a powerful computer graphics technique. The primary objective is to familiarize readers with the multitude of methods employed in volume rendering and offer an in-depth description of one of the most widely acclaimed techniques: ray casting. By exploring the concepts and foundations of volume rendering, the different techniques utilized in volume rendering are extensively examined and classified into two categories: direct volume rendering and indirect volume rendering[67].

Within the realm of direct volume rendering, the chapter focuses on two prominent techniques: Maximum Intensity Projection (MIP) and Shear-Warp[68],[32].

The chapter also introduces the widely used Marching Cubes algorithm[50],[48] as a representative technique for indirect volume rendering.

To aid in the understanding and visualization of volumetric data, the chapter explores the concept of transfer functions[47].

Furthermore, the chapter introduces the Volume Rendering Integral[28], a fundamental concept that underlies many volume rendering algorithms.

Ray casting[59] is presented as a key technique in volume rendering, wherein rays are cast through the volume to calculate color and opacity at each sample point. This approach allows for the creation of high-quality renderings with diverse lighting effects and material properties.

Finally, the chapter presents popular volume rendering tools, including VTK and MITK[4],[3].

1.2 Volume Rendering

Each pixel on our display displays represents a unit area in a two-dimensional array. Each cubic element that makes up a volume is a three-dimensional array that represents a single unit of space. Volume elements, often known as voxels, are individual components of a three-dimensional space. The value at that location is a number assigned to each point in a volume. A scalar field on the volume is the term used to describe the grouping of all these values.[22] A level surface is the collection of all points in the volume that have a specific scalar value. Scalar fields are displayed by a method called volume rendering.

It is a technique for displaying a set of three-dimensional data. Using volume rendering techniques, a data set's internal details are projected onto a display screen. Interior data values along the ray path from each screen pixel are analyzed and encoded for display. Depending on the application, the data are encoded differently for presentation. For instance, seismic data is frequently evaluated to determine the highest and lowest values along each ray. The values can then be color-coded to indicate the interval's breadth and the lowest value. For tissue and bone layers in medical applications, the data values are opacity factors in the range of 0 to 1. Layers of bone are totally opaque, while the tissue is somewhat transparent.[23, 59]

Voxels serve as representations for several physical properties such as density, temperature, velocity, and pressure. The volume records can be used to extract other measures like area and volume.[14, 35] Medical imaging (such as computed tomography, magnetic resonance imaging, and ultrasonography), biology (such as confocal microscopy), geophysics (such as seismic measurements from oil and gas exploration), industry (such as finite element models), molecular systems (such as electron density maps), meteorology (such as stormy (prediction), computational fluid dynamics (such as water flow), computational chemistry (such as new materials), and digital signs are examples of applications of volume visualization (e.g., CSG)[14]

Large 3D datasets are produced via numerical simulations and sampling techniques such as MRI, CT, PET, ultrasonic imaging, confocal microscopy, supercomputer simulations, geometric models, laser scanners, depth images calculated by stereo disparity, satellite imaging, and sonar.

By applying sampling techniques, 3D scientific data can be produced in a number of areas. Volumetric data from biomedical scanners are normally presented as 2D slices of a standard, Cartesian grid, with minor variations occasionally. Uneven grids are a common result of supercomputer and Finite Element Method (FEM) simulations. A series of randomly aligned, fan-shaped slices that make up partially organized point samples make up an ultrasonic scan's raw output. These scanners produce a series of 2D slices, which are then combined to create a 3D volume model. Imaging devices can record a great deal of information necessary for scientific research thanks to their millimeter-scale resolution.[31]

To understand the data being represented, it is frequently required to observe the dataset from continuously shifting angles. The most important need is real-time interaction, which is preferable even if it is depicted in a somewhat less realistic manner. The following justifications justify the necessity of a real-time rendering system: Volume rendering is a group of techniques used in computer graphics and scientific visualization to project a 3D data set into a 2D space using discrete samples. A collection of 2D slice images from an MRI, CT, or Micro CT scanner is an illustration of a 3D data set. For instance, a volume rendering technique can be used to create 3D volume-rendered images from a collection of 2D slice photographs of a human brain

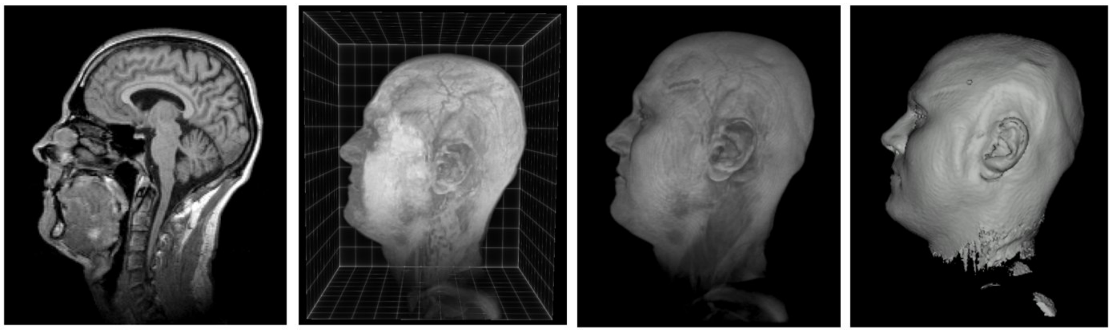


Figure 1.1: Volume rendering examples[53]

Also, it makes it possible for users to see three-dimensional scalar fields. Each industry that generates 3D data sets for analysis, such as physics, medicine, disaster preparedness, and more, should be aware of this.[19, 31].

Normally, users take these slices at regular intervals and with a regular quantity of image pixels, such as one slice per millimeter. In a regular volumetric grid, each voxel or volume element is sampled to gather volumetric data, which is then used to generate a single representative value for that region.[19]

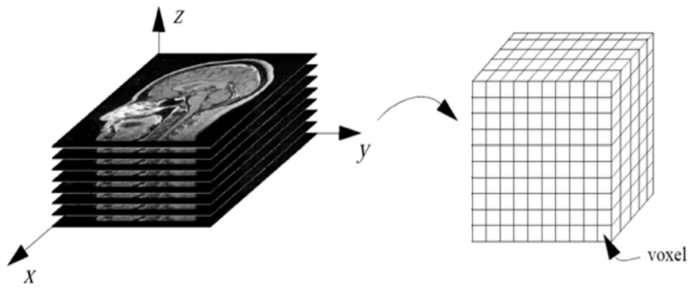


Figure 1.2: Stacking 2D slices to create a 3D volume.[20]

1.3 Volumetric data

Data that depicts a three-dimensional region or volume in detail is referred to as volumetric data. This kind of information is frequently employed in scientific and engineering disciplines including computer graphics, meteorology, geology, and medical imaging.

Volumetric information may be gathered from a variety of imaging techniques, including computed tomography (CT) scans, magnetic resonance imaging (MRI), and ultrasound. These scans provide a number of two-dimensional pictures or slices that may be assembled to depict the scanned item or body part in three dimensions.

Simulations in other disciplines, like fluid dynamics or weather modeling, can provide volumetric data. The features of a volume, like its temperature, pressure, or velocity, are described by the vast volumes of data produced by these simulations.

For presenting volumetric data, a three-dimensional grid or matrix is commonly used, with each point on the grid denoting a particular position in the volume and including details about that site's characteristics. It is possible to display this data in order to provide accurate and instructive representations of the volume using a variety of approaches, including volume rendering, surface extraction, and isosurface rendering.[11]

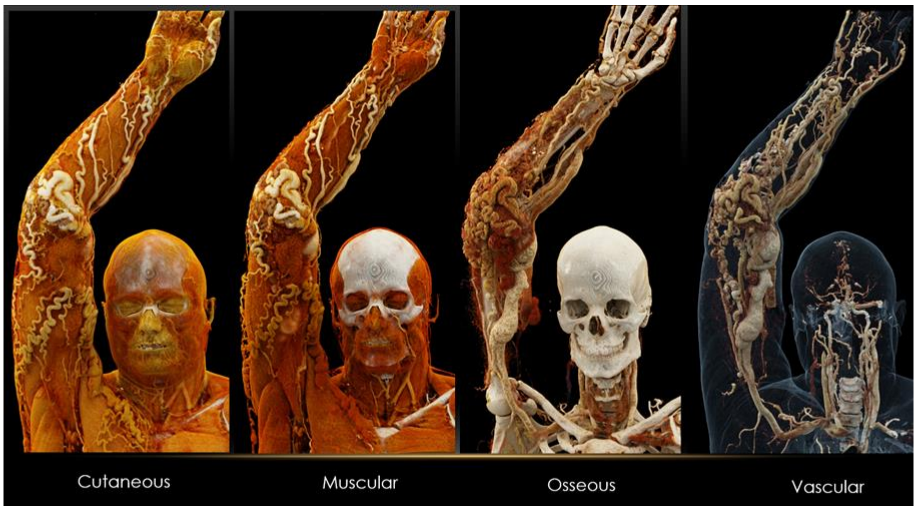


Figure 1.3: Cinematically rendered images based on patient CT scans viewed using different presets for skin, muscle, bone, and vasculature (from left to right).[55]

1.4 Techniques

For the past 20 years, the most active sector of scientific visualization research has been volume visualization. Techniques for volume visualization must provide efficient data processing, comprehensible data representations, and fairly speedy rendering in order to be effective. Scientific users must be able to alter the settings and see the outcome right away. Few modern systems can perform at this level, thus researchers are still exploring a variety of novel approaches to efficiently employ volume visualization.

The various rendering techniques differ in a number of areas that may be used to classify them in depth in different ways[15, 25, 32]. Based on the area of the volume raster set that they produce, two distinct ways are used to visualize volumes coarsely.

There are various volume rendering methods, and choosing the best one depends on a number of variables. The four types of 3D volume rendering techniques include ray marching or casting, resampling or shear-warping, texture slicing, and splatting. Here, there are two main approaches:

- Surface rendering (or indirect volume rendering).
- Volume rendering (or direct volume rendering).

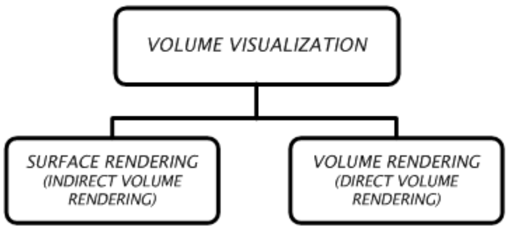

VOLUME VISUALIZATION
SURFACE RENDERING (INDIRECT VOLUME RENDERING)
VOLUME RENDERING (DIRECT VOLUME RENDERING)

Figure 1.4: Classification of volume visualization algorithms.[1]

1.4.1 Direct volume rendering

A method for visualizing a three-dimensional volume of data is volume rendering, commonly referred to as direct volume rendering. It is often used in scientific visualization, medical imaging, and other disciplines that need the analysis and viewing of three-dimensional data.

The main goal of volume rendering is to mimic how light interacts with a volume of data in three dimensions. This entails determining how these qualities vary along rays thrown across the volume after giving optical properties to the data, such as opacity and color. A combination of these rays creates the final picture, which shows the volume's interior structure.

1. The volume rendering method begins with gathering the three-dimensional data volume that has to be shown. Scientific simulations, other data sources, or imaging techniques used in medicine, such as CT or MRI, can all be used to obtain this information.

2. Pre-processing may be required to make the data appropriate for rendering once it has been acquired. In order to reduce background noise or isolate particular characteristics within the volume, this may include using various approaches including filtering or segmentation.

3. The transfer function converts the optical characteristics of the data to color and opacity values, making it a crucial stage in the volume rendering process. This feature may be customized by the user to draw attention to particular aspects of interest in the volume.[16]

4. The optical properties at each point along the beam are then calculated after casting rays through the data volume. Finding the ray and volume intersection point is the first step in doing this. or future society at large, or for evidence-based education methodologies.

5. Compositing: The final image is created by combining the optical parameters that have been determined along each ray. This entails employing a process known as alpha blending to composite the color and opacity values along each ray.

The visual representation of the three-dimensional volume of data that emerges from the volume rendering process may be interactively examined and evaluated. Users may learn more about the internal structure of the data by using various transfer functions to highlight various structures or characteristics within the volume.[42]

A 3D representation of volume data may be immediately obtained using the Direct Volume Rendering (DVR) concept. The data is thought to be represented via a semi-transparent light-emitting material. Therefore, gaseous phenomena can also be simulated. DVR strategies are founded on principles of physics (emission, absorption, dispersal).

The volume data is used in its entirety, allowing us to view the volume's inner structures. Each sample value must have an opacity and a color mapping in order for a direct volume renderer to work[44]. This is accomplished via a "transfer function," which may be a straightforward ramp, a broken-down linear function, or a random table. when changed to an RGBA(red, green, blue, alpha) value, projection of the composite RGBA results onto the relevant frame buffer pixel.

The rendering method will determine how this is accomplished. These methods can be mixed and matched. The aligned slices in the off-screen buffer, for instance, may be drawn using texturing hardware in a shear warp implementation. With DVR, both forward and backward techniques are available. Algorithms for image space and picture order are used in backward approaches. They are executed pixel by pixel. An example that will be covered in more depth below is ray casting. application of object space/object order techniques. The cells are projected onto the picture as these algorithms are run voxel by voxel. Examples of this method include cutting, shearing, and splatting.[6]

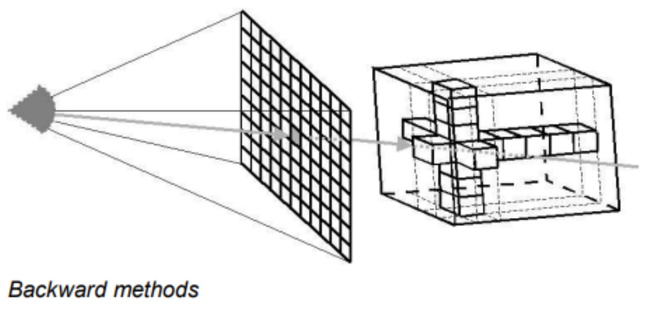


Figure 1.5: Backward methods.[6]

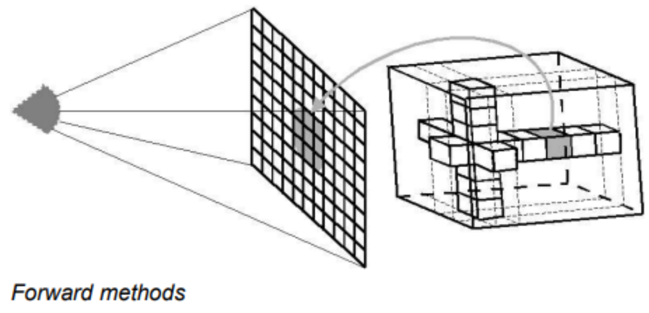


Figure 1.6: Forward methods.[6]

There are several types of direct volume rendering techniques:[10]

1.4.1.1 Maximum Intensity Projection (MIP)

A common method for converting volumetric data into 2D pictures or movies is called maximum intensity projection (MIP). This method allows viewers to visualize the most noticeable aspects of the information, such bones or blood veins, which makes it particularly helpful in scientific visualization and medical imaging.

The MIP method selects the greatest intensity value encountered along each ray after projecting rays across the 3D data. The pixel in the 2D picture that represents the ray's terminus is then colored using this highest intensity value. In other words, MIP decides which voxel along each ray is the brightest and shows that one as the final image.

Let's look at a straightforward example of a 3D dataset with a sphere to better grasp the MIP procedure. The resultant image would display the sphere's borders when viewed in 2D using MIP since the sphere's surface would be represented by the brightest voxels on each ray.

MIP is superior to other volume rendering methods in a number of ways. It can render enormous datasets in real-time or very close to real-time due to its first advantage: processing efficiency. Second, it creates pictures that resemble those from traditional X-rays, making it simple to use and intuitive. Third, it is helpful for determining the dataset's standout characteristics, which makes it especially helpful for scientific visualization and medical imaging[30, 68].

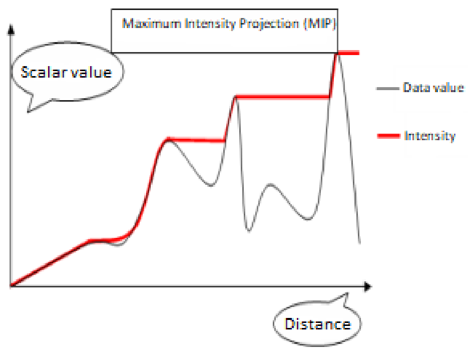


Figure 1.7: Typical profiles for MIP.[68]

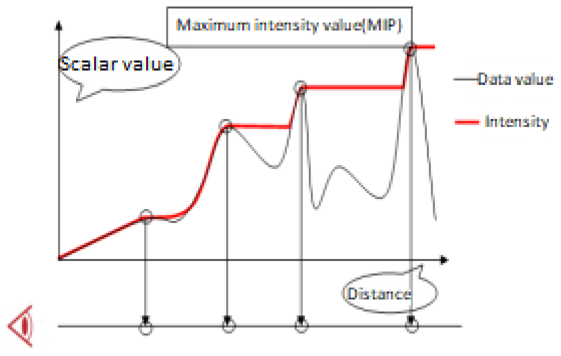


Figure 1.8: To get the crest along a viewing ray.[68]

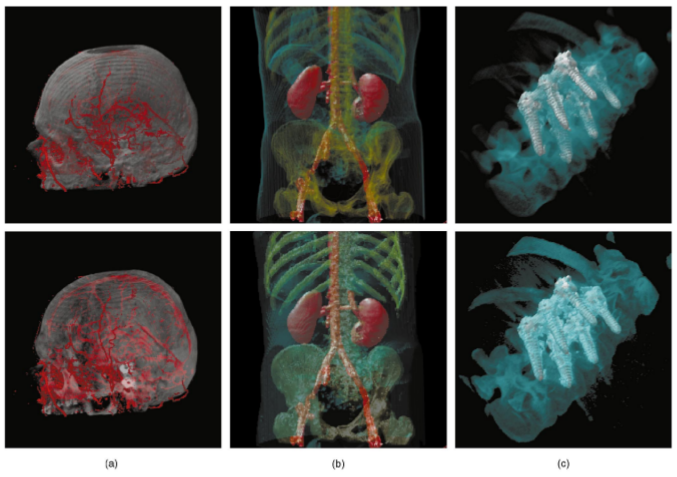


Figure 1.9: Comparisons between standard DVR (first row) and two-level volume rendering (second row). (a) Two-level volume rendering allows to show blood vessels within the head, even near the skull. (b) Using MIP for the visualization of context (bones, skin), the object of interest (blood vessels plus stenosis) becomes well visible from all viewing directions. (c) Again, MIP proves to work fine for context visualization. .[30]

With this approach, the maximum intensity value along each ray is shown, producing a picture that highlights the brightest characteristics in the data set.

1.4.1.2 Shear-Warp

By projecting 3D volumetric data onto a 2D viewing plane, the direct volume rendering technique known as shear warp makes it possible to visualize 3D volumetric data. This method has been extensively utilized in scientific visualization and medical imaging, where it is especially beneficial for real-time rendering of big datasets.

Shearing and warping are the first two phases of the shear warp algorithm. The 3D dataset is changed in the shearing stage by using a shear transformation along one of the three axes. The dataset is deformed by the shearing procedure, but it also lines up the dataset's axis with the viewing plane.

After the dataset has been sheared, a perspective projection is used to warp it onto the 2D viewing plane. In a manner similar to how a camera takes a picture, the projection translates the 3D data onto the 2D plane. To create a high-quality, realistic image, the projected image is next produced utilizing customary graphics methods like shading and texture mapping.

Shear warp has a lot of benefits, including speed. The method may be used on current graphics hardware, such as graphics processing units (GPUs), for real-time rendering because it is highly parallelizable. Shear warp is advantageous for applications that call for the presentation of intricate, volumetric data since it can handle huge datasets as well.

There have been various suggested shear warp variants to alleviate some of these constraints. In order to dynamically change the amount of shear applied to the dataset based on the local curvature of the dataset, adaptive shear warp algorithms, for instance, have been developed. For particular kinds of datasets and applications, several variants, such as object-order shear warp and view-order shear warp, have been created to increase the technique's accuracy and effectiveness.

As a direct volume rendering method, shear warp enables the presentation of 3D volumetric data by projecting it onto a 2D viewing plane. The method is effective for real-time rendering of complicated, volumetric data since it is highly parallelizable and capable of handling enormous datasets. Shear warp, nevertheless, has significant drawbacks and could generate aberrations that could skew the final image's accuracy. Shear warp has undergone a number of modifications to overcome these issues and enhance the technique's accuracy and effectiveness for particular datasets and applications[8, 40].

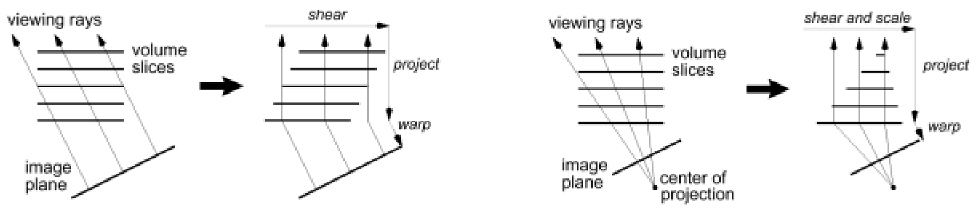


Figure 1.10: The shear-warp for parallel (left) and perspective (right) projections. Figure credit: P. Lacroute.[8]

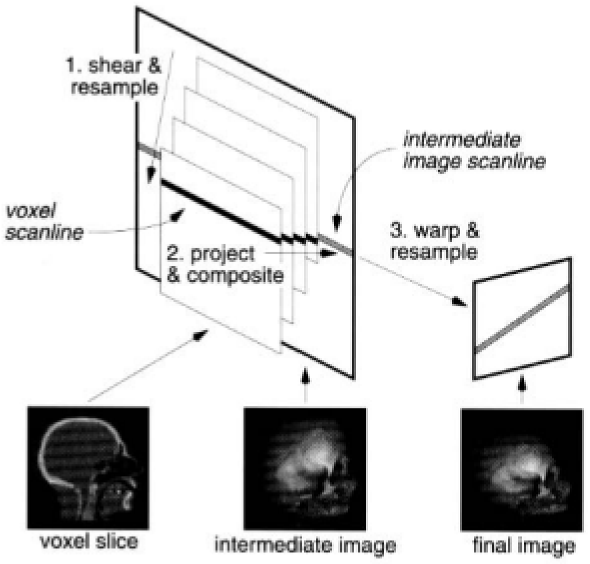


Figure 1.11: The shear-warp algorithm includes three conceptual steps: shear and resample the volume slices, project resampled voxel scanlines onto intermediate image scanlines, and warp the intermediate image into the final image.[40]

1.4.2 Indirect volume rendering

By modeling how light interacts with the material, the computer graphics approach known as indirect volume rendering visualizes data in a volume. This method depicts the lighting and scattering effects of the volume rather than just the volume's surface. Both scientific visualization and medical imaging make extensive use of them.

In order to draw indirect volumes, the volume data must first be imported and converted into a format that can be rendered. The material's illumination and scattering characteristics are then modeled using methods like ray tracing or Monte Carlo simulation. Lastly, to display the simulated data, a picture is created using methods like ray casting.

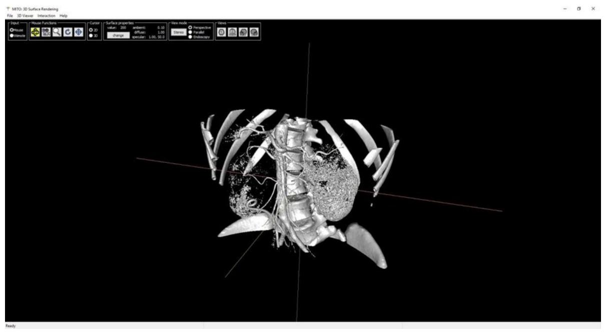


Figure 1.12: 3D indirect volume rendering.[12]

Indirect volume rendering has a number of benefits over conventional visualization methods, including the ability to show intricate interior structures and give a more realistic depiction of the volume's physical characteristics. Nevertheless, it could require a lot of computer work, and the produced pictures might be challenging to understand without the right visualization methods.[50]

A 2D projection of a 3D discretely sampled data set is displayed using the indirect volume rendering method in scientific visualization and computer graphics. It is based on a polygonal surface description of the relevant bodily cavities that creates triangle facets using the equipotential surface of volume data [67]. Compared to direct volume rendering, indirect volume rendering provides a number of benefits, such as quicker rendering times and more control over the final image[58].

The method they create the 2D representation of a 3D data set is the primary distinction between direct and indirect volume rendering. Using methods like volume raycasting or ray marching, direct volume rendering is a computationally demanding job that maps voxels directly onto pixels of a 2D picture plane[67]. When using the equipotential surface of volume data, indirect volume rendering creates triangular facets based on a polygonal surface representation of the corresponding body cavities. Direct volume rendering is inferior to indirect volume rendering in a number of ways, including rendering times and final picture control[58]

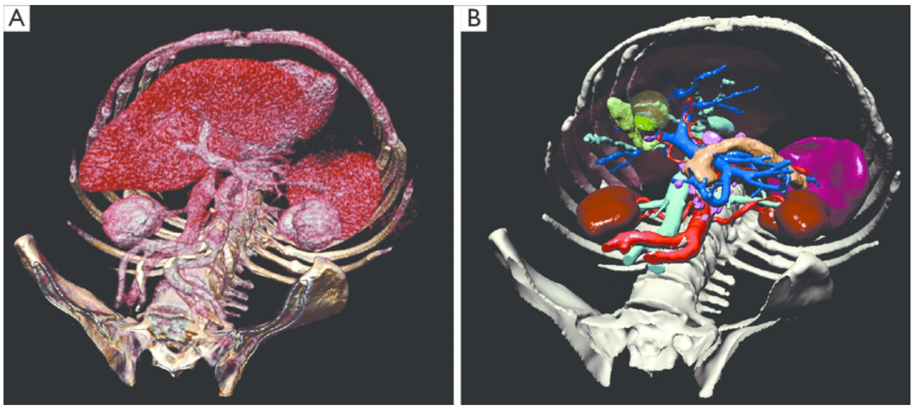


Figure 1.13: Comparison between direct volume rendering (A) and surface rendering after patient modeling (B). [64]

It is also helpful for applications like architectural visualization, medical imaging, and special effects in movies and video games since it can create very realistic pictures with intricate lighting and shading effects. To get the required results, IVR, however, may be computationally costly and necessitates careful parameter optimization.

There are several types of indirect volume rendering techniques, including[49]:

1.4.2.1 Marching cubes

The objective of the surface rendering (SR), indirect volume rendering, and iso-surfacing approaches is to produce a surface with constant density from a 3D dataset. The calculation complexity will be reduced as a result. Various techniques, including contour tracing, marching cubes [48], and marching tetrahedra, were developed for this technique.

The marching cubes algorithm's stages will be more thoroughly detailed in the next section [63].

Marching Cubes (MC): According to the original dataset's Iso-surface, Marching Cubes (MC) comes the closest . It uses triangular meshes to approximate surfaces. By using linear interpolation along cell edges, the surfaces are discovered. Following that, the standard vectors are

employed as the Iso-surface's gradients. In a nutshell, the marching cubes approach produces a surface using 3D datasets. The stages below outline this procedure in detail[51]:

1. Read into memory the first four slices.
2. Use four neighbors from the first slice and four neighbors from the second slice to build a cube.
3. Use the cube's eight density values to compare and create an index for the cube. the surface constant at its vertices.
4. Search a predetermined table's index to find the list of edges.
5. By linear interpolation, locate the intersection of the surface and the edges using the densities at each edge vertex.

6. Employing central differences, calculate a unit normal for each cube vertex. Put the normal into interpolation at each triangle vertex.

7. Vertex normals and triangular vertices should be output.

The biggest disadvantage of this method is as follows: The approach uses a large number of geometric primitives, therefore producing a high-quality picture of the 3D data demands a significant amount of processing. On the other hand, a dataset collection containing less data

See Figure 1.14., details will result in rendering that is less accurate and of lower quality.

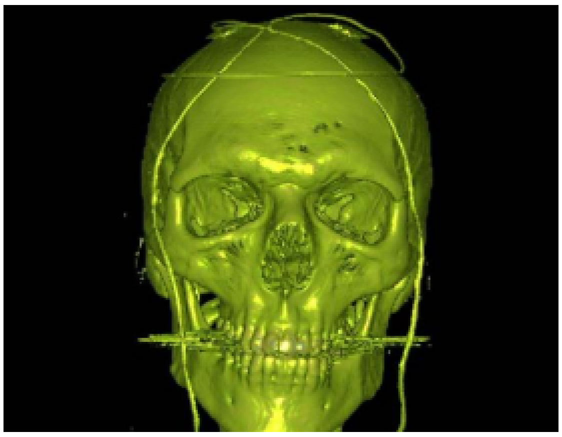


Figure 1.14: An Iso-surface surface rendering of a human skull [57]

1.5 Transfer function

Transforming one set of data into another via a mathematical function is the idea behind a transfer function. The mapping of scalar values of volumetric data to colors and opacities for rendering in computer graphics and visualization is done using transfer functions.

A transfer function in volume rendering is a tool that enables users to define the rendering of the volumetric data. It specifically converts scalar numbers to a palette of colors and opacities that may be used to visualize the data. If the volumetric data, for instance, indicates density, a transfer function may be used to translate low-density values to transparent or bright colors and high-density values to opaque or dark hues.

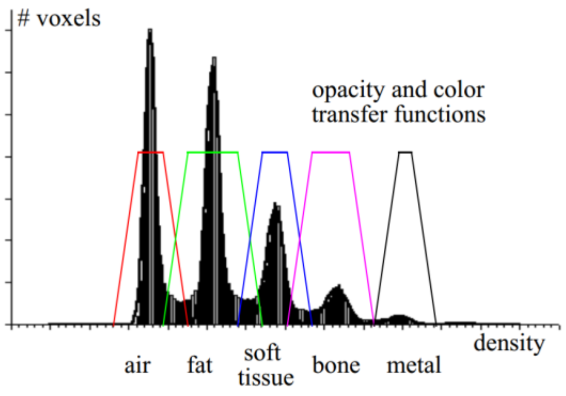


Figure 1.15: Trnasfer function graph

Different methods, such as color maps or transfer function curves, can be used to build and modify transfer functions. In contrast, to transfer function curves, which let users construct a personalized mapping between scalar values and colors or opacity, color maps are predetermined sets of colors that are given to scalar values.

The data visualization can be significantly impacted by the transfer function's design. Important characteristics of the volumetric data can be highlighted by a well-designed transfer function, whereas critical information might be obscured or hidden by a badly constructed transfer function. Transfer functions are thus a crucial component of volume rendering for the visualization and analysis of challenging 3D information.[47]

Transfer functions play a crucial role in volume rendering because they enable the logical and informative presentation of challenging 3D information. Transfer functions are crucial for the following reasons[47]:

- Enhances Visualization: Transfer functions convert scalar values into colors and opacities, making it possible to visualize volumetric data more effectively. Transfer functions can highlight hidden aspects and patterns in the data that might not be obvious from just looking at the raw scalar values by translating distinct scalar ranges to different colors and opacities.

- Transfer functions are quite adaptable, enabling customers to adjust the depiction to meet their own requirements. Users may easily evaluate the data and draw conclusions by highlighting particular data points or removing distracting noise by modifying the transfer function.

- Interactive interfaces are offered by several volume rendering programs, allowing for real-time transfer function modification. Finding the best transfer function for the data is made simpler since users may try out several mappings and see the effects right away in the display.

- Transfer functions have several uses, notably in engineering, science, and medicine. Transfer functions are crucial tools for research and decision-making in these domains because they offer a powerful way to visualize complicated volumetric data.

In conclusion, transfer functions are essential to volume rendering for improving visuals, offering flexibility, enabling interactivity, and servicing numerous applications. Making sense of complicated 3D information and coming to wise judgments would be far more difficult without transfer functions.

1.6 The Volume Rendering Integral

Every physically-based volume rendering algorithm evaluates the volume rendering integral in one way or the other, even if viewing rays are not employed explicitly by the algorithm. The most basic, but also the most flexible, volume rendering algorithm is ray casting, which is introduced in Section 1.7. It might be considered the "most direct" numerical method for evaluating this integral. More details are covered later on, but for this section, it suffices to view ray casting as a process that, for each pixel in the image to render casts a single ray from the eye through the pixel's center into the volume, and integrates the optical properties obtained from the encountered volume densities along the ray.

Note that this general description assumes both the volume and the mapping to optical properties to be continuous. In practice, of course, the volume data are discrete, and the evaluation of the integral is approximated numerically. In combination with several additional simplifications, the integral is usually substituted by a Riemann sum. We denote a ray cast into the volume by $x(t)$ and parameterize it by the distance t from the eye. The scalar value corresponding to a position along the ray is denoted by $s(x(t))$. If we employ the emission-absorption model, the volume rendering equation integrates absorption coefficients $\kappa(s)$ (accounting for the absorption of light), and emissive colors $c(s)$ (accounting for radiant energy actively emitted) along a ray. To keep the equations simple, we denote emission c and absorption coefficients κ as a function of the eye distance t instead of the scalar value s:[28]

$$c(t) := c(s(\mathbf{x}(t))) \quad \kappa(t) := \kappa(s(\mathbf{x}(t))) \tag{1}$$

Figure 1.16 illustrates the idea of emission and absorption. An amount of radiant energy, which is emitted at a distance $t = d$ along the viewing ray is continuously absorbed along the distance d until it reaches the eye. This means that only a portion c' of the original radiant energy c emitted

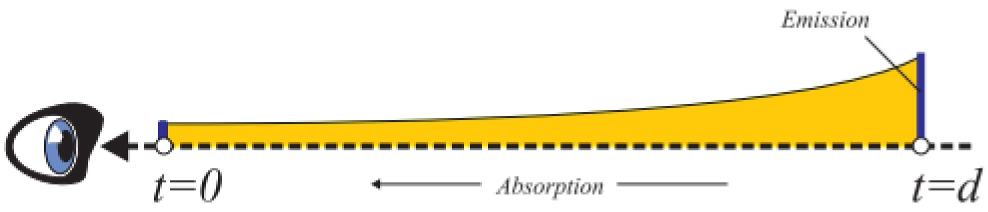


Figure 1.16: An amount of radiant energy emitted at $t = d$ is partially absorbed along the distance d.[28]

at $t = d$ will eventually reach the eye. If there is a constant absorption $\kappa = \text{const}$ along the ray, c' amounts to

$$c' = c \cdot e^{-\kappa d} \tag{2}$$

However, if the absorption k is not constant along the ray, but itself dependent on the position, the amount of radiant energy c' reaching the eye must be computed by integrating the absorption coefficient along the distance d:

$$c' = c \cdot e^{-\int_0^d k(\hat{t})d\hat{t}} \tag{3}$$

The integral over the absorption coefficients in the exponent,

$$\tau\left(d_1, d_2\right) = \int_{d_1}^{d_2} \kappa(\hat{t})d\hat{t} \tag{4}$$

is also called the optical depth. In this simple example, however, the light was only emitted at a single point along the ray. If we want to determine the total amount of radiant energy C reaching the eye from this direction, we must take into account the emitted radiant energy from all possible positions t along the ray:

$$C = \int_0^\infty c(t) \cdot e^{-\tau(0,t)}dt \tag{5}$$

In practice, this integral is evaluated numerically through either front-to-back or back-to-front compositing (i.e., alpha blending) of samples along the ray, which is most easily illustrated in the method of ray casting. Ray casting usually employs front-to-back compositing.

1.7 Ray casting

Ray casting [43] is an image-order direct volume rendering algorithm, which uses straightforward numerical evaluation of the volume rendering integral (Equation 5). For each pixel of the image, a single ray2 is cast into the scene. At equispaced intervals along the ray, the discrete volume data are resampled, usually using tri-linear interpolation as a reconstruction filter. That is, for each resampling location, the scalar values of eight neighboring voxels are weighted according to their distance to the actual location for which a data value is needed. After resampling, the scalar data value is mapped to optical properties via a lookup table, which yields an RGBA quadruplet that subsumes the corresponding emission and absorption coefficients [43] for this location. The solution of the volume rendering integral is then approximated via alpha blending in either front-to-back or back-to-front order, where usually the former is used in ray casting. The optical depth T (Equation **??**), which is the cumulative absorption up to a certain position x(t) along the ray, can be approximated by a Riemann sum[28].

$$\tau(0,t) \approx \tilde{\tau}(0,t) = \sum_{i=0}^{\lfloor t/\Delta t \rfloor} \kappa(i \cdot \Delta t)\Delta t \tag{6}$$

with Delta-t denoting the distance between successive resampling locations. The summation in the exponent can immediately be substituted by a multiplication of exponentiation terms:

$$e^{-\bar{\tau}(0,t)} = \prod_{i=0}^{\lfloor t/\Delta t \rfloor} e^{-\kappa(i \cdot \Delta t)\Delta t} \tag{7}$$

Now, we can introduce the opacity A, which is well-known from traditional alpha blending, by defining

$$A_i = 1 - e^{-\kappa(i \cdot \Delta t)\Delta t} \tag{8}$$

and rewriting Equation 7 as:

$$e^{-\bar{\tau}(0,t)} = \prod_{i=0}^{\lfloor t/d \rfloor} (1 - A_j) \tag{9}$$

This allows the opacity Ai to be used as an approximation for the absorption of the i-th ray segment, instead of absorption at a single point. Similarly, the emitted color of the i-th ray segment can be approximated by:

$$C_i = c(i \cdot \Delta t)\Delta t \tag{10}$$

Having approximated both the emissions and absorptions along a ray, we can now state the approximate evaluation of the volume rendering integral as: (denoting the number of samples by:

$$n = \lfloor T/\delta t \rfloor$$

$$\tilde{C} = \sum_{i=0}^{n} C_i \prod_{j=0}^{i-1} (1 - A_i) \tag{11}$$

Equation 11 can be evaluated iteratively by performing alpha blending in either front-to-back or back-to-front order

Ray casting is a technique used in computer graphics to create a 2D representation of a 3D scene. Using this method, rays from the observer's viewpoint or a camera are projected onto the 3D world through each pixel of a 2D picture plane. The method calculates the hue and brightness of each pixel on the picture plane by determining where the rays meet with objects in the scene. The first step in the ray-casting process is determining the picture plane and the observer's perspective. The next step is to project a ray from the perspective across each pixel on the picture plane. The technique then runs intersection checks between the rays and objects present in the scene to determine the location of the intersection. Ultimately, the intensity and color of the pixel are determined by the characteristics of the crossed objects, such as their lighting, texture, and color. Although this technique necessitates performing numerous intersection tests for each pixel on the image plane, it is highly effective for rendering intricate scenes with realistic shading and lighting effects. Ray casting finds its applications in various fields, including video games, virtual reality, and architectural visualization.

The use of ray casting in the context of volume rendering is similar to that of ray tracing in surface-based computer graphics, but unlike ray tracing, volume rendering only uses primary rays without taking into account secondary rays related to shadows, reflections, or refraction. Ray casting is an image-order technique that is well-suited to this task as there are no surfaces to be rendered; instead, the volume must be carefully traversed in small steps with a light source.
[60]

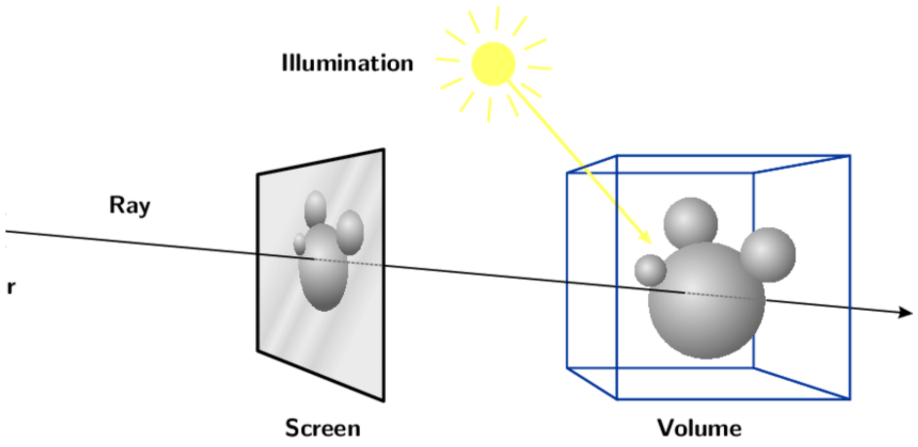


Figure 1.17: Illustration of raycasting. A ray is cast from the viewer through a pixel on the screen. It marches through the volume and integrates the contribution of light, colored samples shaded by the illumination. The result is stored in a pixel and the procedure is performed for all pixels.[46]

For creating 3D pictures from 2D medical images, such as CT (Computed Tomography) and MRI (Magnetic Resonance Imaging) scans, ray casting is a common approach. These are a few typical methods for ray casting in medical imaging:[21]

By projecting rays over a 3D volume of data collected from the medical scan, the volume rendering approach produces a 3D picture. The viewer's perspective causes the rays to be cast, and the characteristics of the voxels (volume elements) along its path define each ray's hue and intensity. After sampling the voxels, the colors and ray intensities are combined to create the final image.

Maximum Intensity Projection (MIP): This method is used to draw attention to the areas of a medical picture that have the most intensity. In order for it to operate, rays are projected through the 3D volume and the greatest intensity value along each ray is chosen. A 2D projection of the 3D volume that highlights the areas of greatest intensity is the resultant picture.[21]

Use the surface rendering technique to create a 3D representation of an object's surface. It works by tracking rays from the viewpoint of the observer and determining where they intersect the surface of the item. A 3D depiction of the object's surface is shown in the final image.

With the ray casting approach, the various components of the 3D volume are given separate colors and opacities using a transfer function. The transfer function, which transforms voxel values into colors and opacities, enables the presentation of intriguing structures inside the medical scan.[37]

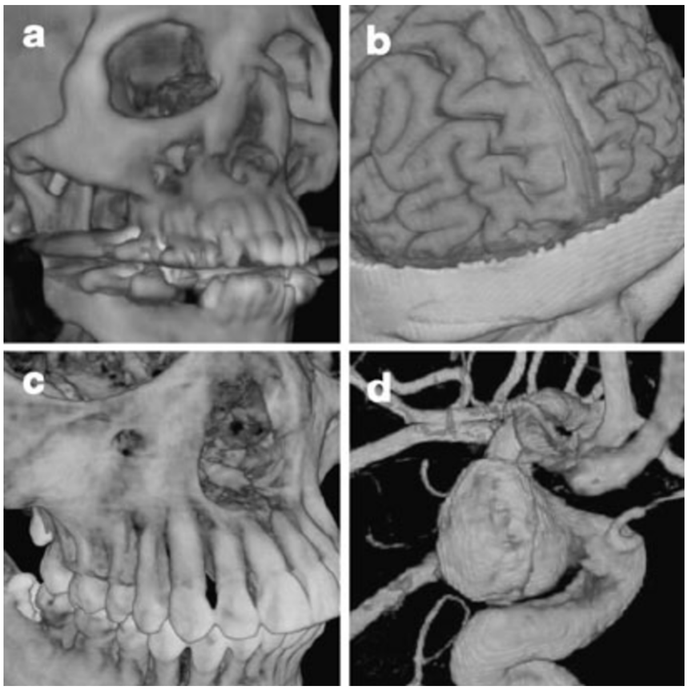


Figure 1.18: Medical images rendered with the GPU-based raycasting: a CT skull, b MR brain, c CT jaw, and d MR cerebral blood vessel. [46]

Overall, ray-casting methods are crucial for medical imaging because they make it possible to produce 3D representations that can help with disease diagnosis and therapy.

1.8 Volume rendering tools

VTK (Visualization Toolkit) and MITK (Medical Imaging Interaction Toolkit) are two popular open-source software libraries used for volume rendering and visualization. VTK is a versatile library developed by Kitware, providing a wide range of rendering techniques and advanced features for scientific and medical visualization. MITK, built on top of VTK, is specifically designed for medical imaging applications, offering a user-friendly interface and specialized tools for medical image processing and analysis. Both VTK and MITK are widely used in the medical and scientific communities due to their flexibility and active developer communities.

1.8.1 VTK (Visualization Toolkit)

For 3D computer graphics, image processing, and visualization, there is the Visualization Toolkit (VTK), an open-source software program. It was first created by the GE Corporate Research and Development Center in 1993, and it is currently being maintained by Kitware, a software provider that specializes in open-source solutions.[61]

VTK is intended to be a strong and adaptable tool for developing representations of difficult scientific data. It offers a large selection of data processing and visualization methods in addition to a complete set of software tools for building unique visuals. Key ideas guiding VTK include the following:[61, 62]

- Data Model: With VTK's data model, users may represent intricate data structures like meshes, pictures, and volumes. The data model is made up of hierarchically arranged data elements. Each data item has a unique collection of features, including colors and opacities.

- The VTK rendering pipeline consists of a series of data processing and visualization phases that take input data and turn it into a visual representation. Data sources, filters, mappers, actors, and renderers are all components of the pipeline. Data sources provide the pipeline with input data, filters change the data, mappers turn the data into primitives for rendering graphics, actors specify how the primitives should be produced, and renderers show the visuals on the screen.

- Visualization Algorithms: The visualization toolkit (VTK) comes with a significant range of data processing and visualization techniques, such as filters for smoothing, decimation, and interpolation, as well as algorithms for volume rendering, isosurface extraction, and streamlines. To produce unique visuals, a variety of combinations of these algorithms can be used.

- VTK is meant to be extremely extendable. The system's capability may be increased by users by adding new data items, filters, mappers, actors, and renderers. Users of VTK may generate unique visualizations by writing short scripts in scripting languages like Python thanks to this functionality.

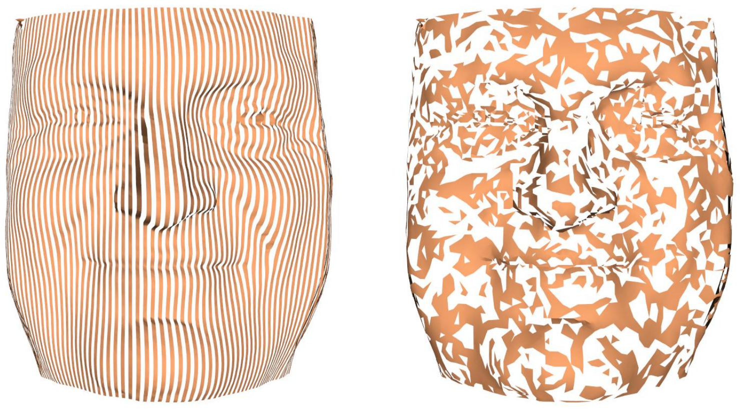


Figure 1.19: Generating triangle strips from a polygonal data set. On the left, the original laser range data is stripped; on the right, an unstructured mesh (after decimation) is stripped. [61]

In general, VTK offers a strong and adaptable tool for producing representations of challenging scientific data. Researchers and developers working on a variety of scientific subjects frequently use it because of its broad collection of data processing and visualization methods, extensibility, and support for scripting languages.

1.8.2 MITK (Medical Imaging Interaction Toolkit)

Medical Imaging Interaction Toolkit, sometimes known as MITK, is a potent open-source software toolkit created to make it easier to create unique medical imaging applications. The main goal of MITK is to offer a collection of tools and frameworks that enable programmers to create interactive medical imaging applications for a variety of uses, including image processing and segmentation, visualization, and registration.

A graphical user interface framework included in the C++ based software package known as MITK makes it simple and quick for programmers to construct interactive applications. A collection of pre-built modules for doing activities related to medical imaging are also included in the software package. These modules include those for segmenting MRI pictures, registering PET and CT images and visualizing 3D medical data.

The global medical imaging community, including researchers, doctors, and developers, favors MITK because of its versatility. It has shown to be very helpful in medical imaging research due to its capacity to ease the implementation of fresh algorithms and methods for analyzing and presenting medical pictures.

The modular design of MITK and its open-source status add to its adaptability and extensibility. The program may be expanded and modified by developers to meet unique needs, making it an excellent option for a variety of medical imaging applications. Furthermore, MITK's open-source nature guarantees that a thriving community of programmers and users continuously updates and enhances the software.[3]

MITK's use and development are motivated by a number of fundamental principles. These core concepts include:[3]

- Open-source: MITK is an open-source software toolkit, which entails that anybody is allowed to use, alter, and distribute its source code. This open-source nature invites developers to participate in cooperation and make improvements to the toolkit.
- Modularity: The functionality of MITK is separated into separate components or modules due to the design's modularity. The toolkit is adaptable and effective because of its modularity, which enables developers to pick and choose only the components they want.
- Extensibility: MITK is created to be extendable, allowing programmers to add new modules or change existing ones to suit their unique requirements. This flexibility makes sure that the toolkit may change with the times and continue to be useful.
- User-centered design: MITK attempts to deliver a user-friendly interface and intuitive processes since it was created with the end user in mind. This user-centered design encourages usability and makes sure that a variety of users may access the toolkit.
- Focused on medical imaging: Because MITK was created expressly for use in medical imaging applications, it has functionality that is adapted to the special requirements of those working in the field. The toolkit is very relevant and efficient for the purpose it is meant for thanks to this concentration.

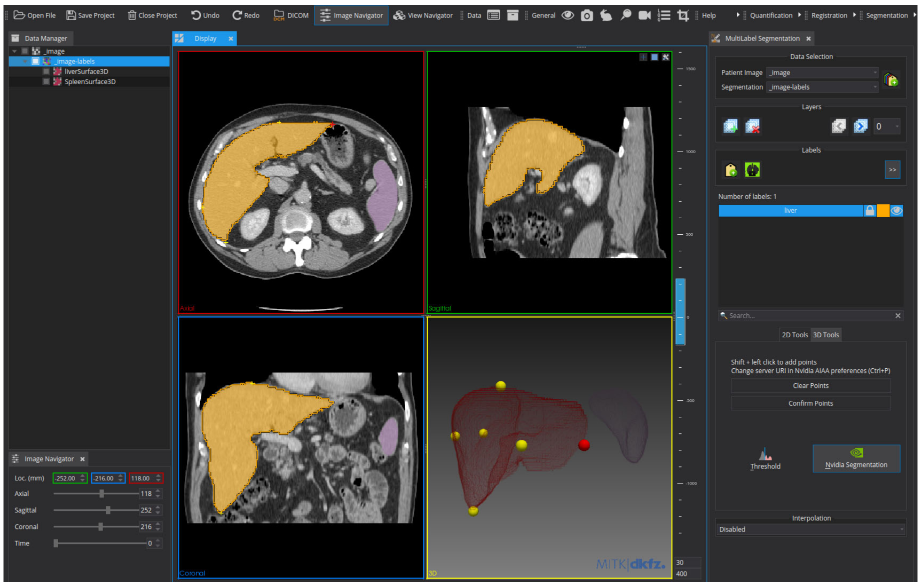


Figure 1.20: MITK ReleaseNotes 2018.04.2 [2]

1.9 Conclusion

The concepts and foundations of volume rendering provide a solid understanding of this essential technique for visualizing volumetric data. We have explored various aspects of volume rendering, starting with an introduction that highlights its significance in generating 2D images from 3D data. The discussion encompassed volumetric data, including different types and their representation.

Techniques in volume rendering were examined, covering both direct and indirect approaches. Direct volume rendering techniques, such as Maximum Intensity Projection (MIP) and Shear-Warp, enable the generation of visualizations that emphasize the most prominent features within the volume. On the other hand, indirect volume rendering techniques, such as Marching Cubes, focus on extracting surfaces from the volumetric data.

The transfer function, a critical component in volume rendering, was discussed as a means to map scalar values to visual attributes, enabling the depiction of various properties within the volume. The Volume Rendering Integral, based on the concept of ray casting, allows for the calculation of pixel values by integrating along the ray path.

Furthermore, we explored volume rendering tools that facilitate the implementation and utilization of these techniques. VTK (Visualization Toolkit) and MITK (Medical Imaging Interaction Toolkit) were highlighted as prominent tools that offer comprehensive capabilities for volume rendering.

Chapter 2

Volumetric Ray casting

2.1 Introduction

volumetric ray casting is a powerful technique used in volume rendering to generate three-dimensional visualizations from volumetric datasets. Volumetric ray casting offers a versatile and efficient approach to accurately capture the intricate details and structures within volumetric data, enabling researchers and professionals to gain valuable insights and make informed decisions.

begins with an examination of the sampling process in volumetric ray casting. Sampling involves casting rays through the volume and gathering information along their paths to determine the properties of the volume at specific points. Within the realm of sampling[38],[37], we delve into irregular sampling techniques, including adaptive sampling and hierarchical sampling. These techniques aim to optimize the sampling process by adapting the sampling density based on the local features of the volume or by utilizing a hierarchical structure to efficiently sample the volume.

Next, the discussion moves on to the concept of empty space skipping[29],[66], a fundamental aspect of volumetric ray casting. Empty space skipping techniques exploit the sparsity of the volume data to accelerate the rendering process by efficiently skipping empty regions that do not contribute to the final image. This section explores various empty space skipping methods and their advantages in terms of computational efficiency and reduction of unnecessary computations.

To further enhance the acceleration of empty space skipping, the chapter explores two popular data structures: KD-tree and Octree[69],[45]. These hierarchical structures partition the volume into smaller regions, allowing for efficient traversal and identification of empty regions. Additionally, we delve into Distance Transform Empty Space Skipping, a technique that employs distance maps to accelerate the empty space skipping process by efficiently computing distances to the nearest object surfaces.

distance maps[18], specifically focusing on the concept of signed distance fields. Signed distance fields provide a compact representation of the distance from each voxel in the volume to the nearest surface, enabling efficient and accurate calculations during the ray-casting process.

Furthermore, the chapter introduces the application of Chebyshev distance maps[17] for volumetric ray casting acceleration. Chebyshev distance maps leverage the Chebyshev distance metric to estimate the maximum distance to the nearest surface, enabling faster identification of empty regions and reducing the number of ray-object intersection tests.

2.2 Sampling

Ray casting, sampling, and integration are three processes in the volumetric ray casting sampling process. Ray casting entails following a ray from the camera through the volume to where it escapes at the rear of the volume. The beam travels via voxels, or sample sites in the volume, which are utilized to sample the data, along the route.

Calculating the data values at the sample sites is a stage in the sampling process. To estimate the data values between sample sites in medical imaging, interpolation techniques like trilinear or tri-cubic interpolation may be used. The data may be directly accessible at the sample points in scientific simulations.

In order to determine the color and opacity of the pixel, the sampled values along the ray are combined in the integration stage. Techniques like compositing, which adds up the contributions of each sample point along the ray depending on its opacity, or shading, which employs lighting models to determine the final color of the pixel, are used to do this.

The sampling method, which dictates the arrangement and spacing of the sample points along the ray, as well as the sampling density, which affects the accuracy and smoothness of the final picture, are a few of the variables to take into account while sampling in volumetric ray casting. Adaptive sampling methods may occasionally be used to improve sampling density and lessen picture artifacts.

In terms of volumetric ray casting, sampling is an important stage since it affects the precision and quality of the final image. From volumetric data, high-quality and aesthetically pleasing pictures may be generated using the right sampling procedures and methodologies
.[38]

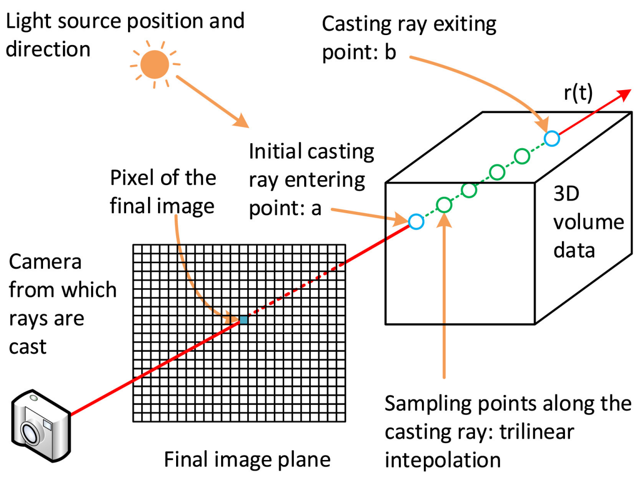

Light source position and direction
Casting ray exiting point: b
r(t)
Initial casting ray entering point: a
Pixel of the final image
3D volume data
Camera from which rays are cast
Sampling points along the casting ray: trilinear intepolation
Final image plane

Figure 2.1: Pipeline of raycasting calculation on GPU graphics hardware: setting camera and image plane, casting ray into 3D texture volume, adding light source and conducting trilinear interpolation based texture sampling.[70]

In volumetric ray casting, a variety of sampling methods may be applied to extract samples from the 3D volume data. Several of the typical sampling methods for volumetric ray casting are listed below[41]:

2.2.1 Irregular sampling

Irregular sampling[13] is a method used in computer graphics and visualization to minimize the number of samples necessary in the rendering process. In volumetric raycasting, irregular sampling is very essential since it may considerably lower the computing complexity of the operation while improving the speed and accuracy of the final image.

One of the key advantages of irregular sampling is that it allows for selective sampling of regions of interest in the volume. This is especially advantageous when working with big and complicated volumes, where uniform sampling can be wasteful and time-consuming. Instead, irregular sampling approaches try to decrease the number of samples necessary while keeping the general structure and features of the volume.

Adaptive sampling is one way to irregular sampling that samples parts of the volume based on their value to the final image. Heuristics are usually used by adaptive sampling algorithms to identify which portions of the volume to sample at increasing resolution. High gradients or rapid changes in intensity, for example, are likely to necessitate greater resolution sampling to accurately capture surface features. Adaptive sampling algorithms may also employ error estimation techniques to assess the difference between the current image approximation and the final picture, and selectively enhance the resolution in locations with large errors.

Adaptive sampling, which chooses which volumetric sections to sample depending on their significance to the final image, is one method for dealing with irregular sampling. Heuristics are frequently used by adaptive sampling algorithms to choose which parts of the volume to sample at increasing resolution. For instance, better resolution sampling is likely needed to adequately capture the surface characteristics in areas with steep gradients or abrupt changes in intensity. In order to boost the resolution in areas with significant errors, adaptive sampling algorithms may also employ error estimation techniques to estimate the error between the current approximation of the picture and the final image.

Other irregular sampling approaches can be employed to increase the efficiency and accuracy of the rendering process in addition to adaptive and hierarchical sampling. Use empty space skipping, for instance, to swiftly go past areas of the volume that are devoid of shape. As a result, the rendering process may use much less samples. Additionally, you may utilize distance maps to only selectively sample the parts of the volume that are closest to the surface. This can further cut down on the number of samples needed for rendering, especially in areas of the volume with little geometry.

Despite the benefits of irregular sampling, there are some drawbacks and difficulties with the method as well. One drawback is that, as compared to conventional uniform sampling methods, irregular sampling algorithms might be more complex to develop. If not done properly, they may potentially add artifacts or faults to the final image. The portions of the volume to sample at greater resolution may also need to be determined in a pre-processing phase for some irregular sampling approaches, which might increase computing costs.

The use of irregular sampling methods in volumetric raycasting is crucial because they enable rendering operations to use fewer samples while maintaining the volume's general structure and fine details. The two most popular irregular sampling methods are adaptive and hierarchical sampling, however there are other methods that may be utilized to increase rendering accuracy and efficiency, including empty space skipping and distance maps. Although irregular sampling methods can be more difficult to execute than typical uniform sampling methods, they can greatly lower the computing complexity of the process and allow for the real-time rendering of intricate and realistic 3D pictures.

2.2.1.1 Adaptive sampling

To modify the sample rate dependent on the complexity of the data characteristics, adaptive sampling is a technique used in volume rendering. By adjusting the sample rate to the areas of the data that demand more or less information, it tries to provide photographs of greater quality while requiring less computing work.

The conventional method of sampling is creating a regular grid structure out of the volume data and computing the color and opacity values at regularly spaced sample points. However, this approach can result in artifacts like the staircase effect, where the regular grid structure is not aligned with the features in the volume data and can be computationally expensive, especially in areas of data with high detail.

By carefully changing the sampling rate depending on the complexity of the data attributes, adaptive sampling overcomes these problems. More detailed data regions are sampled more frequently, whilst less detailed data regions are sampled less frequently. As a consequence, there are fewer artifacts and lower processing expenses for the photographs, which are of greater quality.

Different strategies can be used to implement adaptive sampling. One method is to describe the volume data using a hierarchical data structure, such as an octree or a binary tree. Recursively breaking the volume data into smaller sections until each region has a specific amount of voxels allows for the construction of the tree structure. The sample rate may then be deliberately increased or decreased depending on the size and complexity of the areas using the tree structure.

Utilizing the technique known as gradient-based adaptive sampling is an additional strategy. This method calculates the complexity of the data features by estimating the gradient magnitude of the volume data. greater gradient magnitude portions of the data are thought to be more complicated and are sampled at a greater pace, whereas lower gradient magnitude regions of the data are thought to be less complex and are sampled at a lower rate.

Utilizing the technique known as error-based adaptive sampling is a third strategy. This method calculates the difference in inaccuracy between the displayed and intended images and modifies the sample rate accordingly. Regions of the data that are more responsible for the mistake are sampled more frequently, while those that are less responsible for the error are sampled less frequently.

The quality and effectiveness of volume rendering may be greatly enhanced by adaptive sampling. It is not without its restrictions, though. The adaptive sampling procedure itself may be computationally costly, particularly when employing sophisticated data structures, gradient-based, or error-based methods. Another drawback is that, if not performed appropriately, adaptive sampling might generate its own artifacts, like blurring or ghosting.[27]

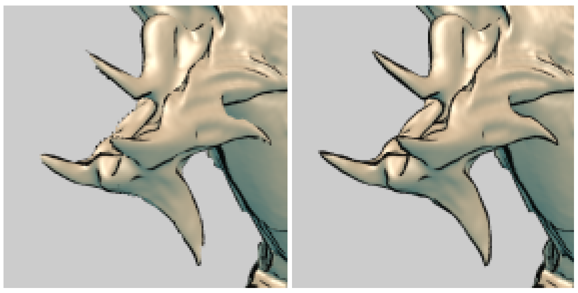


Figure 2.2: The left image illustrates a small detail of the Asian dragon model with a sampling rate of 0.5. On the right, adaptive sampling increases the sampling rate to 4.0 close to the isosurface. Note that except at the silhouettes, there is no visible difference due to the iterative refinement of intersections.[27]

adaptive sampling is a method used in volume rendering to modify the sample rate according to the complexity of the data characteristics. By carefully adjusting the sample rate in data locations that call for more or less information, it can provide pictures of greater quality at a cheaper computing cost. Hierarchical data structures, gradient-based approaches, and error-based techniques are just a few of the many ways that adaptive sampling may be applied. The production of high-quality photographs, however, necessitates careful consideration of the constraints and potential artifacts of adaptive sampling.

2.2.1.2 Hierarchical sampling

In computer graphics and visualization, a method called hierarchical sampling is employed to increase the effectiveness and precision of volumetric rendering. The volume is divided into nested sub-volumes, with each sub-volume being sampled at a different resolution, with the closest sub-volumes near the camera receiving the greatest resolution and the farthest sub-volumes receiving lower resolutions.

The key benefit of hierarchical sampling is that it may drastically cut down on the number of samples needed during rendering while still maintaining the fine details and volumetric structure. When generating big and complicated volumes, this is especially helpful because uniform sampling may be time- and resource-intensive. Hierarchical sampling can lessen the computational complexity of the rendering process and increase the speed and accuracy of the final image by choosing sampling locations of interest at greater resolution and lowering the sample rate in more distant parts.

Octree-based sampling, which requires partitioning the volume into a hierarchical octree structure, is one method of employing hierarchical sampling. First, the volume is divided into eight equal subvolumes, and then each of those subvolumes is further divided into eight subvolumes, and so on. The sub-volumes that are nearest to the camera receive the greatest resolution sampling, while those that are farther away receive lesser resolutions. By sampling distant sub-volumes at a lesser resolution while keeping the features of nearby sub-volumes at a greater resolution, this method minimizes the number of samples needed.

Mipmapping, a distinct method of hierarchical sampling, entails making a number of pre-filtered pictures of the volume at various resolutions. For the sub-volumes that are nearest to the camera, the highest quality picture is used, while lower resolution photos are used for sub-volumes that are farther away. The precision of the resulting picture can be increased by combining mipmapping with additional sampling methods like trilinear interpolation.

Multiresolution modeling, which involves modeling the volume at many levels of detail, is a method linked to hierarchical sampling. Wavelet transformations and multigrid approaches, which enable the volume to be represented at many degrees of detail, can be used to create multiresolution models. When generating volumes that are highly detailed in some areas but uniform in others, this method can be quite helpful.

Hierarchical sampling can be used to increase the quality of the final image in addition to lowering the number of samples needed. Hierarchical sampling can better capture surface features and lessen aliasing problems by choosing sampling locations of interest at higher resolution. By lowering the quantity of rays that must be tracked through the volume, hierarchical sampling can also increase the effectiveness of other rendering techniques like ray marching or ray casting.

Hierarchical sampling provides benefits, but it also has certain drawbacks and difficulties. Choosing the right amount of information for each sub-volume is one of the challenges. The best settings may need to be determined via some trial and error since this necessitates balancing the trade-off between accuracy and processing efficiency. If performed incorrectly, hierarchical sampling can also result in artifacts or flaws in the final picture, such as discontinuities at the borders between sub-volumes or blurring of features in areas where the resolution abruptly changes.

In volumetric rendering, hierarchical sampling is a crucial approach that may greatly increase the process' accuracy and efficiency. Hierarchical sampling techniques include octree-based sampling, mipmapping, and multiresolution modeling. Hierarchical sampling can decrease the computational complexity of the rendering process and enhance the speed and quality of the final image by choosing sampling locations of interest at greater resolution and lowering the sample rate in more distant regions. Although hierarchical sampling has significant drawbacks, it is nonetheless a vital method for producing intricate and realistic 3D pictures in real time[73].

2.3 Empty space skipping

Adaptive sampling, which dynamically modifies sample rate based on scene complexity and current rendering algorithm performance, may also be used in conjunction with empty space skipping. Developers may build rendering systems that effectively analyze massive amounts of data and provide high-quality representations in real-time by combining adaptive sampling with Empty space skipping.

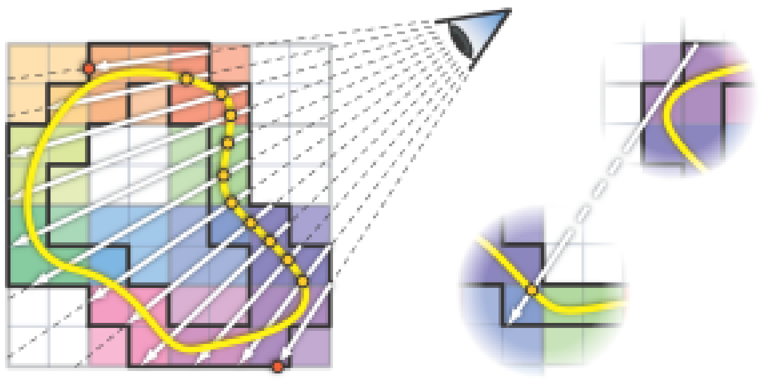


Figure 2.3: Ray-casting with object-order empty space skipping. The bounding geometry (black) between active and inactive blocks that determines start and exit depths for the intersection search along rays (white) encloses the isosurface(yellow). Colored bricks of 2x2 blocks reference bricks in the cached texture. White bricks are not in the cache. Actual ray termination points are shown in yellow and red, respectively.[26]

To increase the effectiveness of the rendering process, volumetric ray casting employs the empty space skipping approach. It operates by ignoring areas of void in the volume, which minimizes the number of voxels that must be examined for ray crossings. In situations with big volumes or complicated geometries, this can considerably speed up rendering.

The method used to achieve the empty space skipping approach looks for intersections between the ray and the volume data. The method determines if the ray crosses over an empty area or collides with a volume voxel. The method stops the ray early and returns a default color value if the ray crosses across empty space. The method keeps casting the ray and accumulating color and opacity data until the ray departs the volume or reaches a maximum depth if the ray meets with a voxel.

By minimizing the amount of voxels that must be assessed during the sampling process, empty space skipping is a crucial approach that may considerably enhance the performance of volumetric rendering.

The empty space skipping approach has the potential to significantly improve the efficiency of volumetric ray casting overall. To ensure that important details are not missed throughout the rendering process, it calls for precise control.[66]

2.4 Empty space skipping acceleration

The application of Empty space skipping acceleration techniques increases the effectiveness and speed of Empty space skipping by reducing the number of voxels that need to be examined for intersections with the ray by skipping over areas of unoccupied space in the volume. When great performance is necessary, like in real-time visualization applications or scientific simulations, these approaches are very crucial. To further enhance the rendering process, other acceleration techniques can be utilized in concert with Empty space skipping.

2.4.1 KD-tree

Often employed in computer graphics, the KD-tree[33] data structure divides 3D space into smaller areas to provide effective spatial indexing and data access. The method of empty space skipping, which is used to speed up ray casting and enhance rendering efficiency, is one way that KD-tree is utilized in computer graphics.

Ray casting uses the empty space skipping approach to bypass areas of space that are devoid of any objects, hence minimizing the number of ray-object intersection checks that must be run. In order to identify which areas of the scene are empty and which are not, the scene is first divided into smaller regions using the empty space skipping technique.

One such spatial data structure that is frequently utilized in empty space skipping is the KD-tree. The fundamental procedure for constructing a KD-tree includes recursively splitting the space into two areas along an axis-aligned plane, keeping doing so until each region is vacant or just partially occupied. As a result, a binary tree structure is produced, with each node denoting an area of space and two child nodes denoting the subregions that were produced as a result of the partitioning.

When performing empty space skipping with a KD-tree, the rays are cast through the tree in a similar way to ray casting. However, instead of testing for intersections with objects at each node, the algorithm checks whether the node represents an empty region of space. If the node is empty, the algorithm skips over the region and moves on to the next node. If the node is not empty, the algorithm continues to traverse down the tree and test for intersections with objects as usual.

When it comes to empty space skipping, KD-tree has a number of benefits. To swiftly determine whether a region of space is empty or not, fast spatial indexing and data access are essential, which KD-tree enables. As a result, there may be a large decrease in the number of intersection tests required, which will speed up rendering.

KD-tree also has the benefit of enabling adaptive sampling and level-of-detail rendering. The regions of space that are most important to the viewing direction may be found by moving through the tree, at which point the sample rate can be changed. The sample is concentrated on the key elements of the picture, allowing for quicker rendering and improved image quality.

Further enhancing rendering performance is possible by combining KD-tree with additional acceleration methods as bounding volume hierarchies (BVHs). For instance, a hybrid approach combining the KD-tree and the BVH may be used to swiftly go through empty space and then carry out intersection tests with objects that are most likely to be struck by the rays.

An effective data structure that is frequently used in computer graphics to speed up ray casting and enhance rendering efficiency is the KD-tree. Faster intersection testing and shorter rendering times are made possible by its effective division of space into smaller parts and ability to spot unoccupied areas. KD-tree is anticipated to be a key component in the development of the next rendering algorithms and software because of the growing need for realistic and high-quality visuals across a variety of sectors.

The location of points or objects inside a space is utilized to divide it into smaller areas using a KD-tree data structure. A KD-tree is used in the context of Empty space skipping to organize the volume data into a hierarchical structure that can be quickly browsed to identify and skip over empty space.

Recursively splitting the volume data along the axis with the highest variance results in the construction of the KD-tree. As a consequence, a tree structure is created, with each node denoting a splitting of the volume data into two areas based on a dividing plane. The tree may be effectively navigated by first determining which area of the volume data the ray interacts with, and then recursively exploring the appropriate child nodes until empty space is skipped over and a non-empty region is identified.

Since a KD-tree enables quick detection and skipping over of empty space, using one in ESS can significantly enhance rendering speed. The method can rapidly decide which sections of the volume data need to be assessed and which may be passed over altogether by moving through the tree in a depth-first fashion.[69]

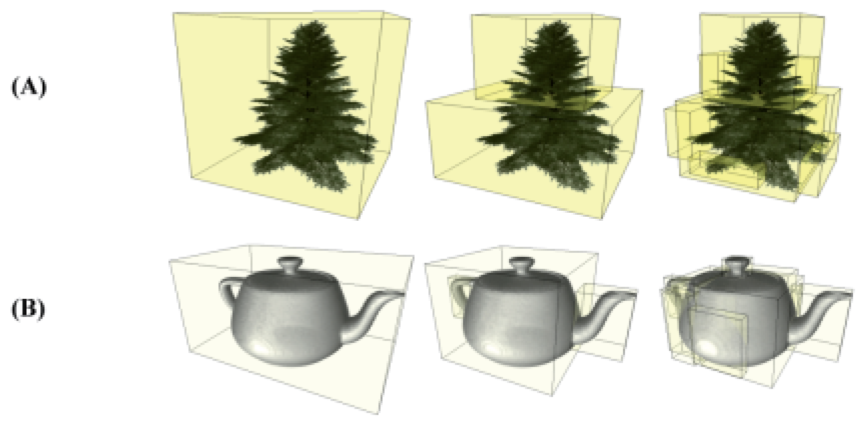


Figure 2.4: An illustration of our empty-space removing method. Empty voxels are progressively removed from the original volume with a kd-tree based spatial partitioning scheme.[65]

2.4.2 Octree

This approach focuses on casting volume rays while avoiding transparent zones. Volume ray casting is depicted in a block diagram in Figure 2.5 . An object boundary must first be swiftly reached by a method by skipping through transparent regions. Even when sample points are situated in a transparent area, color and opacity values are always generated in the traditional volume ray casting approach. However, empty-space jumping techniques are now frequently employed since this is ineffective. Second, the ray computes a precise scalar value by determining if the current sample point is transparent or not using a resampling filter.

The process then calculates a color value for the sample point by assessing gradient vectors in the surrounding eight voxels of a cell if it is determined that the sample point is not transparent. These steps are repeated for each sample point until the early ray termination technique causes the cumulative opacity to equal 1.0 .[45]

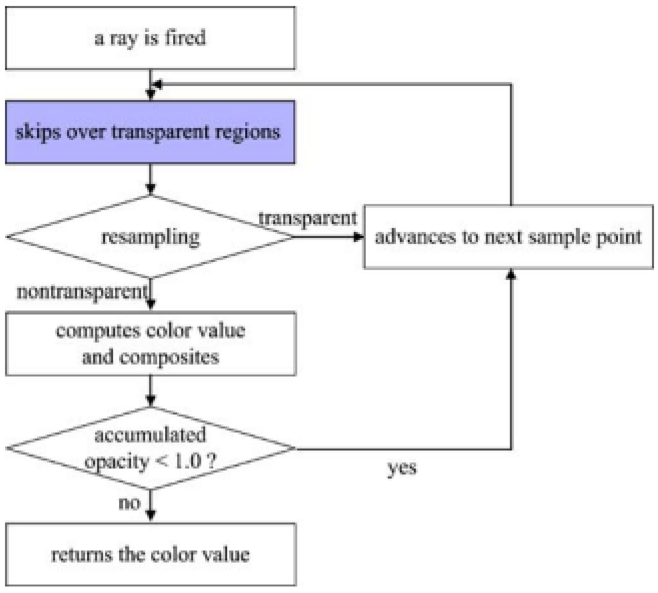

a ray is fired
skips over transparent regions
resampling
transparent
advances to next sample point
nontransparent
computes color value and composites
accumulated opacity < 1.0 ?
yes
no
returns the color value

Figure 2.5: .A block diagram of the volume ray casting. Since transparent regions cannot be contributed to the final image, we can accelerate the rendering speed when the regions are quickly skipped (shaded step).[45]

Since transparent sections don't add anything to the final image, we may speed up the volume ray casting pipeline to minimize the rendering time.

Because it requires little preprocessing and is independent of the opacity transfer function, the octree is one of the data structures that may pass through transparent regions.

A data structure called an octree is made up of several octants produced during the preprocessing phase. The transparency determination value(s) for each octant include the lowest and maximum ranges for each octant.

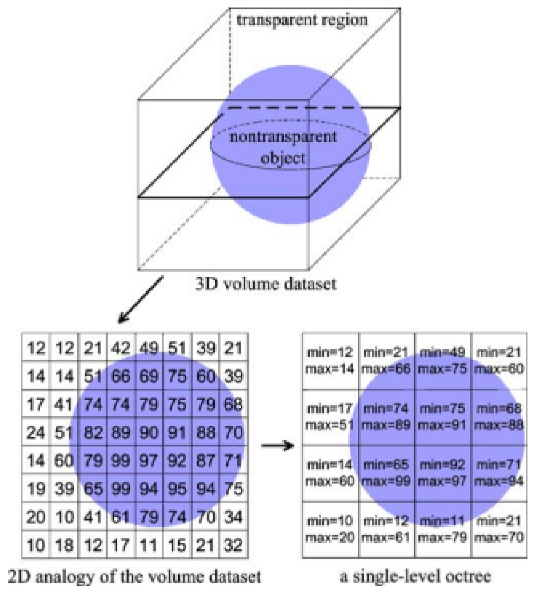
Up until it reaches the root node, the child node repeatedly forms a single parent node. Since rays jump over octants when the octants are assumed to be transparent by the transparency determination value(s), an octree-based technique can efficiently skip over transparent regions. demonstrates how to build a single-level octree from a volume dataset in Figure 2.6.

Following the discharge of a ray from each pixel, an octant's transparency determination value or values are referred to in order to decide whether to skip over the octant or not. The ray leaps across the octant when the value(s) are constrained to a transparent range (i.e., the present octant is a transparent octant).

Because we need to know the separation between two border voxels in this instance, intersection test techniques between the ray and the octant are utilized. But this calculation has an impact

rendering period. Additionally, the algorithm slows down rendering speed if there are a lot of transparent octants 2.8.

This process is repeated until the beam encounters an opaque octant. Conventional volume ray casting is used when the leaf node's transparency determination values are contained inside the nontransparent range[45].



12	12	21	42	49	51	39	21
14	14	51	66	69	75	60	39
17	41	74	74	79	75	79	68
24	51	82	89	90	91	88	70
14	60	79	99	97	92	87	71
19	39	65	99	94	95	94	75
20	10	41	61	79	74	70	34
10	18	12	17	11	15	21	32

min=12 max=14	min=21 max=66	min=49 max=75	min=21 max=60
min=17 max=51	min=74 max=89	min=75 max=91	min=68 max=88
min=14 max=60	min=65 max=99	min=92 max=97	min=71 max=94
min=10 max=20	min=12 max=61	min=11 max=79	min=21 max=70

Figure 2.6: An example of the construction of a single-level octree. For simplicity, the volume dataset is represented as a 2D array. The volume dimension is 8×8, and the quadrant dimension is 2×2. Assume that each quadrant has the minimum and the maximum values for representing the quadrant to determine the transparency.[45]

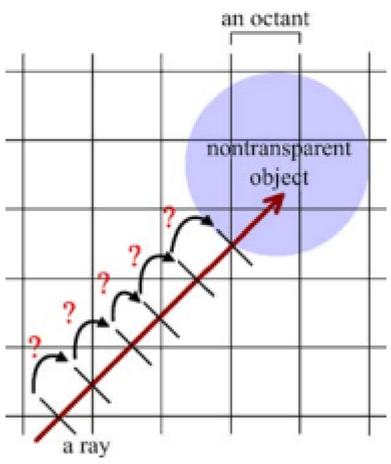


Figure 2.7: Since we estimate the distance from one boundary voxel to another boundary voxel for skipping over a transparent octant, an intersection test algorithm is required. In this figure, five mathematical computations are required to reach a nontransparent object.[45]

Algorithms for volumetric rendering may be made faster and more efficient overall by using Empty space-skipping acceleration methods. Developers may build robust rendering systems that quickly handle huge amounts of data and provide high-quality visuals in real-time by integrating Empty space skipping with KD-tree or octree methods.

2.4.3 Distance Transform Empty Space Skipping

The Distance Transform Empty Space Skipping (DT-ESS) technique is used in image processing and computer vision to quickly determine an image's distance transform. The distance transform of an image shows how far away each pixel is from the closest boundary or item in the image. Numerous applications, including image segmentation, pattern recognition, and form analysis, employ it.

The DT-ESS method is a development of the conventional distance transform algorithm, which calculates the distance transform of an image by repeatedly scanning it from top to bottom and left to right and calculating the distance between each pixel and the closest border pixel. Particularly for huge photos, this technique can be labor-intensive and sluggish.[9]

By avoiding voids in the picture during the computation of the distance transform, DT-ESS solves this problem. The technique begins by locating empty areas in the picture, or areas of the image where no object or border pixels are present. The distance transform computation is therefore bypassed across these places, making the technique quicker and more effective.[56]

Several stages may be separated out of the DT-ESS algorithm:

1. Recognizing empty spaces: The initial stage is to recognize empty spaces in the image. By scanning the image and locating areas devoid of any object or border pixels, this is accomplished.

2. Making a skip map: To store the regions that will be skipped during the computation of the distance transform, a skip map is made. The input picture's dimensions are used to create a binary image called a skip map, where each pixel is assigned a value of 1 if it represents an empty space and a value of 0 otherwise.

3. Initialization of the distance transform: Each pixel that corresponds to an item or boundary is initialized with a big value in the distance transform. Initialized with a value of 0, pixels that represent empty spaces are initialized.

4. The DT-ESS algorithm's forward pass scans the picture from left to right and top to bottom while calculating the distance between each pixel and its closest neighbor that is not in a skipped zone. The pixel is designated as skipped and its value is set to the maximum value if the closest neighbor is in a skipped zone.

5. Backward pass: During the backward pass, the picture is scanned from bottom to top and right to left, and the distance between each pixel and its closest neighbor that is not in a skipped zone is calculated for each pixel. The pixel is designated as skipped and its value is set to the maximum value if the closest neighbor is in a skipped zone.

6. Update of the distance transform values for pixels indicated as skipped during the forward and backward passes is the last stage in the distance transform process. By scanning the skip map and calculating the distance between each skipped pixel and its closest non-skip neighbor, the updated values are calculated.

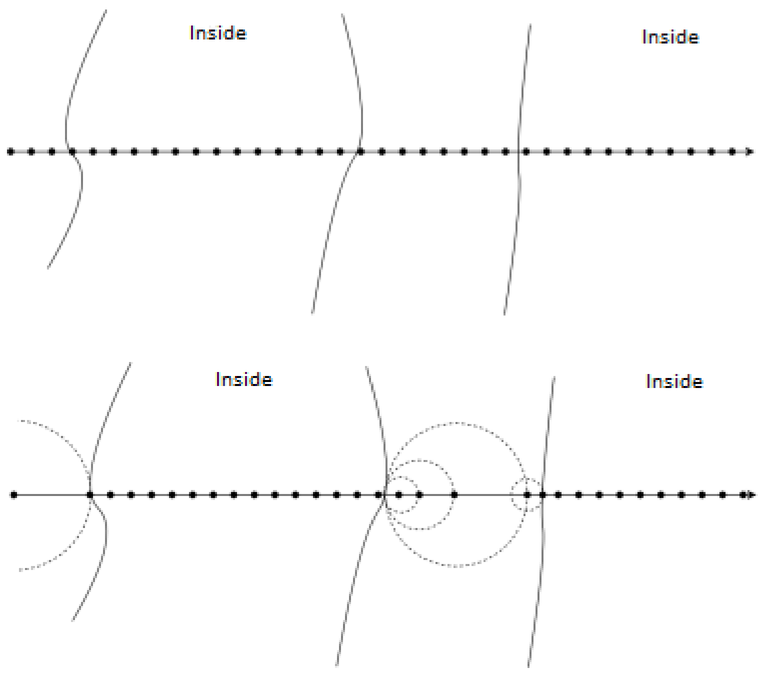


Figure 2.8: : Illustration of one cast ray, sampling a volume with both initial and intermediate empty space in its path. The ray is cast from left to right in the image. Top: Uniform step length across the ray. Ray steps that are outside of the working set are sampled just as densely as the steps inside. Bottom: Using the distance field to accelerate the rays outside the volume. Accelerated steps are marked with dashed radii.[56]

When compared to the traditional distance transform technique, the DT-ESS algorithm provides a number of benefits. First of all, it requires less computations since it bypasses empty areas, making it more efficient. Secondly, it is more accurate since it avoids misleadingly assigning a distance value to pixels in empty regions. Last but not least, it is more adaptable since it can be used with pictures of any size and shape and can manage complicated shapes and limits.

The Distance Transform Empty Space Skipping (DT-ESS) technique is a potent tool used in computer vision and image processing to quickly compute the distance transform of an image. The approach is quicker, more precise, and more adaptable than the traditional distance transform algorithm since it skips over blank areas in the picture during computation. Every image processing or computer vision professional should have access to DT-ESS due to its wide range of applications in fields including geology, robotics, and medical imaging.

2.5 Distance maps

The computational difficulty of tracking rays across the volume and figuring out how they interact with the volumetric data is one of the main difficulties in volumetric ray casting. Volumetric ray casting distance maps are one of the acceleration solutions that researchers have created to overcome this problem.

Distance maps are scalar fields that depict the separation between each point in space and the closest surface or volumetric data boundary. Several algorithms, including the signed distance function and the fast marching method, can be used to calculate them. Distance maps may be used in the context of volumetric ray casting to speed up the ray casting process by offering a rapid way to identify the closest surface or boundary to a certain point on the ray.

The following are the fundamental procedures for utilizing volumetric ray casting to create a distance map[52]:

1. Creating a mesh or sample grid that covers the amount of data to be displayed. The final image must have a degree of detail that can be captured by this grid or mesh.

2. casting rays using a technique like the signed distance function or the fast marching approach from each point on the grid or mesh to the closest surface in the scene. The signed distance function determines whether a location is within or outside of a surface and the distance to the closest surface. Distance maps may be computed quickly using the fast marching method, a numerical approach for solving partial differential equations.

3. Keeping track of the distance that each ray covers and store the information in a scalar field that corresponds to the distance map. Numerous data structures, including octree, and kd-tree, are capable of storing this scalar field.

4. An optional approach is to smooth or filter the distance map afterward to enhance its quality. Consequently, the distance map may experience less noise and artifacts.

5. Using the distance map to speed up ray casting during rendering. Find the closest surface for each ray by using a distance map query, then begin casting the beam from there. In addition to improving the final image's quality, this can dramatically lower the amount of calculations required to produce the picture.

Distance maps have the benefit of being computed offline and saved for later use, which can reduce rendering computation time. Distance maps can also be utilized to speed up tasks other than ray casting, such as collision detection or path planning.

The fundamental distance map approach may be modified in a number of ways to enhance its effectiveness and precision. The distance map might be stored, for instance, in hierarchical data structures like octree or kd-tree, which can minimize memory needs and speed up queries. By altering the distance map's sample density or level of detail depending on the complexity of the scene, adaptive sampling or level of detail may also be utilized to increase performance and accuracy.[24]

A potent method for speeding up the rendering of 3D volumetric data is the use of volumetric ray-casting distance maps. Distance maps may greatly reduce the computational complexity of ray tracing and enhance the quality of the final image by computing and storing a scalar field that indicates the distance to the closest surface beforehand.

2.5.1 Signed distance field

When representing shapes and objects as continuous scalar functions in computer graphics and computer vision, signed distance fields (SDFs)[34] are a potent tool. Contrary to conventional surface representations like polygon meshes or voxel grids, SDFs offer a fluid and effective method for representing complicated structures with precise surface and interior information.

A scalar function known as an SDF assigns a signed distance value—which represents the distance to the closest surface or object—to each point in space. It may be determined if a point is inside or outside of an item by the sign of the distance measurement. Points on an object's surface have a distance value of zero. So, an SDF offers a signed map of the object's spatial location.

Over alternative surface representations, SDFs provide a number of advantages. The first benefit is that SDFs offer a simple means of combining surface and interior data. The normal vector, curvature, and shape derivatives may all be obtained from SDFs by utilizing the distance values as a function of space. This continuous and differentiable information on the shape is made possible by the use of distance values. Due to the need for precise and effective shape information, SDFs are particularly advantageous for applications like collision detection, ray casting, and pathfinding.

Second, SDFs may express complicated forms using only a few parameters. Without the need for many discrete samples or polygons, SDFs may capture the object's underlying geometry and topology by representing the form as a function. For real-time applications where memory and processing limits are crucial, such games and simulations, this makes SDFs very helpful.

Last but not least, SDFs may be calculated and handled effectively utilizing a number of approaches, including signed distance functions, implicit surface representations, and level set methods. These methods enable efficient computing of operations such as union, intersection, and difference between SDFs as well as shape attributes such as distance, intersection, and blending.

For different forms and objects, SDFs may be computed using a variety of techniques. The SDF may be calculated analytically for basic geometric primitives like spheres, cylinders, and cubes by using their implicit equations. SDFs may be calculated for more complicated forms using sampling techniques like voxelization, meshing, or ray casting, and they can subsequently be transformed into a continuous scalar function using methods like the Euclidean distance transform or the Fast Marching Method.

SDFs are employed in applications involving medical imaging. SDFs can be utilized in medical imaging to accurately segment and analyze pictures by representing the surfaces of bodily organs and structures.

The segmentation of the liver and other organs in CT and MRI images is one frequent use of SDFs in medical imaging. The borders of the liver and other organs may be represented using SDFs, enabling precise and effective segmentation of these structures from the surrounding tissue.

In computer graphics and computer vision, signed distance fields are a potent tool that offers a fluid and effective approach to represent complicated shapes and objects as continuous scalar functions. Smooth interpolation, effective representation, and quick calculation are just a few benefits that SDFs have over conventional surface representations. They are widely used in collision detection, graphics, pathfinding, and other areas. SDFs are expected to be used more often in the future as new methods for calculating and manipulating them emerge.[36]

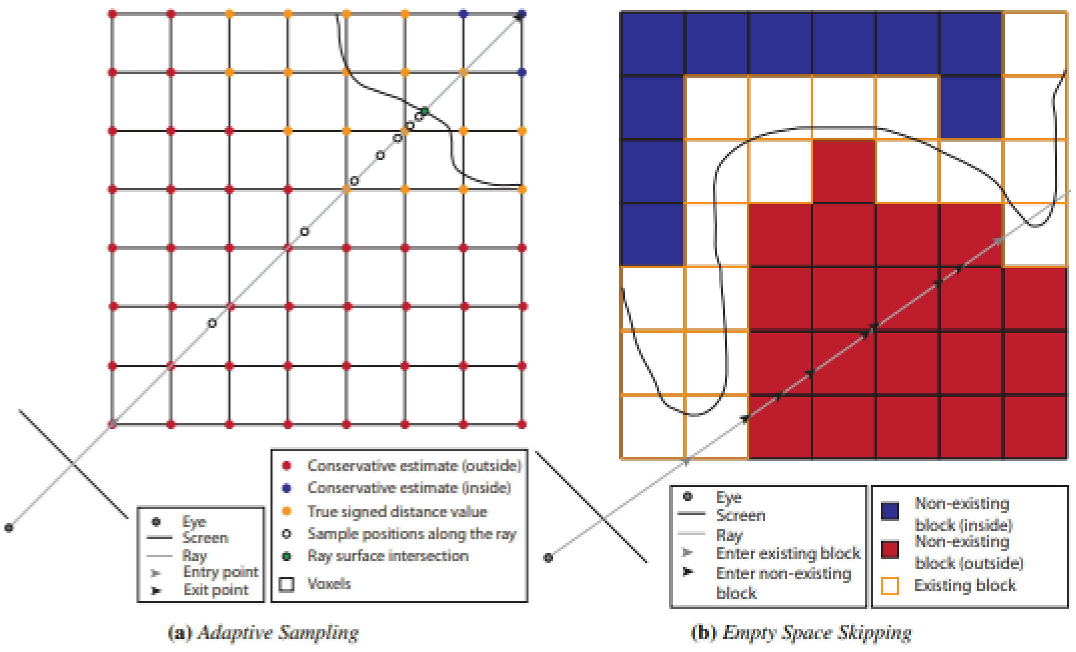


Figure 2.9: : Ray Casting [56]

Figure 2.9 illustrates how a signed distance field provides a natural way to do adaptive sampling, simply by following the sampled value to advance the sample position. As the sample position approaches the surface, the sampled value will automatically decrease in its magnitude because it is a signed distance to the surface. This greatly simplifies and optimizes the search algorithm. Moreover, searching for ray surface intersections is aggressively parallelizable.

If a ray does not hit a bounding box, then it is not processed further. Otherwise, the block origin can be computed from the intersection given the spatial regularity of the sparse data structure. The block origin can then be used to locate the small dense volume managed by the block in the sparse data structure. After that, we start from the entry point of the ray into the block and use trilinear interpolation to sample the small dense volume and to advance the sample position by the sampled value. If the sample value falls within a small threshold distance from the level set, the ray is considered to hit the surface and the sample position is taken as the intersection Sometimes, the signed distance field is not accurate and gives two adjacent sample values of different signs.

In this case, we use linear interpolation of the two sample positions to get an estimate of the ray surface intersection. When the intersection is found, we determine the surface normal for rendering by using finite differences to evaluate the gradient. While an implementation of the sparse data structure might provide a coherent interface for querying sample values, we explicitly avoid using such a feature because a ray can require several samples inside a block and each sample might require an expensive traversal of the tree structure. Working on the dense volume directly, however, gives us the ability to locate necessary values very quickly.

Most rays hit the surface soon after they start out from the initial intersections because the sparse data structure only stores blocks within a narrow band of the level set. There are rays, however, that do not hit the surface before entering a non-existing block. In this case, we use a ray voxel scanline traversal algorithm [7], as shown in Figure 2.9 b to quickly enumerate the blocks that the ray will traverse in increasing distance order. For each enumerated block, its existence is checked in the sparse data structure. If it exists, a ray bounding box intersection is computed and the search restarts, otherwise, we continue enumeration until the ray exits the volume.[71]

2.6 Chebyshev distance maps for volumetric ray casting acceleration

This section examines and the use of Chebyshev distance maps for volumetric ray casting acceleration.We point out both their benefits and drawbacks and draw linkages between the examined material and the strategy.

Visualizing volumetric images, which are composed of three-dimensional grids of scalar values called voxels, is a common practice known as volume rendering. Volume ray casting is a well-liked method for volume rendering that combines the results into a framebuffer by assigning each voxel a color and an opacity [Kruger and Westermann, 2003[39]].

Empty space skipping is one of the methods that have been devised to get over the computational difficulty of volume rendering. The goal of these methods is to locate opaque or transparent voxels that can be rendered without, hence enhancing performance. This objective has been attained through a variety of strategies, such as Early Ray Termination (ERT) and Empty Space Skipping (ESS).

Within this Context, Chebyshev distance maps are used for effective empty space skipping as one of the major improvements in volume rendering techniques. The Chebyshev distance between each voxel and the closest occupied region within the volume is shown on Chebyshev distance maps. With the use of this distance map, rendering can be sped up by effectively skipping transparent or empty areas during volume ray casting.

Due to the fact that it takes into consideration the greatest amount of absolute differences between related voxel positions across all dimensions, the Chebyshev distance metric is especially well-suited for this use. No matter where a voxel is located in three-dimensional space, the algorithm can detect its proximity to the nearest inhabited region by calculating the Chebyshev distance.

Empty space skipping approaches can use Chebyshev distance maps to determine if a specific ray or voxel crosses with an inhabited zone or can be safely skipped. Utilizing the data offered by the distance maps allows for the reduction of pointless computations and texture samples, leading to appreciable performance gains.

Chebyshev distance maps provide several benefits for volume rendering empty space skipping. They first accurately and dependably depict how close each voxel is to the closest occupied area. This enables more accurate identification of transparent or empty sections, resulting in more effective skipping and lessening of the overall computing load.

Additionally, Chebyshev distance maps can be efficiently created and are rather easy to compute. The method of creating a distance map is moving across the volume and figuring out the Chebyshev distance for each voxel based on where it is in relation to the closest occupied area. Then, for reference during the rendering process, this data is saved in a different map or data structure.

However, it is crucial to take into account the restrictions and difficulties related to employing Chebyshev distance maps for ESS. The volume's resolution and the sample density applied to create the maps have an impact on how well the distance computations are performed. The distance maps may need to be updated often in some situations, especially when dealing with volumes that are very complicated or change dynamically, to ensure accurate skipping decisions[17].

Despite these difficulties, using Chebyshev distance maps has demonstrated a great deal of promise in terms of enhancing the effectiveness and performance of empty space skipping in volume rendering [17]. By utilizing this method, rendering algorithms may decide which regions to skip with greater knowledge, resulting in quicker rendering times without sacrificing visual quality.

2.7 Conclusion

the exploration of volumetric ray casting techniques has provided valuable insights into the advancements and optimizations in volumetric rendering. We have examined various aspects, including sampling methods, empty space skipping, acceleration structures, distance maps, and the use of Chebyshev distance maps for acceleration.

Sampling plays a critical role in discretizing volumetric data, and we have explored different approaches such as irregular sampling, adaptive sampling, and hierarchical sampling. These methods enable more efficient representation of volumetric data and improve rendering performance.

Empty space skipping techniques, including KD-trees and octrees, have been examined for their ability to accelerate ray casting by efficiently skipping empty regions. These acceleration structures optimize the rendering process, resulting in faster and more efficient visualization.

Distance maps, particularly signed distance fields, provide valuable information about the distance to the nearest surface and enhance surface extraction and ray casting. Their integration improves the accuracy and realism of volumetric rendering.

Chapter 3

Method

3.1 Introduction

In the real of volume rendering, achieving efficient rendering performance is of paramount importance. To address this challenge, this chapter presents to enhance the speed of volume rendering by effectively skipping empty blocks and leveraging Chebyshev distance maps[17].

By intelligently sampling the occupancy map and identifying empty or occupied blocks, unnecessary computations within empty regions can be avoided, resulting in significant performance improvements.

To overcome the limitations of the simple Empty Space Skipping approach, the Chebyshev Distance Empty Space Skipping method is introduced. This technique utilizes the concept of Chebyshev distance, which measures the maximum difference between two points along any dimension. By calculating and storing Chebyshev distances in a dedicated distance map derived from the occupancy map, multiple blocks can be skipped with a single texture sample leading to accelerated rendering speed.

3.2 Goal and motivation

The goal of this chapter is to present an approach[17] enhancing the performance and efficiency of volume rendering techniques through the introduction of advanced optimization methods. The motivation behind this work stems from the ever-increasing demand for real-time and interactive rendering of volumetric data in various domains, such as medical imaging, scientific visualization, and computer graphics.

Traditional volume rendering techniques can be computationally expensive, especially when dealing with large datasets and complex scenes. This often results in significant rendering times, limiting the interactivity and responsiveness of the visualization process.

The main objective of this project is to develop a technique for effectively skipping empty blocks within the occupancy map during volume ray casting. By identifying and avoiding unnecessary computations within empty regions, rendering performance can be significantly improved. Also, we use Chebyshev distance maps to enable the skipping of multiple blocks with a single texture sample. This approach aims to reduce the number of texture samples required, thereby optimizing memory usage and reducing computational overhead.

3.3 System Overview

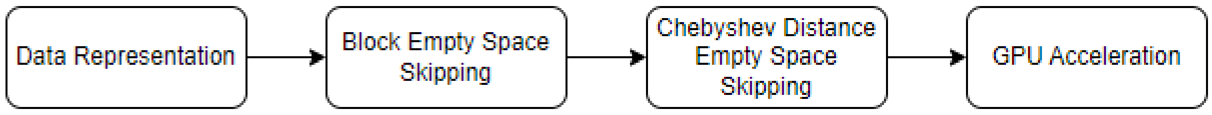

Data Representation
Block Empty Space Skipping
Chebyshev Distance Empty Space Skipping
GPU Acceleration

Figure 3.1: General Scheme of current method

Our overall plan's main elements are as follows:

Data Representation A three-dimensional depiction of a physical item or phenomenon (as a set of 2D images) is referred to as volume data. It is made up of a collection of data (such as intensity or color) that are described in a regular grid layout and are often derived from simulations or medical imaging devices.

Each voxel in the three-dimensional grid or lattice used to represent the volumetric data provides details about the characteristics or attributes of the relevant volume element.

The volumetric data is used to create the occupancy map, which indicates whether a block is empty or not. This map acts as a manual for effective sampling during rendering.

Block Empty Space Skipping Block Empty Space Skipping is the first optimization method used. This method entails splitting the volume into blocks and identifying each block's status as empty or not empty using the occupancy map.

Empty blocks are skipped during the volume ray casting process, minimizing pointless computations and enhancing rendering speed.

The blocks' empty or non-empty status is determined using the normalized texel coordinates of the occupancy map, enabling effective skipping of vacant regions.

Chebyshev Distance Empty Space Skipping We present the Chebyshev Distance Empty-Space-Skipping method to further improve the skipping efficiency.

Chebyshev distance maps allow us to calculate the maximum number of blocks along each dimension that can be skipped before coming to an occupied block.

This method minimizes the number of samples needed and optimizes memory use by allowing numerous blocks to be skipped with a single texture sample.

GPU Acceleration Our overall plan makes use of the contemporary programmable GPUs' processing capacity to quicken the rendering process.

To effectively analyze and depict the volumetric data, GPU-accelerated algorithms and methods are used.

3.4 Block Empty Space Skipping

3.4.1 Mapping between Normalized and Unnormalized Texel Coordinates

The goal of the "Mapping between Normalized and Unnormalized Texel Coordinates" step is to establish a relationship between the normalized and unnormalized texel coordinates used in volume rendering. First, we explain the difference between the two kinds of texel coordinates:

Normalized Texel Coordinates: Positions within the texture's three-dimensional space are indicated by values between 0 and 1 known as normalized texel coordinates. They are normalized in proportion to the texture's dimensions, where the minimum coordinate along each axis is (0, 0, 0) and the maximum coordinate along each axis is (1, 1, 1). These coordinates are frequently utilized in interpolation to determine a texel's color or value based on its surrounding texels. No matter the resolution or size of the texture 3D, normalized texel coordinates offer a consistent and standardized manner to access texels.

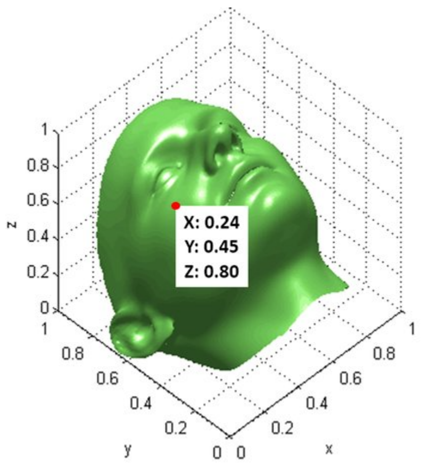


Figure 3.2: The Normalized Coordinate [72]

Unnormalized Texel Coordinates: Integer values that exactly match the texel indices within the texture 3D are known as unnormalized texel coordinates. They are created by multiplying the normalized texel coordinates by the texture's dimensions. Unnormalized coordinates offer a direct mapping to the texture 3D's texels, making direct texel indexing and quick memory access possible. When conducting procedures that involve specific texels or getting voxel information from the volume data, or in other circumstances where exact access to individual texels is necessary, they are especially helpful.

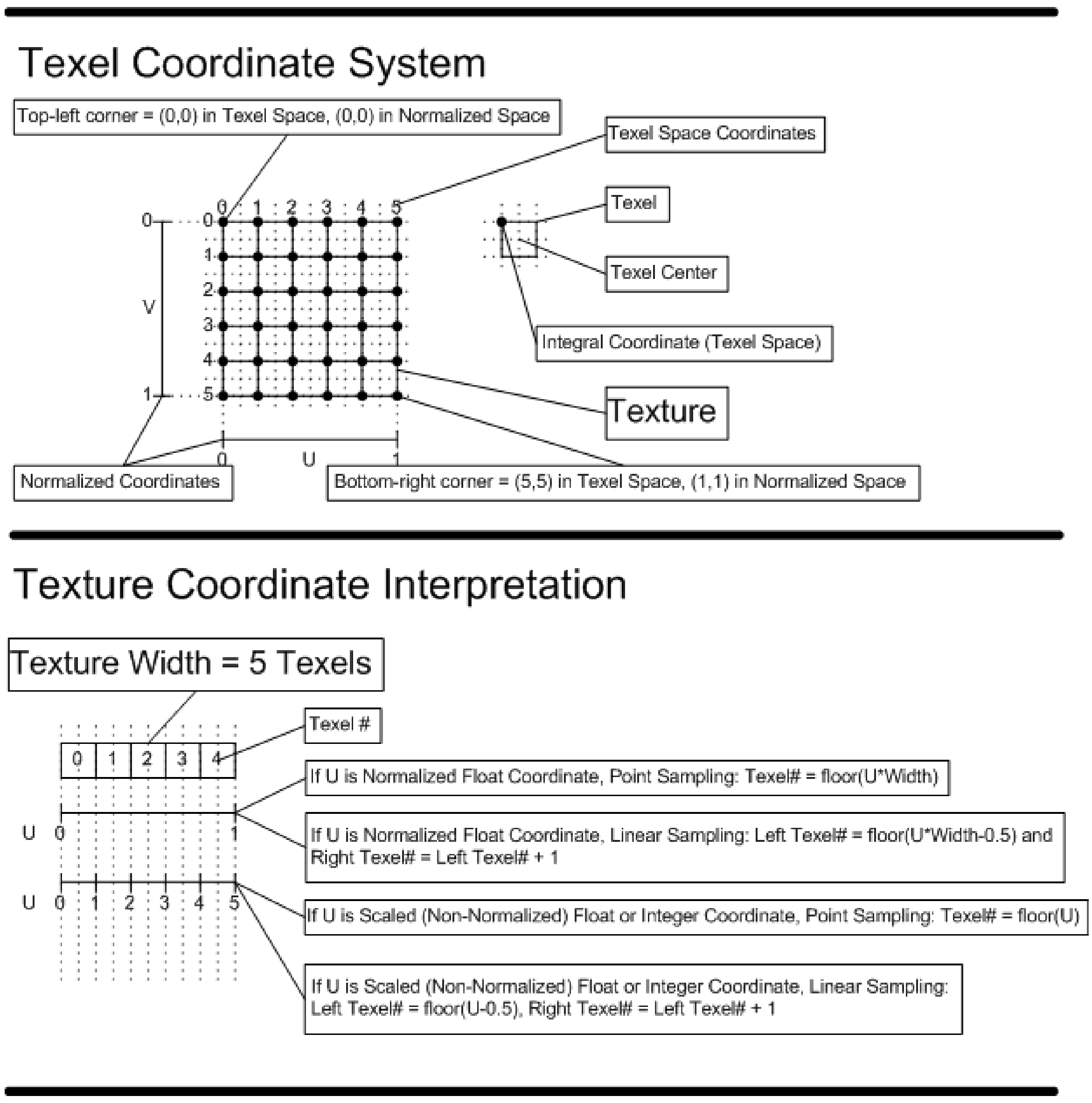

Texel Coordinate System
Top-left corner = (0,0) in Texel Space, (0,0) in Normalized Space
Texel Space Coordinates
Texel
Texel Center
Integral Coordinate (Texel Space)
Texture
Normalized Coordinates
Bottom-right corner = (5,5) in Texel Space, (1,1) in Normalized Space
V
U
Texture Coordinate Interpretation
Texture Width = 5 Texels
Texel #
If U is Normalized Float Coordinate, Point Sampling: Texel# = floor(U*Width)
If U is Normalized Float Coordinate, Linear Sampling: Left Texel# = floor(U*Width-0.5) and Right Texel# = Left Texel# + 1
If U is Scaled (Non-Normalized) Float or Integer Coordinate, Point Sampling: Texel# = floor(U)
If U is Scaled (Non-Normalized) Float or Integer Coordinate, Linear Sampling: Left Texel# = floor(U-0.5), Right Texel# = Left Texel# + 1

Figure 3.3: Texture Mapping[54]

The mapping step serves several important purposes:

Accurate Sampling The mapping ensures that texels within the occupancy map are sampled accurately based on their corresponding positions in the original volumetric image. By establishing a clear relationship between the normalized and unnormalized texel coordinates, the section enables precise sampling of texels, which is crucial for generating an accurate representation of the volume data.

Interpolation Interpolation is a key technique used in volume rendering to estimate values between texels and create smooth transitions. The mapping between normalized and unnormalized texel coordinates facilitates proper interpolation between texels, ensuring that the rendered volume appears visually coherent and realistic.

Consistent Rendering The mapping helps maintain consistency in the rendering process by providing a standardized coordinate system. This allows for consistent access and interpretation of texels within the occupancy map, regardless of the original volumetric image's dimensions or the specific rendering algorithm used. Consistency in rendering is essential for producing reliable and predictable results.

Memory Utilization The mapping between normalized and unnormalized texel coordinates also contributes to efficient memory utilization. By using a normalized coordinate system, the occupancy map can represent the empty or non-empty status of larger blocks within the volume using significantly less memory compared to the original volumetric image. This reduction in memory requirements is particularly beneficial when using larger block sizes, resulting in more efficient storage and processing of the volume data.

Texture 3D Three-dimensional textures, which are a type of data structure for storing and accessing volumetric data, are referred to as ”texture 3D.” The occupancy map is displayed as a 3D texture throughout this section. It is a three-dimensional grid of texels, where each texel corresponds to a voxel in the volume data and provides details on its occupancy state (empty or not empty).

The action of transferring 3D volume data to a 3D texture. It entails converting the volumetric image into a format appropriate for fast rendering and storage, frequently utilizing a method like block-based representation or occupancy maps.

The terms ”normalized” and ”unnormalized” texel coordinates in the context of volume rendering relate to various coordinate systems used to access texels within a 3D texture (also known as a texture 3D) representing the volume data.

To make sure that volume rendering is accurate and consistent, a mapping between normalized and unnormalized texel coordinates is developed. The fast sampling, interpolation, and display of the volume data are made possible by the smooth transitions between the two coordinate systems. Depending on the unique needs of the rendering algorithm and the actions being taken with the texture 3D, one coordinate system may be chosen over another.

The given equation establishes a relationship between normalized texel coordinates (t) and unnormalized texel coordinates (u) in a 3D texture. Here are the key details:

Dimensions: The dimensions of the 3D texture are represented by [width, height, depth], denoted as d = [width, height, depth].

Normalized Texel Coordinates (t): These coordinates have values ranging from 0.0 to 1.0 for each dimension.

Unnormalized Texel Coordinates (u): These coordinates have values ranging from 0.0 to the corresponding dimension value (width, height, or depth).

Mapping Equation: The equation

$$u = d \cdot t \tag{12}$$

represents the mapping between the two coordinate systems.

To convert a normalized texel coordinate (t) to an unnormalized texel coordinate (u) using this equation, you need to multiply each component of t by the corresponding dimension of d. This multiplication results in the corresponding unnormalized texel coordinate, u.

3.4.2 Calculating the Number of Steps Along a Ray in Volume Sampling

Calculating the number of steps along a ray in volume sampling:

The equation :

$$n = \ulcorner \overrightarrow{\mathrm{max}}(d) \cdot \|t_{\text{back}} - t_{\text{front}}\| \cdot f \urcorner \tag{13}$$

allows us to determine the number of steps (n) required for sampling voxel colors and transparencies along a ray intersecting a volume. Here's a breakdown of the variables and terms involved:

n: This represents the number of steps along the ray, determining the sampling density or resolution.

max(d): It corresponds to the maximum dimension (width, height, or depth) of the volume, indicating the largest value among its dimensions.

$\|tback - tfront\|$: This denotes the Euclidean distance between the t-back and t-front points along the ray, representing the length of the ray segment within the volume.

f : It is a scaling factor or parameter used to adjust the sampling density, allowing control over the number of samples taken along the ray.

By combining these elements, the equation calculates the necessary number of steps (n) to sample voxel properties along the intersecting ray within the volume. To compute n using the equation:

Determine the maximum dimension (max(d)) of the volume.
Calculate the Euclidean distance $\|t_{\text{back}} - t_{\text{front}}\|$ between the t_{back} and t_{front} points along the ray.
Multiply max(d), $\|t_{\text{back}} - t_{\text{front}}\|$, and f.
Round down or take the floor of the result to obtain the integer value of n.

It's important to note that the equation assumes equidistant sampling, where the ray is divided into n equal segments to sample voxel properties. The value of n influences the level of detail or accuracy in the sampled information along the ray.

3.4.3 Calculating the Change in Normalized Texture Coordinates (Δt)

In the given equation

$$\Delta \mathrm{t} = \frac{\mathbf{t}_{\text{back}} - \mathbf{t}_{\text{front}}}{n - 1} \tag{14}$$

Δt represents the change in normalized texture coordinates between each sample point along the ray. Here's what we need to know:

Δt: It signifies the step size or increment in the normalized texture coordinates between consecutive sample points.

t-back: This refers to the normalized texture coordinate corresponding to the back intersection point of the ray with the volume.

t-front: This represents the normalized texture coordinate corresponding to the front intersection point of the ray with the volume.

n: It denotes the number of steps or samples along the ray, determining the sampling density or resolution.

To calculate Δt using the equation (14)

```
$t_{\text{Front}} \leftarrow$ value of tFront
$t_{\text{Back}} \leftarrow$ value of tBack
$n \leftarrow$ value of n
$difference \leftarrow t_{\text{Back}} - t_{\text{Front}}$
$\Delta t \leftarrow \frac{difference}{n-1}$
**return** $\Delta t$
```

By using this Δt, you can determine the increment in normalized texture coordinates required to move from one sample point to the next along the ray.

3.4.4 Calculating Normalized Texel Coordinates along a Ray

The equation

$$\mathbf{t}_i = \mathbf{t}_{\text{front}} + i \cdot \Delta\mathbf{t} \tag{15}$$

you can use it to calculate the normalized texel coordinates (ti) at the ith point along the ray within the volume. Here's what you need to know:

ti: It represents the normalized texel coordinates at the ith point along the ray, indicating the texel's position in normalized texture coordinates.

t-front: This corresponds to the normalized texture coordinate at the front intersection point of the ray with the volume.

Δt: It signifies the change in normalized texture coordinates between each sample point along the ray, representing the step size or increment.

i: It is the index or number of the point along the ray, determining the position of the texel along the ray.

By following these steps, you can determine the normalized texel coordinates at each point along the ray within the volume.

3.4.5 Calculating Normalised texel coordinates of the occupancy map

$$\mathbf{t}_M = \frac{\mathbf{d}}{B \cdot \mathbf{d}_M}\mathbf{t} \tag{16}$$

allows you to calculate the normalized texel coordinates (tM) of the occupancy map:

tM: It represents the normalized texel coordinates of the occupancy map, indicating the position of the occupancy map texel in normalized coordinates.

d: It denotes the dimensions of the original volumetric image, given as [width, height, depth].

B: This is the block size, typically 4 or larger.

dM: It signifies the dimensions of the occupancy map, given as $\lfloor \mathbf{d}/B \rfloor$, (floor division), representing the number of blocks in each dimension of the volume.

To calculate tM using the equation, we divide each dimension of d by (B · dM) to obtain the scaling factor for each dimension. Multiply the scaling factor by the original normalized texel coordinates (t) to obtain tM. We can determine the normalized texel coordinates of the occupancy map. The occupancy map provides a memory-efficient representation of the volume, storing information about the empty or non-empty status of blocks within the volume. This optimizes memory usage and rendering performance by reducing the required memory compared to storing the entire volumetric image.

3.4.6 Change in Unnormalized Texel Coordinates and Ray Steps in the Occupancy Map

The equation

$$\Delta \mathbf{u}_M = \mathbf{d}_M \cdot \mathbf{t}_M = \frac{\mathbf{d}}{B} \Delta \mathbf{t} \tag{17}$$

describes the change in unnormalized texel coordinates of the occupancy map per step. This equation is derived from the relationship between the unnormalized texel coordinates ($\mathbf{u}_M$), the voxel size ($\mathbf{d}_M$), and the normalized texel coordinates ($\mathbf{t}_M$). The voxel size is scaled by the ratio of the total volume size ($\mathbf{d}$) to the number of blocks (B), resulting in $\frac{\mathbf{d}}{B}$. The change in normalized texel coordinates per step ($\Delta \mathbf{t}$) is multiplied by this scaled voxel size to obtain the change in unnormalized texel coordinates ($\Delta \mathbf{u}_M$).

This relationship follows from the equation $u = d \cdot t$, which relates the unnormalized texel coordinate (u) to the total distance (d) and the normalized texel coordinate (t). Additionally, the equation $\mathbf{t}_M = \frac{\mathbf{d}}{B \cdot \mathbf{d}_M} \mathbf{t}$ defines the normalized texel coordinates ($\mathbf{t}_M$) in terms of the voxel size and the total distance.

The number of ray steps required to move from one voxel to the next along a specific dimension (j) can be derived geometrically based on these equations. This calculation allows for accurate traversal of the occupancy map in the volume rendering process, ensuring proper sampling and rendering of the volume data.

3.5 Chebyshev Distance Empty Space Skipping

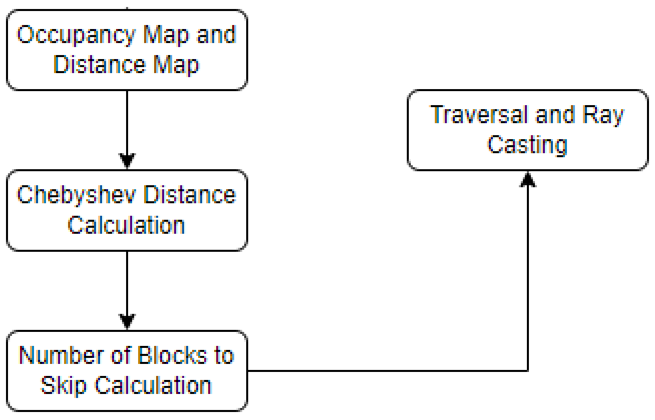

Occupancy Map and Distance Map
Chebyshev Distance Calculation
Number of Blocks to Skip Calculation
Traversal and Ray Casting

Figure 3.4: Empty Space Skipping by Chebyshev Distance Map

On GPUs, bypassing empty blocks in the volume can increase the efficiency of volume ray casting. One block could only be skipped per sample of the occupancy map using the straightforward empty space skipping (ESS) method that was covered in the preceding section. In this section, we present the Chebyshev distance ESS algorithm that tackles the performance drawback of the straightforward ESS strategy.

The largest difference between two places along any dimension is known as the Chebyshev distance. We may calculate the number of blocks that can be skipped along each dimension before arriving at an occupied block by calculating the Chebyshev distance between a block and the closest occupied block. Using a GPU-accelerated modification of Saito and Toriwaki's method, we compute and store these distances in a distance map that is derived from the occupancy map. A block that is occupied has a distance of 0.

To calculate the number of steps required to move to a block D blocks away on dimension i, we use the equation:

$$\Delta i_j = \begin{cases} \left(1 + \lfloor \mathbf{u}_{M_j} \rfloor - \mathbf{u}_{M_j}\right) \Delta \mathbf{u}_{M_j}^{-1} & \text{if } \Delta \mathbf{u}_{M_j} > 0 \\ \left(\lfloor \mathbf{u}_{M_j} \rfloor - \mathbf{u}_{M_j}\right) \Delta \mathbf{u}_{M_j}^{-1} & \text{otherwise.} \end{cases} \tag{21}$$

We then obtain the actual number of steps using the equation:

$$\Delta i = \max\left(\overrightarrow{\min} \ulcorner (step(-\Delta uM) + \operatorname{sign}(\Delta uM) \cdot D + rM) \cdot \Delta uM^{-1} \urcorner, 1\right) \tag{22}$$

By replacing the previous equation with this new equation, the raycaster can skip multiple blocks with just a single sample of the distance map. Notably, the utilization of Chebyshev distances for GPU-accelerated ray casting builds upon the work of Sramek and Kaufman, offering improved efficiency and the ability to resume traversal in empty space within and behind objects. The distance map is maintained at a lower resolution than the volume, resulting in performance enhancements and reduced memory overhead.

When using volume ray casting on GPUs, the Chebyshev distance ESS technique offers an improved method for empty space skipping. Chebyshev distances and a distance map created from the occupancy map allow for the effective skipping of numerous blocks with a single texture sample. This method boosts efficiency, uses less memory, and enables traversal to continue in void areas. The Chebyshev distance ESS algorithm provides a more effective and efficient method for volume ray casting on contemporary programmable GPUs by overcoming the drawbacks of the standard ESS methodology.

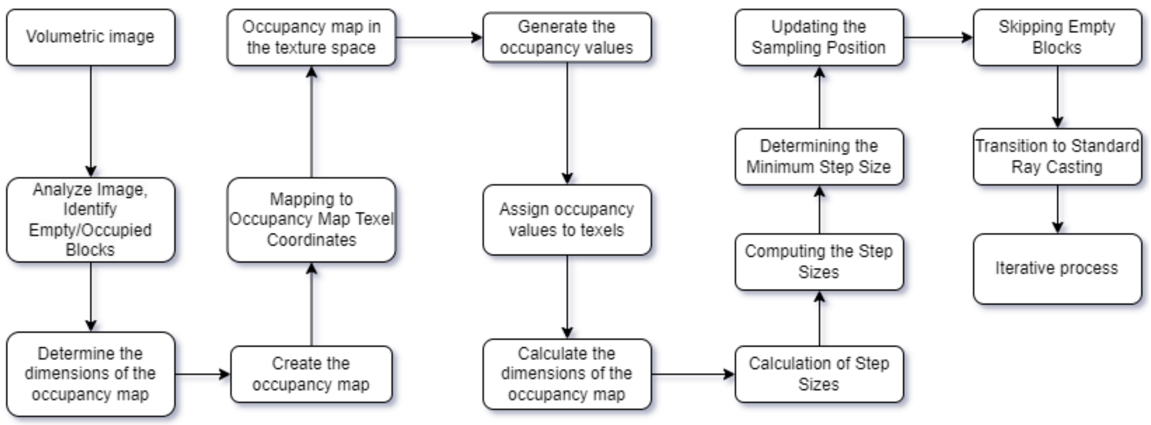

Volumetric image
Analyze Image, Identify Empty/Occupied Blocks
Determine the dimensions of the occupancy map
Create the occupancy map
Mapping to Occupancy Map Texel Coordinates
Occupancy map in the texture space
Generate the occupancy values
Assign occupancy values to texels
Calculate the dimensions of the occupancy map
Calculation of Step Sizes
Computing the Step Sizes
Determining the Minimum Step Size
Updating the Sampling Position
Skipping Empty Blocks
Transition to Standard Ray Casting
Iterative process

Figure 3.5: Chebyshev Distance Empty Space Skipping (CDESS) Algorithm: Optimizing Volumetric Rendering

Comparing the Chebyshev Distance Empty Space Skipping (CDESS) approach to the Block Empty Space Skipping (BESS) method, this methodology aims to optimize the rendering performance of volumetric data. The occupancy map, a downsampled representation of the volumetric data, is the first thing the algorithm determines. These dimensions are calculated by dividing the volumetric picture dimensions by the block size (B), which results in the formula

$$\mathbf{d}_M = \left\lfloor \frac{\mathbf{d}}{B} \right\rfloor \tag{23}$$

where d stands for the volumetric image dimensions. A new 3D texture is made specifically to store the occupancy map after taking into account its dimensions, ensuring effective storage and retrieval of occupancy data.

The method then generates occupancy values for each texel on the map in order to fill it with useful data. In this generation process, the occupancy status of each block is determined by examining the voxel values or other pertinent volumetric data properties. Blocks that are vacant are given a specific value that shows their status as vacant, and blocks that are occupied are given a value that indicates their occupancy.

After being determined, the occupancy values are applied to the appropriate texels in the occupancy map texture. This procedure makes sure that the occupancy data is precisely saved in the texture so that it may be quickly and effectively accessed throughout the rendering process.

The approach transfers the normalized texel coordinates (t) used for sampling the volumetric picture to the equivalent texel coordinates (tM) in the occupancy map in order to efficiently skip vacant blocks. The formula 16, Is used to create this mapping. The algorithm guarantees that the sampling points line up with the proper texels in the occupancy map by using this mapping, which makes traversing and bypassing empty blocks easier.

The next step is to determine the step sizes (i) for sampling along each of the occupancy map's dimensions (X, Y, Z). These step sizes are established using a predetermined sampling factor and the Chebyshev distance metric. The direction in which the occupancy map will be traversed is then determined by finding the minimum step size ($min(iX, iY, iZ)$) among the three dimensions.

The algorithm iterates across the occupancy map, incrementing the sampling position based on the step size and direction once the sample point and step sizes have been determined. As the algorithm develops, this procedure enables empty blocks to be skipped. The algorithm switches to the conventional volume ray casting procedure when an occupied block is found, sampling the volumetric picture and computing the output voxel's final color and transparency values using methods like trilinear interpolation.

Until the full volume is drawn, this iterative process of bypassing empty blocks and executing volume ray casting continues. Due to the effective use of the Chebyshev distance measure, the occupancy map and volumetric picture are traversed according to the computed step sizes, bypassing numerous blocks in each sample. By lessening the computing effort needed for rendering volumetric data, this improvement dramatically increases rendering performance.

As shown by the equation 16 and the use of the Chebyshev distance metric, the integration of Chebyshev distance empty space skipping into the CDESS algorithm allows for the skipping of multiple blocks per sample, which leads to faster rendering times and more effective use of computing resources when compared to the BESS technique.

3.6 Conclusion

In this chapter, we presented a method for efficient volume ray casting, focusing on block empty space skipping and the utilization of Chebyshev distance empty space skipping. Our goal was to improve the rendering performance of volumetric data visualization by optimizing the ray casting process and reducing the number of unnecessary computations, we aimed to enhance the overall rendering speed and enable real-time exploration of complex volumetric data. By considering the Chebyshev distance between blocks, it becomes possible to skip multiple blocks in a single sample, further improving the efficiency of volume ray casting.

Chapter 4

Implementation and results

4.1 Introduction

The implementation of a software tool for volumetric picture rendering using raycasting and distance mapping methods is the focus of this project. The main objective is to develop an interactive visualization tool that allows users to explore and understand volumetric data in a simple and effective manner. To achieve this goal, the project utilizes the features of the PyQt5 and VTK libraries, implementing the solution in Python.

In this project, dataset selection is a critical stage, as it forms the basis for rendering and visualization. The chosen dataset should be representative of the target domain and should include discrete anatomical or structural features with a range of intensity values. Factors such as complexity, data size, availability of ground truth or reference data, and compatibility with the VTK library and DICOM format are taken into account during the dataset selection process.

4.2 Implementation Overview

The project implementation comprised the creation of a software tool for volumetric picture rendering using raycasting and distance mapping methods. The main goal is to develop an interactive visualization tool that enables users to explore and understand volumetric data simply and effectively.

Utilizing the features of the PyQt5 and VTK libraries, the implementation was carried out using the Python programming language. The graphical user interface (GUI) was created with ease thanks to PyQt5, which offered a solid platform for GUI development. The main library used to manage volumetric data and run computations for raycasting and distance mapping was called VTK (Visualization Toolkit).

The implemented code adhered to a modular and object-oriented design to guarantee maintainability and scalability. User interface elements, data loading and preprocessing modules, rendering components, and interaction handlers were some of its essential components. These elements combined to provide a complete and simple user experience.

The capabilities of VTK's volumetric rendering were heavily utilized. For effective raycasting rendering, the solution used the vtkGPUVolumeRayCastMapper class. To produce high-quality renderings, this class used a variety of sampling techniques, including trilinear interpolation and adaptive sampling. These methods guaranteed precise volumetric data representation and seamless changes in intensity values.

Distance maps were essential in the implementation for accelerating the raycasting process. Both Euclidean and Chebyshev distance maps were used, and Chebyshev distance maps were implemented using the methods covered in the approach chapter. The Euclidean distance maps were created by determining the distance between each voxel in the volume and a reference point using the vtkImageEuclideanDistance filter. The rendering performance was enhanced by the effective skipping of voids during raycasting made possible by these distance maps.

The depiction of volumetric data was made possible by the implementation's combined use of distance maps and the raycasting technique. The databases allowed for interactive exploration and comprehension of complex structures. The combination of PyQt5 and VTK libraries offered a strong platform for the smooth integration of rendering activities with GUI elements, improving the overall user experience.

We will go into more detail about the implementation in the following sections, covering topics like dataset selection, GUI design, data loading and preprocessing, camera setup, and visualization techniques used. We will assess the implementation's success and the results obtained after a thorough examination of the findings.

Given the updates you provided regarding the application of Euclidean and Chebyshev distance maps as well as the implementation of Chebyshev distance maps based on the techniques covered in the approach chapter, we will highlight the use of various distance map techniques to speed up rendering with raycasting in the relevant sections.

4.3 Dataset Selection

A critical stage in carrying out the project is choosing a suitable dataset. The chosen dataset should sufficiently represent the target domain since it will act as the basis for rendering and visualization.

The dataset selection procedure for this project includes locating and gathering volumetric data that is pertinent to the application domain. Datasets with comprehensive anatomical or scientific information, such as those from medical CT or MRI scans, scientific simulations, or industrial scans, were the main focus.

The complexity and amount of the data, the availability of ground truth or reference data for validation, and the compatibility with the VTK library and DICOM format were all taken into account when choosing the dataset. A popular standard for storing and transferring medical pictures, DICOM (Digital Imaging and Communications in Medicine) is a good option for smooth integration with VTK.

The data collection method required getting the dataset in DICOM format after a suitable dataset had been found. This often involved gaining access to a database or obtaining information from scientific or medical organizations that make anonymized datasets available for research.

The chosen dataset should include discrete anatomical or structural features and a range of intensity values. This guarantees that the fundamental information included in the volumetric data can be successfully captured and represented by the visualization techniques, such as ray casting and distance mapping.

The technical requirements and relevance to the project's goals were taken into account during the dataset selection process. The implementation can successfully demonstrate the capabilities of the built visualization tool by selecting an acceptable dataset.

The particular processes necessary to load and preprocess the chosen dataset will be covered in detail in the sections that follow, ensuring that it is prepared for rendering and visualization. In the later stages of implementation, the dataset selection is crucial since it allows the application to demonstrate its functionality on meaningful and representative data.

This section emphasizes the significance of choosing a suitable dataset and establishes the framework for the upcoming phases in data loading and preprocessing by giving a more thorough explanation of the dataset selection procedure.

4.4 Experimental Setup

Evaluation of the performance and efficacy of the adopted strategy heavily relies on the experimental arrangement. It entails specifying the hardware and software setups utilized for the visualization tool's testing and validation.

4.4.1 Hardware Configuration

The computational resources used to operate the implemented software application are included in the hardware configuration. It contains information about the computer system's specifications, including the processor, memory (RAM), and graphics card. The speed and responsiveness of the visualization tool are directly influenced by the computing power of the hardware.

A powerful computer system was used for this project to enable the effective and efficient implementation of the adopted strategy. The system was made up of a multicore CPU with enough memory to fulfill volumetric data processing and storage needs. To achieve real-time interactive visualization, a potent graphics card that can support complex rendering techniques was also used.

4.4.2 Software Configuration

An integrated development environment (IDE) that enables Python development made up the development environment. To create, test, and debug the code, tools like PyCharm, Visual Studio Code, or Jupyter Notebook were employed.

The implementation largely depended on the PyQt5 library to provide the graphical user interface (GUI) in terms of libraries and frameworks. A complete collection of tools and widgets are available in PyQt5 for creating interactive desktop apps. For processing the volumetric data, computing ray casting, and distance maps, and allowing enhanced visualization features, the VTK library was used.

4.4.3 Experimental Data

A set of experimental data was chosen in order to assess the method that was put into practice. This information included volumetric datasets from pertinent fields including industrial scanning, scientific simulations, and medical imaging. To fully assess the capabilities of the visualization tool, the data should reflect a variety of structures, intensities, and complexity.

Medical CT or MRI scans that have been anonymized, volumetric datasets produced by scientific simulations, or actual volumetric scans from industrial operations may all be utilized as the experimental data for the assessment phase. The primary data for the experimental setup is the dataset chosen in the "Dataset Selection" stage (Section 4.3).

The experimental setup provides a defined and controlled environment for assessing the performance and capabilities of the implemented visualization tool by specifying the hardware and software specifications and choosing suitable experimental data. The outcomes achieved throughout the assessment phase may be compared and analyzed on the basis of this information.

The procedures specifically followed to apply the implemented technique to the experimental data will be covered in detail in the parts that follow, along with the outcomes. The experimental design lays the groundwork for the analysis and discussion that follow the results.

4.5 Data and Application Interface

The implemented code was run on the chosen DICOM dataset, and its effectiveness was evaluated using the specified evaluation criteria.

We have used three different datasets: D1(Head), D2(Ankle), and D3 (Chest). The following table gives each dataset its resolution, and the number of 2d images that contain the number of voxels obtained after 3D reconstruction.

Table 4.1: Dataset Information

Dataset	Resolution (X, Y)	Number of slices 2D	Voxels Number
D1	(512, 512)	234	61,341,696
D2	(512, 512)	150	39,321,600
D3	(512, 512)	197	51,642,368

You can load the DICOM images series by clicking the buttons "Ray casting rendering".

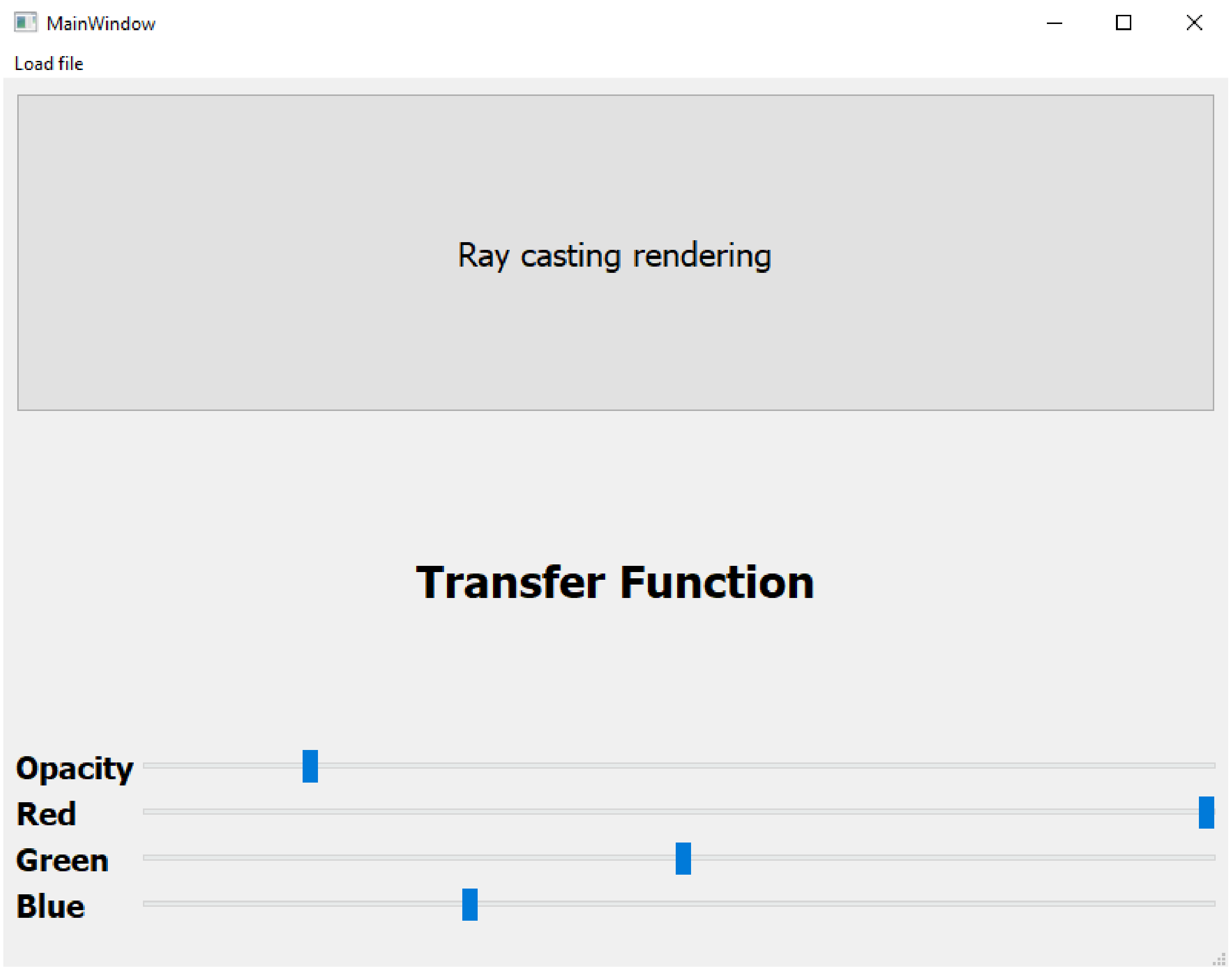


Figure 4.1: GUI

4.6 Experimental Results

The outcomes illustrated the code's capacity to produce interactive volumetric visualizations. It was possible to visualize and analyze the produced volumes in great detail since they provided accurate and slick representations of the underlying data.

4.6.1 Basic raycasting Results

We start with presenting results obtained by applying raycasting without using distance maps. Different transfer functions are applied by adjusting: Opacity, color, or both.

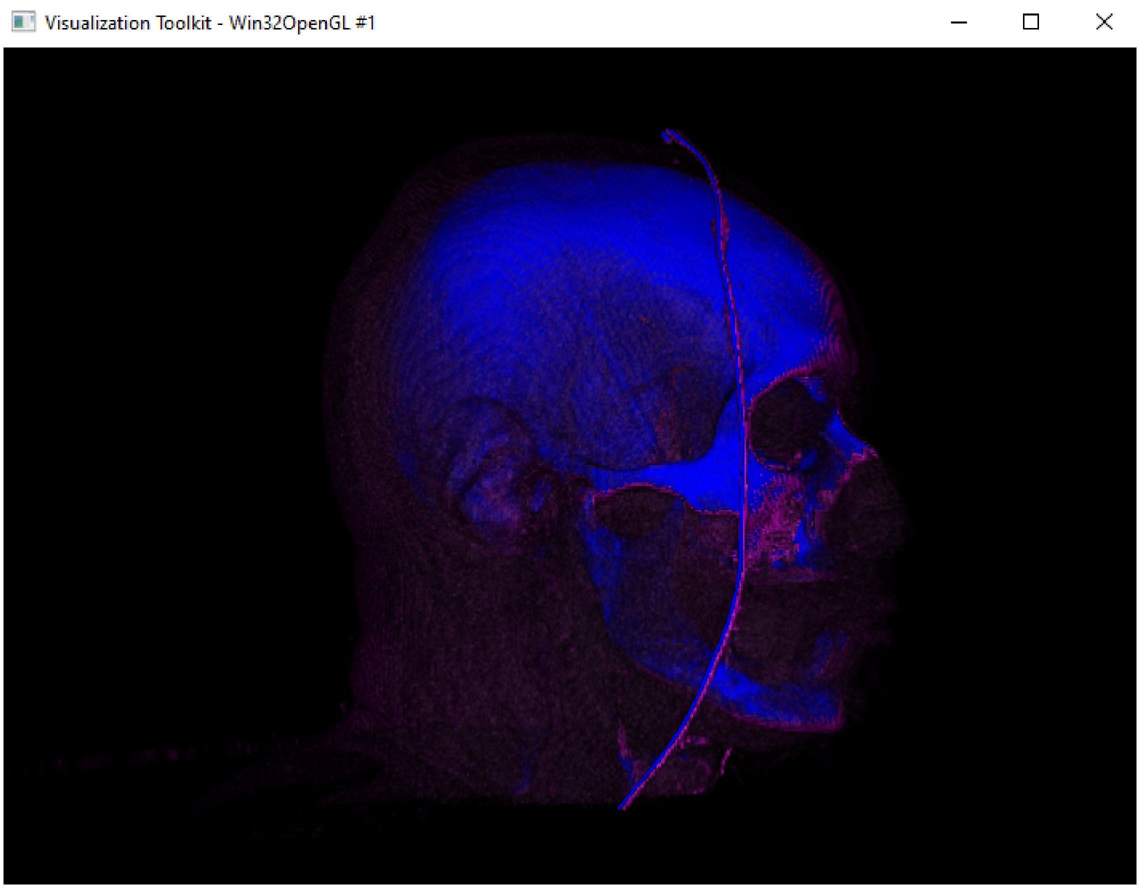


Figure 4.2: rendering Human Head using only ray casting (Rendering time:5.1171s)

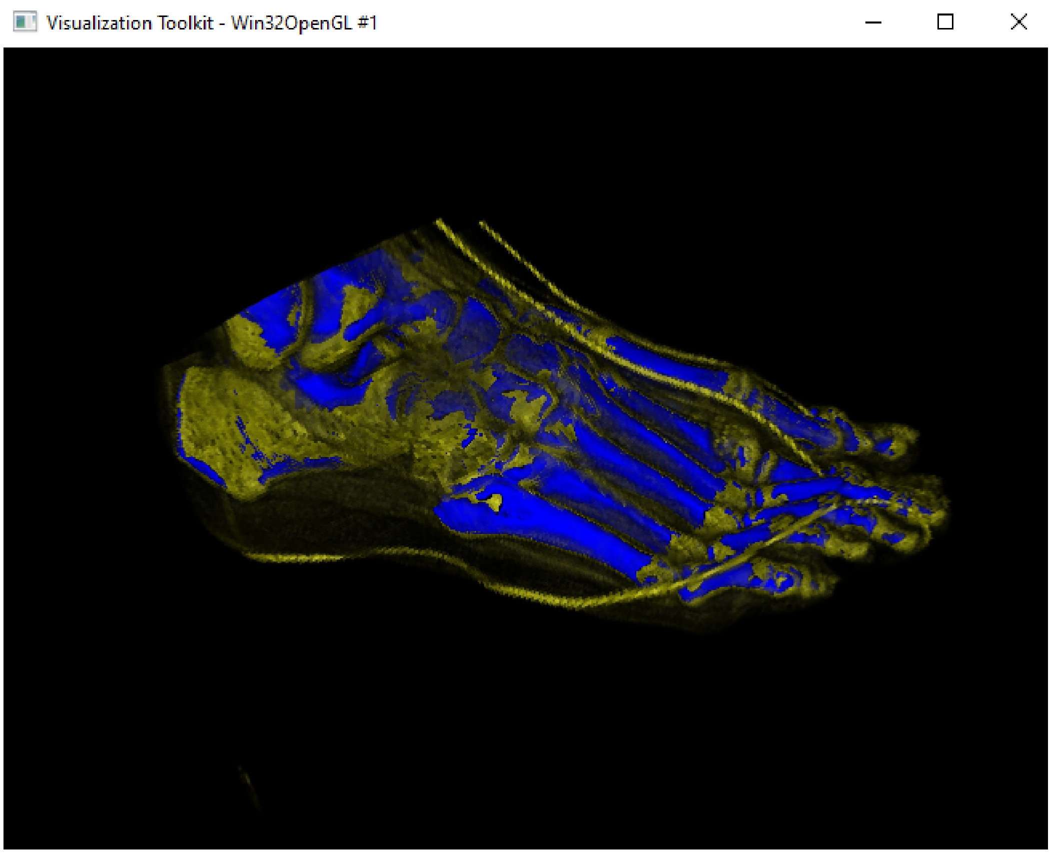


Figure 4.3: rendering Human Abkle using only ray casting (Rendering time:2.9694)

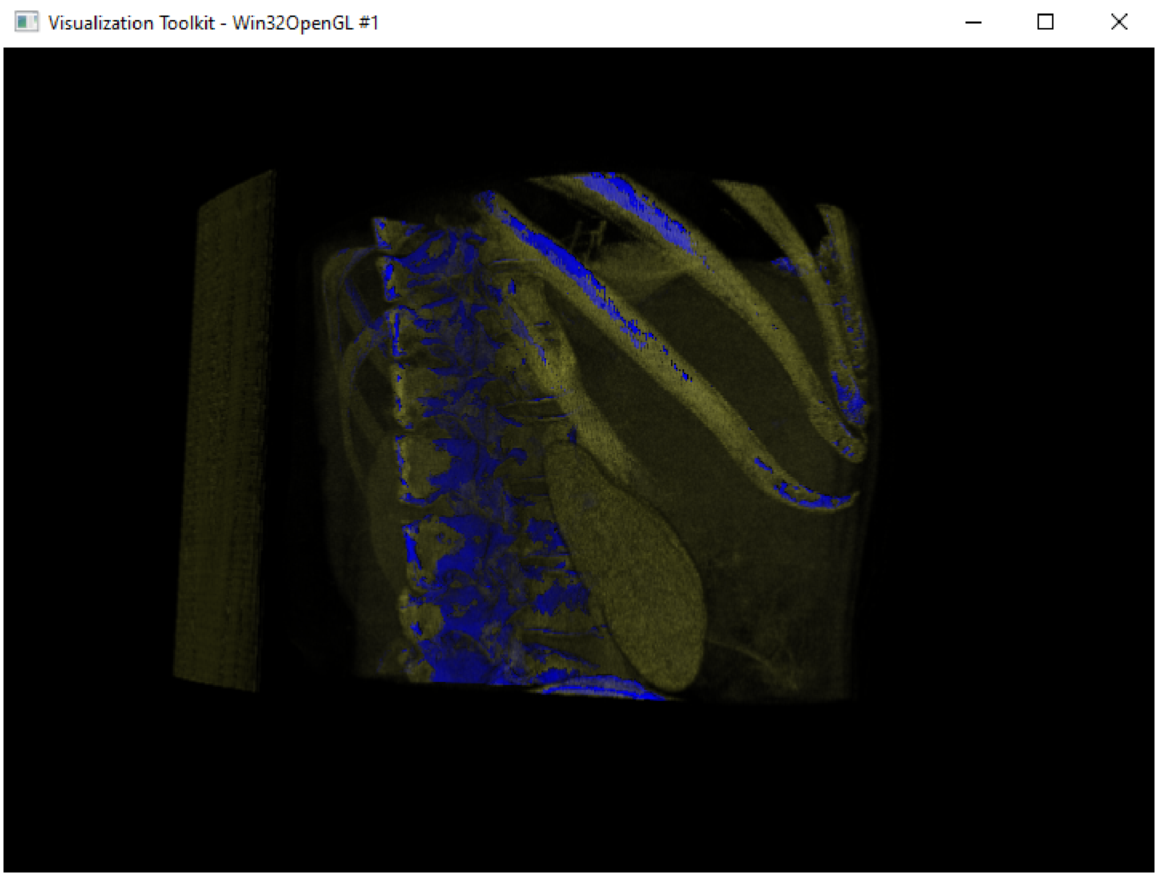


Figure 4.4: rendering Human Chest using only ray casting(Rendering time:4.1107s)

4.6.2 Results with Chebychev Distance Maps

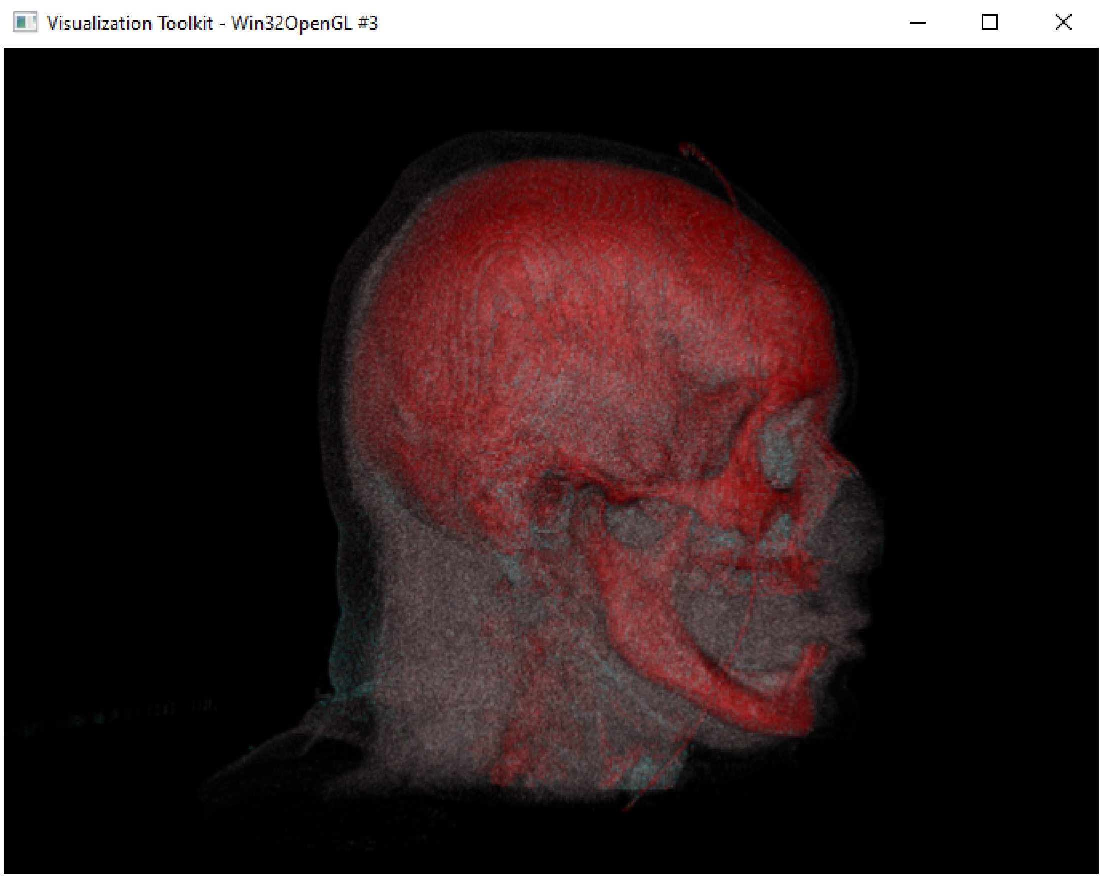


Figure 4.5: Rendering Human Head using Chebyshev Distance Maps based raycasting (Rendering time:1.5029s)

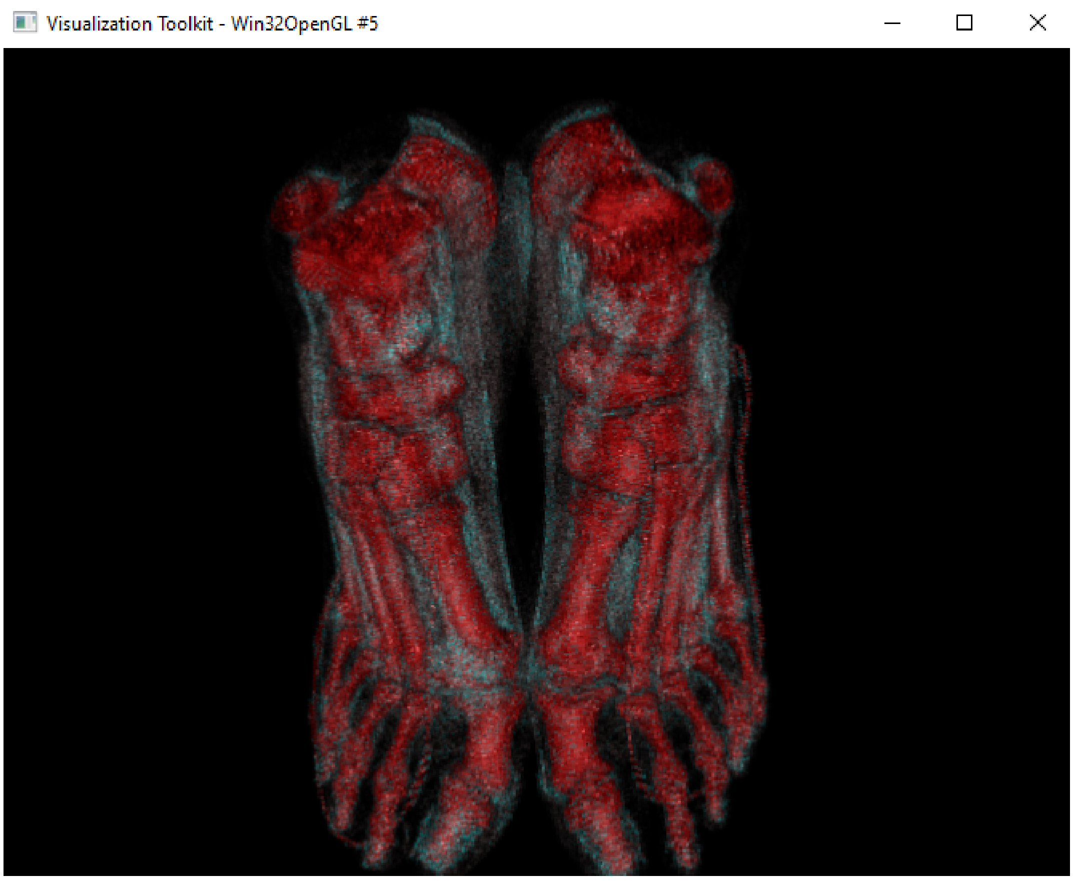


Figure 4.6: Rendering Human Ankle using Chebyshev Distance Maps based ray casting (Rendering time = 1.1495s)

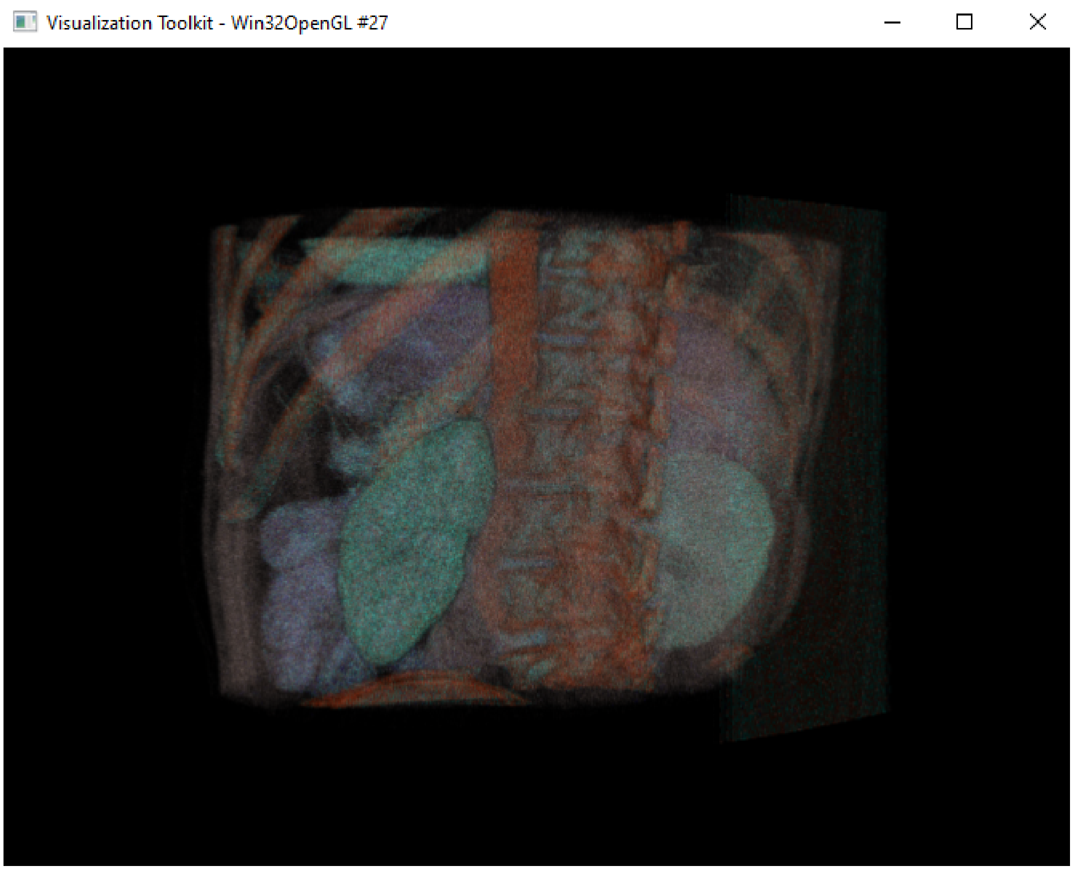


Figure 4.7: Rendering Human Chest using Chebyshev Distance Maps based ray casting (Rendering time: 1.9244s)

4.7 Analyze and Discussion

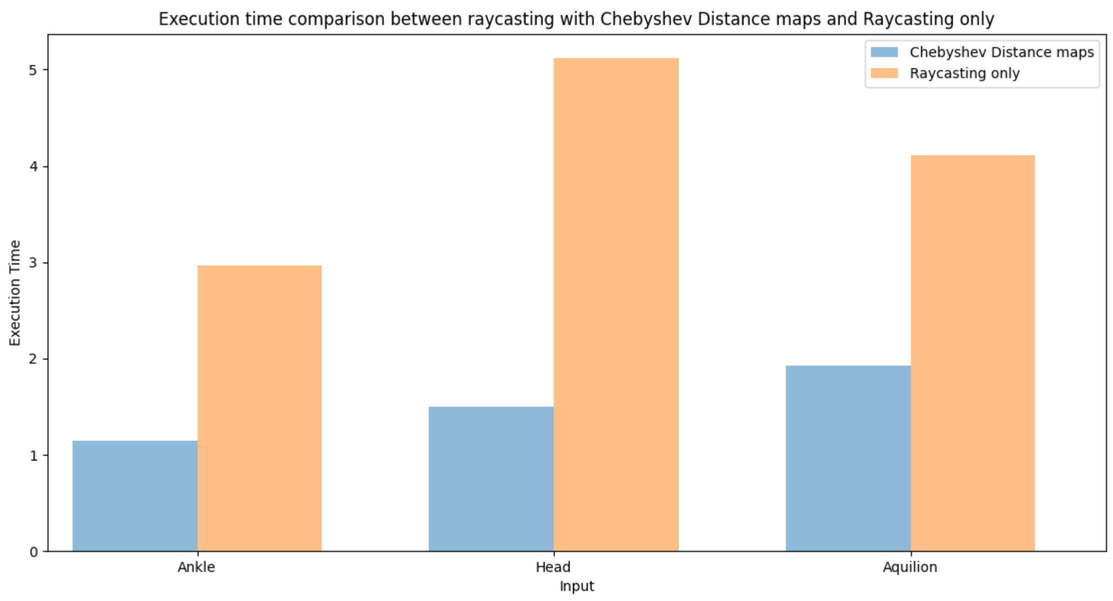


Figure 4.8: Execution time comparison between raycasting with Chebyshev Distance maps and Raycasting only

The analysis reveals that utilizing Chebyshev Distance Maps leads to faster computations compared to raycasting-only methods. By incorporating Chebyshev Distance Maps, which involve ray-casting techniques, the computational efficiency surpasses that of the raycasting-only approach. This highlights the superior performance achieved by employing Chebyshev Distance Maps for computational tasks. The method's ability to optimize and streamline calculations, leveraging the power of Chebyshev Distance Maps, results in expedited solutions and improved execution speed. Embracing Chebyshev Distance Maps showcases the relentless pursuit of efficiency and progress in the realm of computational science, offering enhanced productivity and the potential for advancements in scientific and technological endeavors.

4.8 Conclusion

This project successfully implemented a software tool for volumetric picture rendering using raycasting and distance mapping methods. The utilization of PyQt5 and VTK libraries, along with a modular and object-oriented design, enabled the development of an interactive visualization tool that simplifies the exploration and understanding of volumetric data.

The dataset selection process ensured the choice of appropriate volumetric data that represents the target domain, considering factors such as complexity, data size, availability of ground truth or reference data, and compatibility with VTK and DICOM format. This allowed the visualization tool to demonstrate its capabilities on meaningful and representative data.

The experimental setup, comprising the hardware and software configurations, provided a controlled environment for testing and validating the visualization tool. A powerful computer system and the use of PyQt5 and VTK libraries ensured efficient implementation and real-time interactive visualization.

General Conclusion

In this project, we demonstrated an implementation that uses raycasting rendering methods along with Euclidean and Chebyshev distance maps to produce effective and excellent volumetric visuals. The used Chebychev Distance maps-based method was designed to hasten the rendering process and enhance the visual fidelity of volumetric images. We efficiently bypassed empty spots and enhanced the sampling process by combining the two distance map algorithms, lowering computing costs and increasing volume rendering efficiency. We were able to leverage raycasting techniques to speed up the rendering process. We eliminated unnecessary computations and increased performance by taking advantage of the unique properties of the Chebyshev distance.

In the theoretical section of the method chapter,

Chapter 1 provided an overview of the key concepts and foundations of volume rendering. It explained the importance of visualizing volumetric data and discusses the process of generating 2D images from 3D volumetric data. The chapter covered different types of volumetric data, volume rendering techniques (direct and indirect), transfer functions, and popular volume rendering tools.

Chapter 2 focused on volumetric ray casting and explores various aspects of this technique. It discussed sampling methods, which aim to capture the details of the volumetric data. The chapter also covered empty space skipping, an efficient technique for bypassing empty regions during the ray-casting process. It further examined acceleration techniques and distance maps, which provide valuable information about the distances from voxels to surfaces within the volume.

Chapter 3 presented the developed method for efficient volume ray casting. It introduced the Block Empty Space Skipping (Block ESS) method and the Chebyshev Distance Empty Space Skipping technique.

Lastly, Chapter 4 focused on the implementation details and results of the efficient volume ray casting method. It described the experimental setup, including hardware and software configurations, and discussed the criteria used for dataset selection. The chapter presents the experimental results achieved through the utilization of the Block ESS method and the integration of the Chebyshev Distance Empty Space Skipping technique.

Further research can be done to determine the best way to use GPU resources and sophisticated sampling techniques, which could speed up and improve rendering even further. Furthermore, combining several distance map methods and examining alternate volume rendering algorithms may result in representations that are even more precise and lifelike.

Bibliography

[1] *Advances in discrete geometry applied to the extraction of planes and surfaces from 3D volumes.* PhD thesis, 2000. URL `http://e-learning.byclb.com/Chapter-5-Classification-of-Volume-Rendering-Algorithms`.

[2] Nvidia annotation plugin., 2019. URL `https://www.mitk.org/wiki/NvidiaAnnotationPlugin`.

[3] Medical imaging interaction toolkit., 2022. URL `https://docs.mitk.org/2022.10/`.

[4] Esalm Adel. Visualization toolkit (vtk). URL `https://sbme-tutorials.github.io/2019/CG/presentations/8_week8/#21`.

[5] B Afeez Mayowa, B Lateef Adebayo, and Taofikat Abidemi A Maxwell Obubu. Visualization of voxel volume emission and absorption of light in medical biology. curr tre biosta & biometr 1 (3)-2019. *CTBB. MS. ID*, 114, .

[6] B Afeez Mayowa, B Lateef Adebayo, and Taofikat Abidemi A Maxwell Obubu. Visualization of voxel volume emission and absorption of light in medical biology. curr tre biosta & biometr 1 (3)-2019. *CTBB. MS. ID*, 114, .

[7] John Amanatides, Andrew Woo, et al. A fast voxel traversal algorithm for ray tracing. In *Eurographics*, volume 87, pages 3–10, 1987.

[8] Brett Beeson, David G Barnes, and Paul D Bourke. A distributed data implementation of the perspective shear-warp volume rendering algorithm for visualisation of large astronomical cubes. *Publications of the Astronomical Society of Australia*, 20 (3):300–313, 2003.

[9] Gunilla Borgefors. Distance transformations in digital images. *Computer vision, graphics, and image processing*, 34(3):344–371, 1986.

[10] Christian Boucheny, Georges-Pierre Bonneau, Jacques Droulez, Guillaume Thibault, and Stéphane Ploix. A perceptive evaluation of volume rendering techniques. *ACM Transactions on Applied Perception (TAP)*, 5(4):1–24, 2009.

[11] Stefan Bruckner and M Eduard Groller. Exploded views for volume data. *IEEE transactions on visualization and computer graphics*, 12(5):1077–1084, 2006.

[12] Mario Ciampi, Luigi Gallo, and Giuseppe De Pietro. Mito: An advanced toolkit for medical imaging processing and visualization, 07 2018.

[13] John Congote, Luis Kabongo, Aitor Moreno, Alvaro Segura, Andoni Beristain, Jorge Posada, Oscar Ruiz, et al. Volume ray casting in webgl. *Computer Graphics*, pages 157–178, 2012.

[14] Joseph M Cooke, Michael J Zyda, David R Pratt, and Robert B McGhee. Npsnet: Flight simulation dynamic modeling using quaternions. *Presence: Teleoperators & Virtual Environments*, 1(4):404–420, 1992.

[15] F Dachille. Volume visualization algorithms and architectures. *Research Proficiency Examination, SUNY at Stony Brook*, 1997.

[16] Francisco de Moura Pinto and Carla MDS Freitas. Design of multi-dimensional transfer functions using dimensional reduction. In *Proceedings of the 9th Joint Eurographics/IEEE VGTC conference on Visualization*, pages 131–138, 2007.

[17] Lachlan Deakin and Mark Knackstedt. Accelerated volume rendering with chebyshev distance maps. In *SIGGRAPH Asia 2019 Technical Briefs*, pages 25–28. 2019.

[18] Lachlan J Deakin and Mark A Knackstedt. Efficient ray casting of volumetric images using distance maps for empty space skipping. *Computational Visual Media*, 6:53–63, 2020.

[19] Robert A Drebin, Loren Carpenter, and Pat Hanrahan. Volume rendering. *ACM Siggraph Computer Graphics*, 22(4):65–74, 1988.

[20] Oscar Figueiredo. *Advances in discrete geometry applied to the extraction of planes and surfaces from 3D volumes*. PhD thesis, Verlag nicht ermittelbar, 1999.

[21] Charles Florin, Romain Moreau-Gobard, and Jim Williams. Automatic heart peripheral vessels segmentation based on a normal mip ray casting technique. In *Medical Image Computing and Computer-Assisted Intervention–MICCAI 2004: 7th International Conference, Saint-Malo, France, September 26-29, 2004. Proceedings, Part I 7*, pages 483–490. Springer, 2004.

[22] James D Foley, Foley Dan Van, Andries Van Dam, Steven K Feiner, and John F Hughes. *Computer graphics: principles and practice*, volume 12110. Addison-Wesley Professional, 1996.

[23] Joseph L Gabbard, J Edward Swan, and Deborah Hix. The effects of text drawing styles, background textures, and natural lighting on text legibility in outdoor augmented reality. *Presence*, 15(1):16–32, 2006.

[24] Sarah FF Gibson. Using distance maps for accurate surface representation in sampled volumes. In *Proceedings of the 1998 IEEE symposium on Volume visualization*, pages 23–30, 1998.

[25] Thomas Günther, Christoph Poliwoda, Christof Reinhart, Jürgen Hesser, Reinhard Männer, H-P Meinzer, and H-J Baur. Virim: A massively parallel processor for real-time volume visualization in medicine. *Computers & Graphics*, 19(5):705–710, 1995.

[26] Markus Hadwiger, Christian Sigg, Henning Scharsach, Katja Bühler, and Markus H Gross. Real-time ray-casting and advanced shading of discrete isosurfaces. In *Computer graphics forum*, volume 24, pages 303–312. Citeseer, 2005.

[27] Markus Hadwiger, Christian Sigg, Henning Scharsach, Katja Bühler, and Markus H Gross. Real-time ray-casting and advanced shading of discrete isosurfaces. In *Computer graphics forum*, volume 24, pages 303–312. Citeseer, 2005.

[28] Markus Hadwiger, Patric Ljung, Christof Rezk Salama, and Timo Ropinski. Advanced illumination techniques for gpu volume raycasting. In *ACM Siggraph Asia 2008 Courses*, pages 1–166. 2008.

[29] Markus Hadwiger, Ali K. Al-Awami, Johanna Beyer, Marco Agus, and Hanspeter Pfister. Sparseleap: Efficient empty space skipping for large-scale volume rendering. *IEEE Transactions on Visualization and Computer Graphics (Proceedings IEEE Scientific Visualization 2017)*, 24(1):974–983, 2018.

[30] Helwig Hauser, Lukas Mroz, G Italo Bischi, and M Eduard Groller. Two-level volume rendering. *IEEE transactions on visualization and computer graphics*, 7(3):242–252, 2001.

[31] Jürgen Hesser, Reinhard Manner, Günter Knittel, Wolfgang Straßer, Hanspeter Pfister, and Arie Kaufman. Three architectures for volume rendering. In *Computer Graphics Forum*, volume 14, pages 111–122. Wiley Online Library, 1995.

[32] Jürgen Hesser, Reinhard Manner, Günter Knittel, Wolfgang Straßer, Hanspeter Pfister, and Arie Kaufman. Three architectures for volume rendering. In *Computer Graphics Forum*, volume 14, pages 111–122. Wiley Online Library, 1995.

[33] Denis Horvat and B Zalik. Ray-casting point-in-polyhedron test. *Proceedings of the CESCG*, 2012.

[34] Yue Jiang, Dantong Ji, Zhizhong Han, and Matthias Zwicker. Sdfdiff: Differentiable rendering of signed distance fields for 3d shape optimization. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 1251–1261, 2020.

[35] Mark W Jones. The production of volume data from triangular meshes using voxelisation. In *Computer Graphics Forum*, volume 15, pages 311–318. Wiley Online Library, 1996.

[36] Srinivas Kaza et al. *Differentiable volume rendering using signed distance functions.* PhD thesis, Massachusetts Institute of Technology, 2019.

[37] Aaron Knoll, Younis Hijazi, Rolf Westerteiger, Mathias Schott, Charles Hansen, and Hans Hagen. Volume ray casting with peak finding and differential sampling. *IEEE Transactions on Visualization and Computer Graphics*, 15(6):1571–1578, 2009.

[38] Aaron Knoll, Younis Hijazi, Rolf Westerteiger, Mathias Schott, Charles Hansen, and Hans Hagen. Volume ray casting with peak finding and differential sampling. *IEEE Transactions on Visualization and Computer Graphics*, 15(6):1571–1578, 2009.

[39] Jens Kruger and Rüdiger Westermann. Acceleration techniques for gpu-based volume rendering. In *IEEE Visualization, 2003. VIS 2003.*, pages 287–292. IEEE, 2003.

[40] Philippe Lacroute and Marc 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*, pages 451–458, 1994.

[41] Byeonghun Lee, Jihye Yun, Jinwook Seo, Byonghyo Shim, Yeong-Gil Shin, and Bohyoung Kim. Fast high-quality volume ray casting with virtual samplings. *IEEE Transactions on visualization and computer graphics*, 16(6):1525–1532, 2010.

[42] Marc Levoy. Display of surfaces from volume data. *IEEE Computer graphics and Applications*, 8(3):29–37, 1988.

[43] Marc Levoy. Display of surfaces from volume data. *IEEE Computer graphics and Applications*, 8(3):29–37, 1988.

[44] Marc Levoy. A hybrid ray tracer for rendering polygon and volume data. *IEEE Computer graphics and Applications*, 10(2):33–40, 1990.

[45] Sukhyun Lim and Byeong-Seok Shin. A distance template for octree traversal in cpu-based volume ray casting. *The Visual Computer*, 24:229–237, 2008.

[46] Patric Ljung. Efficient methods for direct volume rendering of large data sets. 03 2023.

[47] Patric Ljung, Jens Krüger, Eduard Groller, Markus Hadwiger, Charles D Hansen, and Anders Ynnerman. State of the art in transfer functions for direct volume rendering. In *Computer Graphics Forum*, volume 35, pages 669–691. Wiley Online Library, 2016.

[48] WE Lorensen and HE Cline. Marching cubes: a high resolution 3d surface construction algorithm. siggraph comput. graph. 21 (4), 163–169 (1987).

[49] Dening Luo. Interactive volume illumination of slice-based ray casting. *arXiv preprint arXiv:2008.06134*, 2020.

[50] Amr S Mady and Samir Abou El-Seoud. An overview of volume rendering techniques for medical imaging. 2020.

[51] Amr S Mady and Samir Abou El-Seoud. An overview of volume rendering techniques for medical imaging. 2020.

[52] Jennis Meyer-Spradow, Timo Ropinski, Jörg Mensmann, and Klaus Hinrichs. Voreen: A rapid-prototyping environment for ray-casting-based volume visualizations. *IEEE Computer Graphics and Applications*, 29(6):6–13, 2009.

[53] NEUBIAS. Volume rendering examples., 2019. URL `https://neubias.github.io/training-resources/volume_viewer/index.html`.

[54] NEUBIAS. Texture coordinate interpretation, 2019. URL https://microsoft.github.io/DirectX-Specs/d3d/archive/D3D11_3_FunctionalSpec.htm#3.3.3%20Texture%20Coordinate%20Interpretation.

[55] Dr. Savvas Nicolaou. cinematic rendering., 2019. URL https://www.auntminnie.com/index.aspx?sec=ser&sub=def&pag=dis&ItemID=126659.

[56] Tobias Nilsson. Optimization methods for direct volume rendering on the client side web, 2019.

[57] Christian John Noon. *A volume rendering engine for desktops, laptops, mobile devices and immersive virtual reality systems using GPU-based volume raycasting*. Iowa State University, 2012.

[58] Bernhard Preim and Dirk Bartz. Chapter 07 - indirect volume visualization. In Bernhard Preim and Dirk Bartz, editors, *Visualization in Medicine*, The Morgan Kaufmann Series in Computer Graphics, pages 155–182. Morgan Kaufmann, Burlington, 2007. isbn:978-0-12-370596-9. doi:https://doi.org/10.1016/B978-012370596-9/50010-9. URL https://www.sciencedirect.com/science/article/pii/B9780123705969500109.

[59] Harvey Ray, Hanspeter Pfister, Deborah Silver, and Todd A Cook. Ray casting architectures for volume visualization. *IEEE Transactions on Visualization and Computer Graphics*, 5(3):210–223, 1999.

[60] Scott D Roth. Ray casting for modeling solids. *Computer graphics and image processing*, 18(2):109–144, 1982.

[61] W Schroeder, K Martin, and B Lorensen. The visualization toolkit, 4th edn. kitware. *New York*, 2006.

[62] William J Schroeder, Lisa Sobierajski Avila, and William Hoffman. Visualizing with vtk: a tutorial. *IEEE Computer graphics and applications*, 20(5):20–27, 2000.

[63] Peter Shirley and Allan Tuchman. A polygonal approximation to direct scalar volume rendering. *ACM Siggraph Computer Graphics*, 24(5):63–70, 1990.

[64] Luc Soler, Stéphane Nicolau, Patrick Pessaux, Didier Mutter, and Jacques Marescaux. Real-time 3d image reconstruction guidance in liver resection surgery. *Hepatobiliary surgery and nutrition*, 3:73–81, 04 2014. doi:10.3978/j.issn.2304-3881.2014.02.03.

[65] Vincent Vidal, Xing Mei, and Philippe Decaudin. Simple empty-space removal for interactive volume rendering. *Journal of Graphics Tools*, 13(2):21–36, 2008.

[66] Yinong Wang, Weibei Dou, and Jean-Marc Constans. Accelerating volume ray casting by empty space skipping used for computer-aided therapy. pages 661–667, 07 2012. isbn:978-1-4673-0173-2. doi:10.1109/ICALIP.2012.6376699.

[67] Chaoqing Xu, Guodao Sun, and Ronghua Liang. A survey of volume visualization techniques for feature enhancement. *Visual Informatics*, 5(3):70–81, 2021. ISSN 2468-502X. `doi:https://doi.org/10.1016/j.visinf.2021.08.001`. URL `https://www.sciencedirect.com/science/article/pii/S2468502X21000358`.

[68] Ling Yang, Feng Ling, Ni-Ni Rao, and Zhong-Ke Wang. A new algorithm of maximum intensity projection based on context-preserving illustrative volume rendering model. In *2010 3rd International Congress on Image and Signal Processing*, volume 5, pages 2381–2386. IEEE, 2010.

[69] Stefan Zellmann, Jürgen P Schulze, Ulrich Lang, H Childs, and F Cucchietti. Rapid kd tree construction for sparse volume data. In *EGPGV@ EuroVis*, pages 69–77, 2018.

[70] Qi Zhang. Medical data visual synchronization and information interaction using internet-based graphics rendering and message-oriented streaming. *Informatics in Medicine Unlocked*, 17:100253, 2019.

[71] Meng Zhu, Donald House, and Mark Carlson. Ray casting sparse level sets. In *Proceedings of the Digital Production Symposium*, pages 67–72, 2012.

[72] Xiangyu Zhu, Xiaoming Liu, Zhen Lei, and Stan Z Li. Face alignment in full pose range: A 3d total solution. *IEEE transactions on pattern analysis and machine intelligence*, 41(1):78–92, 2017.

[73] Karel J Zuiderveld, Anton HJ Koning, and Max A Viergever. Acceleration of ray-casting using 3-d distance transforms. In *Visualization in Biomedical Computing'92*, volume 1808, pages 324–335. SPIE, 1992.