University of Ljubljana
Faculty of Computer and Information Science

Jon Škoberne

Volume Rendering of Medical Image Data in Mixed Reality

master's thesis

the 2nd cycle master's study programme
computer and information science

Supervisor: izr. prof. dr. Matija Marolt
Co-supervisor: doc. dr. Jan Egger

Ljubljana, 2021

Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Jon Škoberne

Volumetrično izrisovanje medicinskih podatkov v mešani resničnosti

magistrsko delo

magistrski študijski program druge stopnje
računalništvo in informatika

Mentor: izr. prof. dr. Matija Marolt
Somentor: doc. dr. Jan Egger

Ljubljana, 2021

Acknowledgments

Worth mentioning in the acknowledgment is everyone who contributed to your thesis.

Jon Škoberne, 2021

To all the flowers of this world.

"The only reason for time is so that everything doesn't happen at once."

— Albert Einstein

Contents

List of used acronmys

acronym	meaning
CA	classification accuracy
DBMS	database management system
SVM	support vector machine
...	...

Abstract

Title: Volume Rendering of Medical Image Data in Mixed Reality

This sample document presents an approach to typesetting your BSc thesis using LaTeX. A proper abstract should contain around 100 words which makes this one way too short. A good abstract contains: (1) a short description of the tackled problem, (2) a short description of your approach to solving the problem, and (3) (the most successful) result or contribution in your thesis.

Keywords

Volume rendering, augmented reality, head and neck cancer surgery

Povzetek

Naslov: Volumetrično izrisovanje medicinskih podatkov v mešani resničnosti

V vzorcu je predstavljen postopek priprave magistrskega dela z uporabo okolja LaTeX. Vaš povzetek mora sicer vsebovati približno 100 besed, ta tukaj je odločno prekratek. Dober povzetek vključuje: (1) kratek opis obravnavanega problema, (2) kratek opis vašega pristopa za reševanje tega problema in (3) (najbolj uspešen) rezultat ali prispevek magistrske naloge.

Ključne besede

volumetrično izrisovanje, obogatena resničnost, operacija raka na glavi in vratu

Razširjeni povzetek

To je primer razširjenega povzetka v slovenščini, ki je obvezen za naloge pisane v angleščini. Razširjeni povzetek mora vsebovati vse glavne elemente dela napisanega v angleščini skupaj s kratkim uvodom in povzetkom glavnih elementov metode, glavnih eksperimentalnih rezultatov in glavnih ugotovitev. Razširjeni povzetek naj bo strukturiran v podpoglavja (spodaj je naveden le okvirni primer in je nezavezujoč). Čez palec navadno razširjeni povzetek nanese okoli 10 odstotkov obsega celotnega dela.

I Kratek pregled sorodnih del

II Predlagana metoda

III Eksperimentalna evaluacija

IV Sklep

poljuben tekst poljuben tekst

poljuben tekst poljuben tekst

Chapter 1

Introduction

1.1 Problem definition

The evolution of medical visualization stands on the following foundations: firstly, the long tradition of scientists to illustrate their work, a quintessential example would be Da Vinci's paintings. The second pillar is computer graphics, which encapsulate the efficient algorithms that render this imaging data. Medical imaging data is inherently 3D, for example computed tomography (CT), magnetic resonance imaging (MRI) scans, and is very large, thus it requires efficient visualization algorithms. Medical practitioners use medical data for preoperative decision making, setting up a diagnosis, providing intraoperative guidance, documentation, educational purposes and medical research [1, 2]. Although Radiologists are trained in mentally mapping from 2D slices, such as CT and MRI scans, to 3D structures, they can still benefit from a 3D overview, when they encounter complex 3D structures. Physicians, on the other hand, do not have this training and work mostly in the spatial domain, such as surgery or therapy, and would profit from visualized 3D data in its natural and physical reference space with additional options of interactivity [3]. In this regard, medical augmented reality aims to support physicians in their daily routine, from decision making to intraoperative guidance. Essentially granting them "X-ray vision". Therefore, an augmented reality

system for head and neck cancer surgery using an optical see-through head mounted display (OST-HMD) is being developed at the Technical University of Graz [4]. The system displays surface renderings and 2D slices, which are overlaid with the patient. We wanted to use volume rendering algorithms to extend the system to visualize CT and MRI scans in 3D with added interactivity. We developed a system that uses volume rendering techniques for displaying 3D data, such as CT and MRI scans, that can be interacted with. The volume rendering and interactions were developed in Unity game engine. The application runs on a desktop PC from where the visuals are streamed to the Microsoft HoloLens. On the Microsoft HoloLens the user is able to see and interact with the 3D visualization.

1.2 Term definitions

1.3 Review of literature

Something similar to our work has already been done [5], where the authors use HoloLens with additional dedicated hardware to perform the rendering and stream it to the HoloLens. They manage to get around 18 frames per second (FPS) running the system on a desktop with NVIDIA GTX 1080 GPU, which is still too slow for real world use, though with scaling up the hardware the goal of 30 FPS could be achieved. In [6] authors present their implementation of a hardware accelerated volume renderer for augmented reality using a video see-through head mounted display. Their system allows users direct real-time stereo visualization with additional implementation of a virtual mirror and occlusion handling with physician's hands and medical instruments. Authors in [7] implemented an augmented reality (AR) framework for minimally invasive surgeries (MIS), which achieves interactive geometric aware AR in endoscopic surgery with stereo views. With this framework they are able to interactively add annotations, localization of tumors and vessels and other measurement labellings. Another MIS is liver

resection surgery [8], where authors developed an AR system, which increases surgeon's vision by providing virtual transparency of the patient. The system overlies the rendered data with the patient to achieve virtual transparency. They conclude that the results are promising, but point out that the AR is still limited due to organ deformation during surgery. In [9] authors used AR to utilize computed tomography angiography (CTA) in 3D world. They delineated from preoperative CTA scans volumes of osseous, vascular, skin, soft tissue structures and generated 3D images using segmentation software packages. These were converted to polygon models and displayed on the HoloLens head mounted display. The surgeon was able with the system to show precise and efficient localization of perforating vessels. In [10, 11, 12, 13] a novel approach called cinematic rendering (CT) is discussed and compared to established techniques [14]. Cinematic rendering is a 3D rendering algorithm that tries to simulate the light propagation and interaction as they pass through the volume. It is used to render a more photorealistic 3D image. In [15] researchers used CT for rendering the appearance of a tumor, named odontogenic myxofibroma, in the mandible. Although more work needs to be done in order to better compare CT versus traditional methods of visualization, they point out that realistic shadowing effects create a better depth and relative distances between objects in the volume perception. A major topic in volumetric rendering is the transfer function (TF), which servers a role in transforming data into color and opacity to enhance perception of relevant features in the data. TF is also important, because it allows for interactive volumetric exploration of data. In [16] authors provide an overview of research of TF. In [17] an enhanced real-time volumetric visualization is presented. Authors undertake the issue of visualizations with indistinct tissue features and non-optimal depth perception. They implement a volumetric rendering pipeline, which includes volumetric median and Gaussian filtering, boundary and feature enhancement, depth encoding and lighting into a ray casting volume rendering model. This pipeline is then used in real time for 3D visualizations of pathology and maneuvers during the operation. Authors

conclude that it produces better results than the standard ray casting technique. In the following work [18] an in depth comparison between volume rendering (VR) and cinematic rendering (CR) is done by the authors. Both techniques are used for visualizing 3-D data, especially for high density and high contrast regions such as bones and contrast enhanced vessels, but CR should provide a more natural and realistic illumination of the data. Volume rendering (VR) is nowadays a standard technique, but at the beginning it had some struggles, especially because of its computational cost and therefore long processing times. Today a new variation is being discussed in the medical field, named cinematic rendering (CR), which traces its origins back to the entertainment industry. There the technique is used for rendering photorealistic animation (movies). VR consists of two major steps: classification of voxels and image projection. Classification, in the basic sense, determines how each point on the ray that passes through the image data, contributes to the final value in the image. For each voxel in the dataset we compute a weighted sum of each tissue type that is present in the voxel. Each tissue type in the voxel has determined attenuation threshold levels and is assigned to specific colour and opacity. VR displays the image using the projection technique done by shooting rays through the volume consisting of classified voxels. Ray's colour and transparency is affected by the voxels it passes through. Final image can be modified by additional effects (reflections or shadows). The advantages of VR are: 3-D representation of CT image data, coloured display of CT image data and a relatively simple evaluation of different anatomical structures in a larger region. The most prominent downsides of VR are that it can obscure anatomical information - for example 3-D CT angiography with bone subtraction provided much better results than the procedure without bone subtraction. Therefore one should always compare (correlate) results produced by VR with 2-D images, which were produced from the same image data. Physically based VR method, called cinematic rendering (CR), computes in real-time the complex physics of lighting effects. It is able to render shadows, ambient occlusion, colour transmittance,

multi-scattering and camera properties (aperture, exposure, shutter speed). It offers a more photorealistic visual of the medical image data, focusing on enhanced depth and shape perception. While VR only considers emission and absorption of energy along the ray and represents scattering effects with a local gradient shading model and thus ignores complex light paths, CR integrates the light scattered from all possible directions along a ray. Path tracing used in CT integrates a high number of rays (in theory they are infinite), therefore a Monte Carlo approach is necessary. The final value in the images is obtained by averaging Monte Carlo samples, which represent radiance at random positions with light scattered in random directions. Both VR and CR have the same foundational concept in common, which is segmentation of data based on voxel attenuation and colour look-up tables (brightness and opacity). Therefore CR has the same problem as VR and it can mask important information in the 3-D image. CR can provide the same display functions as VR, such as flying-through, flying-around and multiple views. Where CT really excels (or differs from VR) is in its more realistic lighting representation. Lighting in 3-D is a function of ambient, diffuse and specular light properties. Voxel brightness is determined by its light properties and position relative to other voxels and artificial light position. Because VR simplifies light paths (light scattering), the differences to the light emitted to the voxels are small. On the other hand, CR uses a more complex model discussed above, which bears in mind other voxels and subsequent reflections and introduces shadows, where artificial light is blocked by body parts. Authors of the study found out that the CR method is especially good with high-contrast structures (contrast-enhanced vessels and bones). CR majorly improves perception of depth and soft tissue structured, because of a more photorealistic image of medical data. The cost for this realistic representation that the CR is able to provide us with, is a higher computational power demand. Real-time display using the CR method is usually interrupted by recalculation processes, which can take several seconds. Authors stress that their research was focused on showing the image differences between the two

techniques and not diagnostic capabilities. Authors in their work [19] present their approach to achieve a realistic volume visualization. Many works before theirs have shown that a realistic illumination in volume visualization improves people's 3D shape perception. Most of previous works focused on improving the standard direct volume rendering (DVR) with approximations of ambient occlusion, shadows, light scattering and global illumination. Physically-based modeling of illumination was avoided, because it was considered to be quite expensive. Authors implemented a Monte Carlo (MC) ray tracer, which combined with physically based light transport, accurately simulates real-world light interaction. They introduced hybrid scattering that combines surface and volumetric scattering.

Their DVR implementation creates multiple estimates every second, which are then combined to form a high quality image. At first the image is of lower quality, but it improves as new estimates are arriving. Though, when camera, transfer function or lighting are changed, the MC algorithm must restart from the start. The pipeline consists of stohastic ray-caster, which returns a high dynamic range (HDR) MC estimate. This estimate is then filtered with a Gaussian kernel (anti-aliasing). MC integration is done with cumulative moving average. Then HDR MC estimate is tone mapped (transforming HDR to low dynamic range (LDR)) and gamma corrected. The resulting LDR image is then an-isotropically filtered to reduce noise. The stohastic ray-caster models a real world camera with a thin lens model. With this model the ray-caster can produce realistic depth-of-field effects. The ray is constructed with two points - one point sampled on the lens and another sampled from the film plane. Rays that do not intersect with the volume's bounding box are discarded. Usually rays are propagated through the volume with a ray marching (RM) technique, but because shadow calculations are required at every sample point, which makes RM very costly, authors decided to use the Woodcock tracking. Woodcock tracking propagates the ray through the volume with steps of random length and only returns a single scattering point. Because only a single scattering point is returned, fewer

shadow calculations are computed. Another bonus of Woodcock tracking is that it is unbiased. Woodcock tracking calculates where the scattering effect occurs. It randomly selects the length, which is then accepted with some probability. This procedure is then repeated until the selected length for collision is accepted. Single scattering contribution at the scattering point is computed in the following manner, the contribution of light that travels along the ray is calculated by stohastically sampling a point on stohastically chosen light source. Scattering contribution is obtained by sampling the scattering function and finally both results, the light contribution and the scattering contribution, are combined using multiple importance sampling. Most MC ray-tracers are not concerned with shading normals and use phase functions for volumetric scattering computation. Authors claim that they achieve more vivid images with their hybrid scattering method, which mixes surface and volumetric scattering. In areas, where gradient vectors are well defined, surface scattering is used, otherwise the method uses volume scattering. The framework was tested in how many estimates (frames) per second were achieved at resolution of $800x600$ pixels and the rate of convergence, which was measured with normalized root mean squared (NRMS). This is an error between the running MC estimate and fully converged MC estimate. On five data sets, the framework achieved from 32-65 estimates per second and a NRMS of 10% in just a fraction of a second.

1.4 Description of the remaining chapters

Chapter 2

Theory and methods used

2.1 Medical image data

2.1.1 Data

In this section we will take a look at how the image data is structured and obtained. Medical image data requirements are the following:

- relevant anatomy must be entirely shown,
- minimize the costs and negative effects on the observed patient,
- resolution and image quality should be high enough for medical practitioner to be able to use it in their work process.

Volume data set is composed of many images, which represent only a thin slice of a captured body part with the computed tomography (CT) or magnetic resonance imaging (MRI). This image is composed of many pixels that are arranged on a two dimensional (2D) grid. These points are positioned a constant distance away from each other in both x and y direction. This distance is also called a pixel distance. Pixel $P_{i,j}$ is a pixel on a grid point at position i on the X axis and position j on the Y axis.

As mentioned before, volume data set, as it can be seen in the image 2.1, is composed of many images. These images or 2D grids are stacked next to

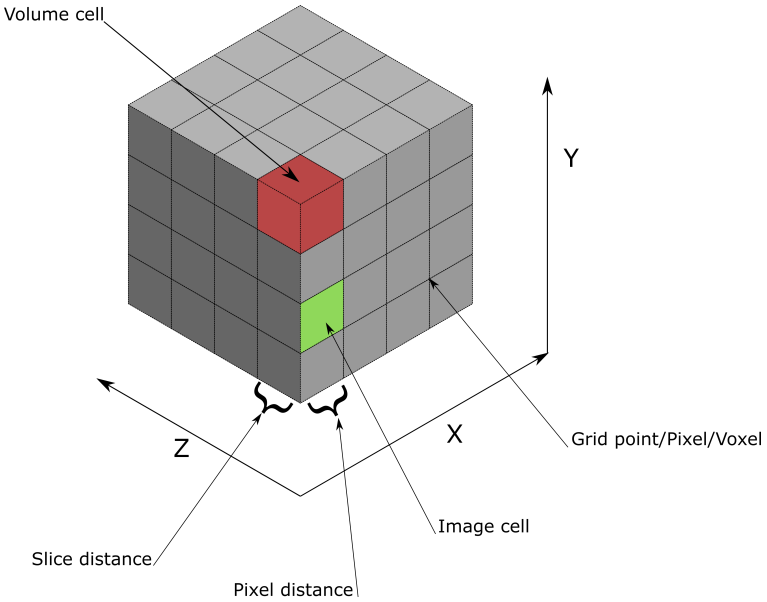

Figure 2.1: Volume data.

each other at a certain distance, which is named a slice distance. The slice distance is usually much larger than the pixel distance, therefore most data sets are anisotropic (on the other hand, isotropic data sets are those data sets, where pixel distance equals the slice distance). In this 3D case of data, the grid point is called a voxel. The voxel $V_{i,j,k}$ position is defined by voxel spacing, which are the three distances in X, Y and Z directions, and voxel's indices (i, j, k). Another important entity in the volume data set is a volume cell or cuboid. Volume cell is formed by eight neighbouring voxels as you can see in the image 2.2.

It is important to note that data is only defined at voxel positions. Unfortunately, we often have to calculate sample points, which are within the volume cell. In the old days of volumetric data processing, researchers often used nearest neighbour interpolation. This method just assigns the value of the nearest voxel to the sample point. It is a fast method, but the image quality is not good (blocky visuals). Nowadays trilinear interpolation is the most widespread method, which can also be seen in image 2.2. Trilinear

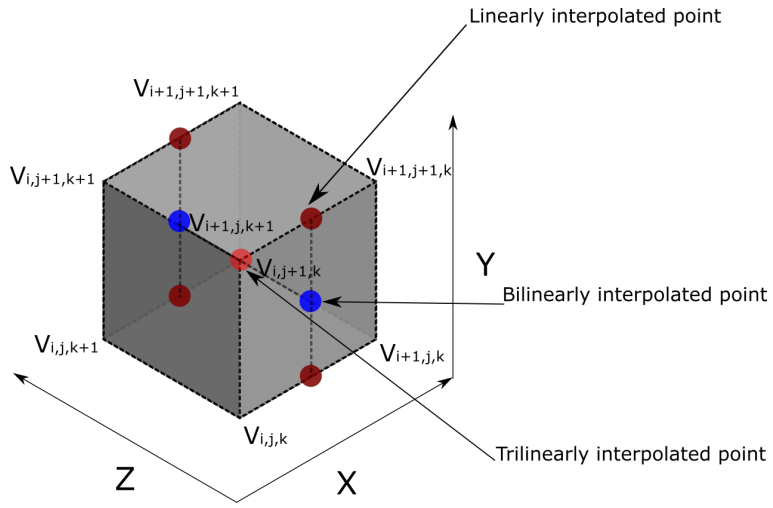

Figure 2.2: Volume cell and the sample point computed with trilinear interpolation.

interpolation first linearly interpolates the front and back edges, which results in four linearly interpolated points. Then it interpolates the two frontal points to get a point on the frontal face and does the same for the two points on the backward cuboid edges. To calculate the final sample point, the method interpolates these two (frontal and backward) face points. The general equation can be seen in (2.1).

$$\begin{aligned} L(\alpha) &= V_0 \cdot (1-\alpha) + V_1 \cdot \alpha, \qquad 0 \leq \alpha \leq 1 \\ B(\alpha_1, \alpha_2) &= L_1(\alpha_1) \cdot (1-\alpha_2) + L_2(\alpha_1) \cdot \alpha_2, \qquad 0 \leq \alpha_1, \alpha_2 \leq 1 \\ T(\alpha_1, \alpha_2, \alpha_3) &= B(\alpha_1, \alpha_2) \cdot (1-\alpha_3) + B(\alpha_1, \alpha_2) \cdot \alpha_3, \qquad 0 \leq \alpha_1, \alpha_2, \alpha_3 \leq 1 \end{aligned} \tag{2.1}$$

To use information from a larger neighbourhood and not just the observed volume cell, a tricubic spline function or a Gaussian filter can be used. The neighbourhood is a larger area around the volume cell and provides the spatial context of many operations. There are two relevant neighbourhoods: 6-neighbourhood and 26-neighbourhood. The first one consists only of the

voxels left/right, top/bottom, front/back from the observed voxel. In other words, it represents all volume cells that share a face with the considered volume cell. The 26-neighbourhood considers additional voxels on the two diagonals from the observed voxel. It represents all volume cells that share an edge or vertex with the currently observed volume cell.

Normalization of data

Volume data can be normalized between 0 and 1 by the following formula:

$$normalized_i = (data_i - data_{min})/data_{range} \tag{2.2}$$

where $data_i$ is data at index i, $data_{min}$ is the smallest value and $data_{range}$ is the delta between the largest and the smallest value.

Gradient - central differences

$$\nabla f(x_i, y_j, z_k) = (\frac{x_{i+1} - x_{i-1}}{2 \cdot h}, \frac{y_{j+1} - y_{j-1}}{2 \cdot h}, \frac{z_{k+1} - z_{k-1}}{2 \cdot h})^T \tag{2.3}$$

In (2.3) we can see the definition of the central differences method, which is used for normal calculation in volume data sets. Voxels x_i, y_j and z_k are surrounding the observed voxel in the x, y and z axis. The normal of the sample point is computed by trilinear interpolation of the normals of voxels of the volume cell. The method assumes a uniform grid (identical constant spacing in all three dimensions), which unfortunately results in visual artifacts if used on CT data that is anisotropic.

Gradient - central differences with applied Gaussian filter

The gradient computation is exactly the same as in 2.1.1, the only difference is that we also smooth the gradients with Gaussian smoothing.
Gaussian smoothing is a low-pass filter, meaning we filter out the high frequencies, which results in a smoother (blurry) image, we obtain it by convolving the image with a Gaussian function ($2D$ case):

$$G(x, y) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{x^2+y^2}{2\sigma^2}} \tag{2.4}$$

The $3x3$ kernel for $2D$ case:

$$G = \frac{1}{16} \cdot \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix} * A_{img} \tag{2.5}$$

which can also be decomposed in order to reduce the number of multiplication operations:

$$G = \frac{1}{16} \cdot \begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix} * (\begin{bmatrix} 1 & 2 & 1 \end{bmatrix} * A_{img}) \tag{2.6}$$

The $3x3x3$ Gauss kernel for the $3D$ case is the following:

$$Kernel_1 = \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix} \tag{2.7}$$

$$Kernel_2 = \begin{bmatrix} 2 & 4 & 2 \\ 4 & 8 & 4 \\ 2 & 4 & 2 \end{bmatrix} \tag{2.8}$$

$$Kernel_3 = \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix} \tag{2.9}$$

The (2.7), (2.8) and (2.9) together compose a $3D$ kernel of the Gauss low-pass filter, which can also be decomposed in a similar way as the one from the $2D$ case. Normalization value here is $\frac{1}{64}$.

Gradient - Sobel filter

Sobel filter or Sobel-Feldman operator, which is named after Irwin Sobel and Gary Feldman, is an image gradient operator that computes an approximation of the gradient of the image intensity. Let us first look at the $2D$

example of the $3x3$ operator:

$$G_x = \begin{bmatrix} +1 & 0 & -1 \\ +2 & 0 & -2 \\ +1 & 0 & -1 \end{bmatrix} * A_{img} \tag{2.10}$$

$$G_y = \begin{bmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} * A_{img} \tag{2.11}$$

The above kernels in (2.10) and (2.11) can be decomposed into a product of averaging and differentiation kernels as follows:

$$G_x = \begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix} * (\begin{bmatrix} +1 & 0 & -1 \end{bmatrix} * A_{img}) \tag{2.12}$$

$$G_y = \begin{bmatrix} +1 \\ 0 \\ -1 \end{bmatrix} * (\begin{bmatrix} 1 & 2 & 1 \end{bmatrix} * A_{img}) \tag{2.13}$$

Now that we have a definition for the $2D$ case, we can also present the $3D$ case, which is a lot more useful for us since we are dealing with $3D$ data. We now need to compute the gradient in Z direction as well. As an example, we will provide the $3D$ kernel for the differentiation in the Z direction:

$$Kernel_{z1} = \begin{bmatrix} -1 & -2 & -1 \\ -2 & -4 & -2 \\ -1 & -2 & -1 \end{bmatrix} \tag{2.14}$$

$$Kernel_{z2} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \tag{2.15}$$

$$Kernel_{z3} = \begin{bmatrix} +1 & +2 & +1 \\ +2 & +4 & +2 \\ +1 & +2 & +1 \end{bmatrix} \tag{2.16}$$

2.1.2 Imaging technology

X-Ray imaging

X-ray images are produced by X-ray quanta, which are absorbed or scattered, while travelling through the body. Different body parts absorb different amounts of X-ray quanta. Absorbing depends on the thickness and density of the object. For instance, bones absorb the most and thus appear white on the image, while hair's absorption of X-ray quanta is low and therefore appears black on the image. The X-ray quanta are generated by accelerating the electrons through the electric field and then colliding them with the target material. The collision produces heat and X-ray quanta. We can divide the X-ray quanta into two categories: hard beam X-rays and soft beam X-rays. The higher voltage of the electric field generates hard beam X-rays and the lower voltage the soft beam X-rays. Hard beam X-rays are mostly absorbed in the bones and are thus used for visualizing bone structures. Soft beam X-rays are absorbed in soft tissue. This makes them great for visualizing soft tissue, but it also makes them more dangerous to the patient, because this results in more energy absorption by the body, which results in more tissue damage.

Computed tomography (CT)

CT data is a volumetric data that consists of multiple X-ray images. Emitter and receiver rotate around the object, which is placed on a moving table, producing a slice of the observed object. After the slice is acquired, the table moves forward, so that the next slice can be taken. This is repeated until the entire object is scanned. X-ray CT offers better localization of anatomical structures than regular X-ray, because it records data for small volume elements and doesn't aggregate values of many tissues. It also provides better sensitivity and can thus better discriminate between different types of soft tissue. CT image data can be defined by the number of pixels of a slice and number of slices. For an example, a $256x256x200$ CT data has 200 slices each

having a resolution of $256x256$ pixels. Pixels or grid points hold intensity values produced by the CT, which represent tissue densities of the scanned object. Their dynamic range is 12 bit, which is packed into 16 bits or 2 bytes. These values are normalized with the equation (2.17). After normalization the values are measured in Hounsfield units (HU). HU of air is -1000 and HU of water is 0.

$$HU = \frac{\mu - \mu_{H_2O}}{\mu_{H_2O}} \cdot 1000 \tag{2.17}$$

Magnetic resonance imaging (MRI)

The MRI system consists of a powerful magnet, radio wave emitter, gradient and a computer. The human body is made up of around 70% of water. Hydrogen atoms act as small magnets and are affected by the magnet in the MRI system. The magnet produces a strong magnetic field, which is adjusted by the gradient in order to isolate the region of interest. Protons in the hydrogen atom align either parallel or anti-parallel with the magnetic field. The radio waves are applied orthogonal to the magnetic field and force the protons to change their orientation by either 90 degrees or 180 degrees. The radio signal is then terminated and the protons release energy and return to their previous (natural) orientation in the magnetic field. This relaxation period is different for different tissue types and it is detected by the system and used for visualization.

2.2 Transfer functions

Transfer functions are functions that map image data to optical characteristics, such as colour and opacity. In its simplest form, the transfer function can be described as a 1D function of volume scalar values, that assigns to each of them an opacity. This is an opacity transfer function (OTF). Colour transfer function (CTF) works in a similar fashion, but instead of opacity,

assigns colour to volume scalar values. Colour can be in a HSV format (hue, saturation and value) or RGB format (red, green, blue).

Transfer function

Expand on Transfer function theory description. Chp. 2.6 (at least 2 pages, if diagram present 3).

Transfer function description of implementation. -¿ check the files, open the program, take a few shots of the menu. (4 pages).

Show examples of different TF. -¿ draw a few TFs, show the result. (at least 3-6 image examples)

Overall the topic of TF should increase the thesis by circa 10 pages.

—

Notes for theory: The intensity values represent the domain, the visual properties represent the range of the TF. In the medical field, color, opacity and shades of gray are usually of the primary concern.

1d help separate materials, their density, while 2d (where the 2nd dim is gradient, help emphasize also the material boundaries)

1d TF specification is a two step process - Classification - materials are classified, as in bones, skin tissue etc. Assignment - color values and opacities are assigned to the classified materials.

Classification is done by assigning shapes to the intensity values, for example a trapezoid, where the slope represents the degree of uncertainty, analogical to the fuzzy logic.

Assignment of visual properties is then also done in the editor and it is assignment of colors and opacities to the shapes created in the first step.

The optimal solution would be automatic classification and the user would then just need to choose the visual properties of his or her liking, but due to the degree of variation in the data, that is nearly impossible and most techniques are semi-automatic, which means the user specifies some parameters of the classification algorithm and it does the rest.

Opacity TF One of the most important functions in the TF family is the Opacity TF. Areas (materials) with higher importances are assigned higher

opacities, those of lower importance, lower opacities. In this way we prevent materials of lower importance obscuring the desired materials. The problem with 'creating' the suitable TF is, that the parameter space is large and predicting how the visualization of the volume will change with changing the values in the TF editor is very difficult..

Histogram Histogram viewer is one of the most useful tools to the TF designer since it provides the view into the data. A few different approaches are used to refine the visualization of the data: - zooming - logarithmic scales - usage of statistical values such as: std, average, minimum, maximum ...

Different approaches to 1D TF design unassisted - as the name implies, the designer is on his own, no guidance is provided. Data driven - these are based on the analysis of the data - local clusters, visibility ... Image driven - support of analyzing parts of rendered images.

Interactive approaches are usually called also unassisted, but that's not entirely correct, since interactive approaches can also exploit data and image driven support.

Overview of unassisted design The core concept of the unassisted design is the TF editor. With this editor the user can define the shape and visual characteristics of the TF.

In clinical practice the editor is too complex and cumbersome, a simpler, more common approach is called windowing. The user clicks the right mouse button and then with vertical and horizontal movements sets the center and the width of the opacity ramp. This has the advantage that the user does not need to switch his focus between the editor and the image. The disadvantage of this technique is that it can only produce relatively simple TF.

In medical research TF editor is a necessity, since more control is required. Thus the user should at least be able to add, remove, reposition the points that define the TF.

As already stated, creating TF is a difficult process, that's why the editors usually provide some predefined TFs that the users can easily load. Then there are other approaches that aim to simplify the creation and try

to minimize the direct control of the TF via the editor, where the user needs to work in the histogram space, which might not at all be intuitive. One of them is stroke based TF. The technique proves itself quite useful, when the user needs to categorize an object that has a very small footprint and cannot be confidently categorized in the editor. The user draws strokes that act as the boundaries of the region that should be emphasized.

Overview of data driven support / design We will provide a brief overview of data driven support, but we won't delve into too much detail, since none of these approaches were implemented by us.

Clustering

Clustering is a category of unsupervised learning, where elements are grouped together into clusters based on the similarity between them. These metrics are applied to actual individual data or the lower tier clusters. We can judge the similarity of voxels or feature vectors, which can be composed of: intensity, gradient magnitude, etc. Important thing to note is, that for volume clustering, also the position of the voxel plays an important role and not just its attributes.

Clustering Algorithms:

K-Means In K-Means the initial number of clusters must be given. K-Means first randomly selects the initial centroids from the elements. The algorithm then goes over all the elements and assigns them to the nearest centroid based on the given similarity metric. Then the centroids of clusters are recomputed based on some given compactness function and after the procedure is repeated until convergence, which is some non-reproducible local optimum. The result is heavily influenced by initial centroid selection.

K-Means++ K-Means++ is similar to K-means, but it is less sensitive on the initial centroid selection. The algorithm is nearly the same, the only difference is in the initial centroid selection. The original K-Means++ chooses the first centroid uniformly at random from the given elements. The next centroid is then chosen from the remaining elements based on the point's probability which is proportional to its squared distance from the closest al-

ready existing centroid point). We repeat this process until we obtain the required number of centroids (required number is k).

In a sense we want to spread the initial clusters as much as possible.

DBSCAN, OPTICS One limitation of K-Means is that it can only produce convex shapes of clusters, which might prove limiting at times. Fortunately, with Density-based clustering (DBSCAN) and Ordering Points To Identify the Clustering Structure (OPTICS) we are able to produce any kind of clustering shape. Instead of the number of clusters, these two algorithms require two inputs, one is the minimal number of points the clusters ought to have and the other, the maximum distance to consider for a cluster (epsilon).

Hierarchical clustering The basic strategy is as follows: Each element is its own cluster. The two most similar clusters based on the similarity metric, are grouped together. Repeat until only one cluster remains.

Two simple methods for the second step are: single linkage and complete linkage. The first one considers the closest pair of elements, the latter the maximum distance between a pair of items.

Clustering different results Clustering algorithms return different types of results - hierarchical returns a nested n-tier clustering tree and on the other hand, partitioning algorithms return a single partitioning of the data. Also important note is, that elements can be assigned to the clusters directly, or with weights, in which case we are talking about fuzzy clustering algorithms.

2.3 Direct volume visualization

We have two major categories for volume visualization: direct volume visualization and indirect volume visualization or surface rendering. Surface rendering creates an intermediate representation of data, which is a surface that can be rendered. Direct volume visualization or direct volume rendering (DVR) directly presents the data (without intermediate representation), which is done by projecting the data information onto the viewing plane. Transport theory of light (modeled by linear Boltzmann equation in the sta-

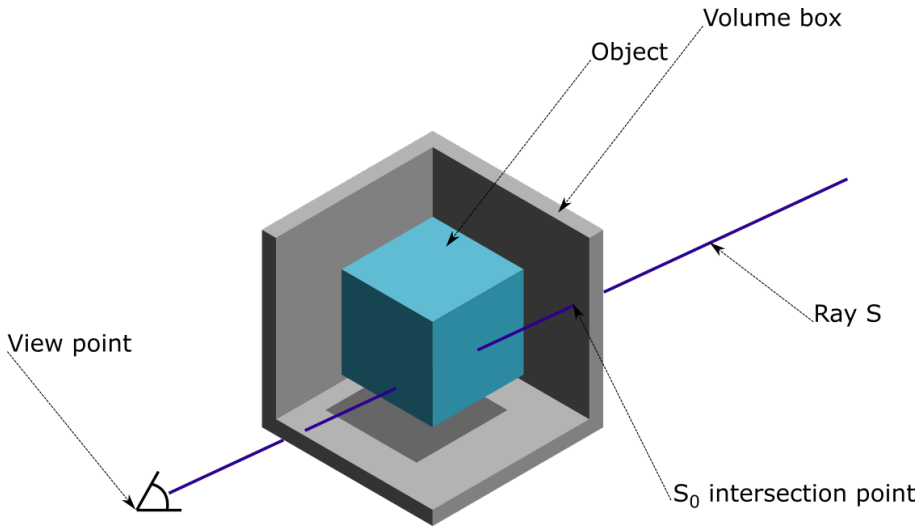

Figure 2.3: Ray passing through the volume data.

tionary state) describes the interaction of light with objects and the medium in between. These interactions are emission, absorption, scattering. In practice this complex physical model is simplified - we assume that the influence of changing medium is none and we do not model scattering and diffraction nor different wavelenghts. If we assume a low albedo scenario, we can further remove refraction and reflection and end up with the density emitter model, which only takes into account emission and absorption. Although there is still a diffuse lighting component, which is usually approximated with the Phong model. In a sense every contributing particle is modeled as a tiny light source, whose light is emitted and attenuated by other particles until it reaches the viewing plane.

2.3.1 The density emitter model

As it was said in the paragraph above, the density emitter model only takes into account emission and absorption.

Emission

Let us first explain the emission only model, afterwards we will explain the absorption only model and then we will combine the two to arrive at the volume rendering equation.
As you can see from the graphic 2.3, ray S enters the volume at point S_0 and traverses the volume particles until it exits the volume and reaches the view point. As the ray passes through the volume it accumulates the light that the particles are emitting. The light source is modeled by the source term $Q_\lambda(s)$, where λ specifies the wavelength and s the direction of light or ray. We simplify the source term by modeling it for only one fixed wavelength $Q_\lambda(s) \longrightarrow Q(s)$. Differential equation (2.18) tells us how intensity $I(s)$ changes at point s along a ray S, based on the source term $Q(s)$.

$$\frac{\partial I}{\partial s} = Q(s) \tag{2.18}$$

with a solution

$$I(s) = I_{S_0} + \int_{S_0}^{S} Q(t)dt \tag{2.19}$$

In the solution (2.19) to the differential equation (2.18) we have the intensity term $I(s)$ at point s on the ray S, which is computed as a sum between the intensity at initial entrance point (can be a default background colour) I_{S_0} and the integral of source terms $Q(t)$ along the ray S in the volume. The source term $Q(s)$ is usually obtained by applying the transfer function to the sampled value from the volume data set at position s. Transfer functions are described in more detail in 2.2. A possible extension to the basic density-emitter model is to also illuminate the source term $Q(s)$ by the lighting model, such as the Phong lighting model.

Absorption

Now that we defined the emission only model, let us take a look at the absorption only model, which assumes that light is only attenuated by the particles present in the volume. Attenuation factor τ (or also called the

extincition coefficient) models the differential density of the material. It attenuates or reduces light intensity of the ray S as it passes through the volume. The attenuation factor $\tau(s)$ is obtained by applying the opacity transfer function on the sampled value at position s. Differential equation (2.20) defines how the light intensity $I(s)$ is attenuated by the attenuation factor $\tau(s)$.

$$\frac{\partial I}{\partial s} = -\tau(s) \cdot I(s) \tag{2.20}$$

with the following solution

$$I(s) = I_{S_0} \cdot e^{-\int_{S_0}^{S} \tau(t)dt} \tag{2.21}$$

where the so called accumulated transparency is $T_{S_0}(s) = e^{-\int_{S_0}^{S} \tau(t)dt}$, therefore we can rewrite equation (2.21) as

$$I(s) = I_{S_0} \cdot T_{S_0}(s) \tag{2.22}$$

Accumulated transparency T_{S_0} is higher, if the attenuation factor τ is lower and vice versa, if τ is higher, the T_{S_0} is lower, which results in a lot less initial light intensity traversing through the volume.

Volume rendering equation

Now let us combine the emission only and absorption only models.

$$\frac{\partial I}{\partial s} = Q(s) - \tau(s) \cdot I(s) \tag{2.23}$$

with a solution

$$I(s) = I_{S_0} \cdot e^{-\int_{S_0}^{S} \tau(t)dt} + \int_{S_0}^{S} Q(p) \cdot e^{-\int_{S_0}^{S} \tau(t)dt} dp = I_{S_0} \cdot T_{S_0}(s) + \int_{S_0}^{S} Q(p) \cdot T_p(p) dp \tag{2.24}$$

Since (2.24) cannot be solved analytically, we need to discretize it (we can see this in 2.4) and solve it using the Riemann sum. Also, since in the discrete solution our entry point S_0 starts at 0, we can simplify the attenuation function T_x to $T_{S_0} \longrightarrow T_0$. Our volume data exit point has a value of $n - 1$.

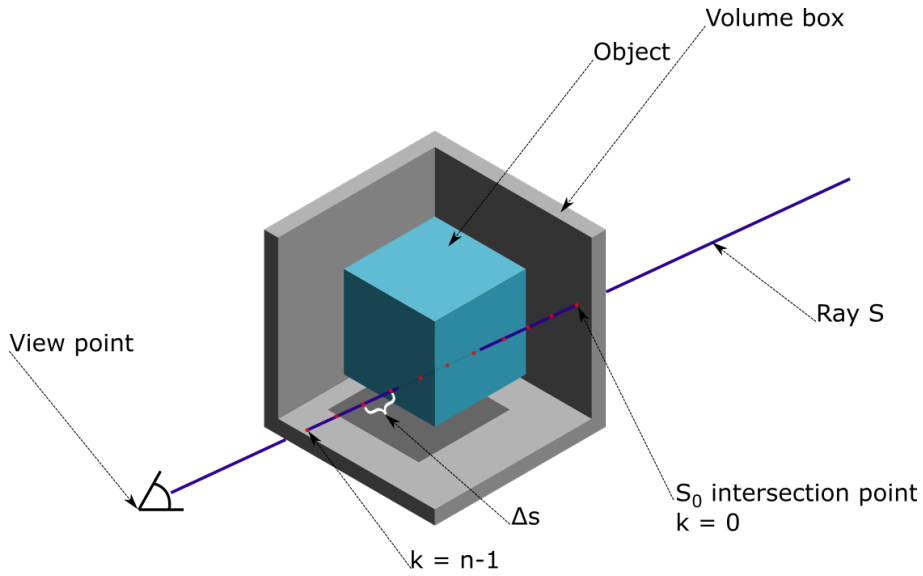

Figure 2.4: Ray passing through the volume data (discretized).

We sample the volume data at lengths Δs, which is used to move our sample point k as $k \cdot \Delta s$. The source term $Q(s)$ becomes $Q_k = Q(s_k)$. In the (2.25) we first discretize the attenuation function and in the (2.26) we have our final discrete version of direct volume rendering equation.

$$T_0(s) = e^{-\int_{S_0}^{S} \tau(t)dt} = e^{-\sum_{k=0}^{n-1} \tau(k\cdot\Delta t)\Delta t} = \prod_{k=0}^{n-1} e^{-\tau(k\cdot\Delta t)\Delta t} = \prod_{k=0}^{n-1} t_k \tag{2.25}$$

$$I(s) = I_0 \cdot \prod_{k=0}^{n-1} t_k + \sum_{k=0}^{n-1} Q(k \cdot \Delta s) \cdot \Delta s \prod_{j=k+1}^{n-1} t_j \tag{2.26}$$

We have used a convention, where we move along the ray from the back to the front. If we were to switch the direction, the boundaries on the integral (2.24), would need to be switched.

2.3.2 The pipeline

The pipeline consists of sampling, classification and illumination and finally compositing. We are only computing values at discrete steps. There are two

important factors that affect sampling. The first one is the Δs. If the value is smaller, we take more samples and increase the quality of the image at a cost of more computations. If the value is higher, we take less samples and we decrease the amount of computations, which might lead to visual artifacts in the image. The second factor is the interpolation function. The data is only defined at the grid locations, therefore we have to combine multiple values from the neighbourhood. One way to achieve this is to use a third order tricubic interpolation function, which is essentially a low pass filter. Once the sampling point is determined, we get its actual (source term) value from the transfer function. This is called classification and the result is $Q_d(s)$, or a colour value. The obtained colour value $Q_d(s)$ is then shaded. Shading is usually modelled with the Phong model, which simulates the ambient, diffuse and specular light. In the (2.27) the point's light contribution is computed. The light contribution of the sampled point is affected by the opacity at that sampled point - the higher it is, the higher the contribution. Opacity is obtained from the OTF, which returns the attenuation factor $\tau(s)$ (or in discrete situation t_j). Compositing is then the accumulation of values along the ray S.

$$\begin{aligned} Q(s) &= light_{ambient} + light_{diffuse} + light_{specular} = \\ &= I_a k_a O_d(s) + I_d k_d O_d(s)(N(s) \cdot L) + I_s k_s (V \cdot R)^n \end{aligned} \quad (2.27)$$

In (2.27) the k_a, k_d and k_s are reflection coefficients, I_a is light intensity, I_d and I_s are light source diffuse and specular components. $O_d(s)$ is object's diffuse colour, $N(s)$ is the surface normal or volume gradient at position s, L is the direction of the light in the environment. V and R vectors represent viewer's direction and the direction of the specular reflection.

Preclassified volume rendering pipeline

In contrast to the postclassified volume rendering pipeline in 2.3.2, in the preclassified volume rendering pipeline, the sampling stage comes after the classification and illumination stage. The major benefits of this particular

method are: expensive illumination can be done in the preprocessing stage (at a cost of additional memory space), it can be used, when illumination cannot be computed in real-time and you know in advance, which voxels will contribute to the final image. A few drawbacks of the method are visual artifacts, which can be present in the final image. These are colour bleeding (red colour shift), staircasing and aliasing. They can occur because of change of sampling domain from data to colour values and high frequency details in the transfer function.

2.3.3 Compositing

Compositing focuses on the accumulation of the sampling point's contributions. Overoperator (an example with three sampling points and $C_k = Q(k \cdot \Delta s)$ can be seen in (2.28)) is a composition operator, which is associative, therefore we can change the order of evaluation. These leads to two compositing variants: front-to-back and back-to-front.

$$\begin{aligned} I(s) = I_0 \cdot \prod_{k=0}^{2} t_k + \prod_{k=0}^{2} C_k \cdot \prod_{j=k+1}^{2} t_j = \\ = I_0 \cdot (t_0 \cdot t_1 \cdot t_2) + C_0 \cdot (t_1 \cdot t_2) + C_1 \cdot t_2 + C_2 = \\ = C_2 + t_2 \cdot (C_1 + t_1 \cdot (C_0 + I_0 \cdot t_0)) = \\ = C_2 \; over (C_1 \; over (C_0 \; over \; I_0)) \end{aligned} \quad (2.28)$$

Front-to-back composes samples on the ray from the front of the volume data to the back of the volume data. We start our compositing at sample $k = n - 1$ and progress to $k = 0$. Intensity I_{k-1} is computed by using the current intensity I_k and the current colour C_k, which is attenuated by the

current attenuation factor $\hat{t}_k$. The formula is

$$\begin{aligned} I_{n-1} &= C_{n-1} \\ \hat{t}_{n-1} &= t_{n-1} \\ I_{k-1} &= I_k + C_k \cdot \hat{t}_k, \forall k = n-2, ..., 0 \end{aligned} \quad (2.29)$$

We also need to explicitly update the accumulated attenuation factor

$$\hat{t}_{k-1} = \hat{t}_k \cdot t_k, \forall k = n-2, ..., 0 \quad (2.30)$$

The accumulated end result is stored in $\hat{t}_{k_0}$ and I_0.

Back-to-front composes samples on the ray from the back of the volume data to the front of the volume data. The initial value I_0 can be the background colour. Then the current intensity I_k is computed as

$$I_k = I_{k-1} \cdot t_k + C_k, \forall k = 1, ..., n-1 \quad (2.31)$$

First hit

In first hit compositing we are sampling rays until two samples above and below threshold are found. This reconstructs or approximates the isosurface. If our sample rate is too low, we might experience visual artifacts.

Pseudo X-Ray or averaging

The method averages all the sampled values across the ray. The image looks like an x-ray image, because colour and opacity are ignored. If threshold sensitive compositing is not used, we use all the values and because there can be a lot of empty space in the volume data, we might end up with a darker image than expected. In threshold sensitive compositing we use only the values that are above a certain threshold. Comparison between the pseudo x-ray and threshold sensitive compositing can be seen in 2.5 and 2.6.

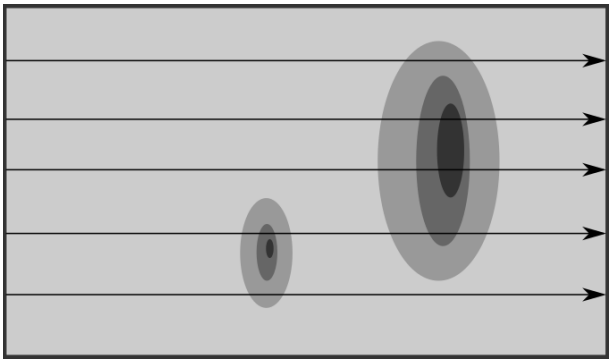

Figure 2.5: Pseudo X-Ray compositing.

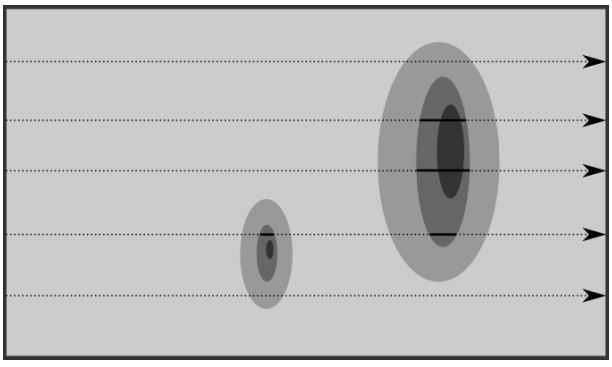

Figure 2.6: Threshold sensitive compositing.

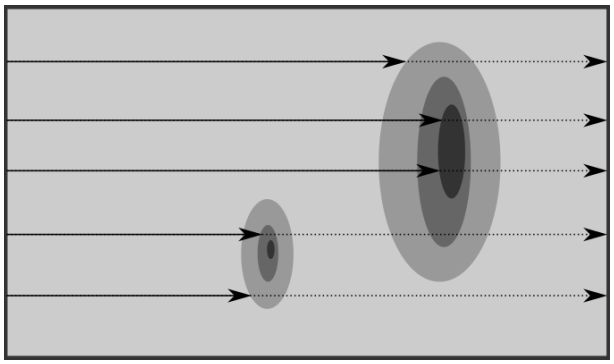

Figure 2.7: MIP compositing method.

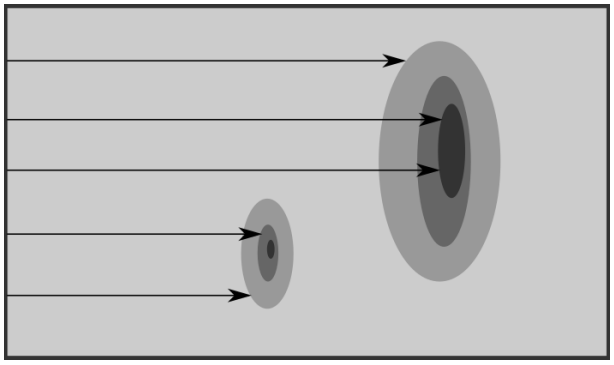

Figure 2.8: CVP compositing method.

Maximum intensity projection (MIP)

It traverses the whole length of the ray, which is time consuming, in order to determine the maximum intensity value. It is excellent for visualizing bone structures and contrast enhanced vascular structures. There a few problems though. The resulting image can be the same from the front or the back view, because no attenuation factor is used. Bright structures can outshone the smaller or less bright structures. To solve this issue the closest vessel projection can be used. In this compositing method, the ray takes the first sample above a specified threshold. The comparison between the two can be seen in 2.7 and in 2.8.

Maximum intensity difference accumulation (MIDA)

MIDA is used to solve the issue of the lack of depth perception in MIP. MIDA combines the standard DVR and MIP. It keeps track of every new maximum encountered along the ray. Compositing and also shading is done at each of these points.

2.3.4 Preintegrated volume rendering

As we have said, if our sample rate is not high enough, we can end up with visual artifacts or anomalies. Instead of compositing values at sample points, we can rather try compositing ray segments between sampling points. We assume linear variation of intensity and because we have the same stride length Δs, the value depends only on τ_k, τ_{k+1} and C_k, C_{k+1}. We can precompute a $2D$ lookup table for all k and $k+1$ combinations. At 8 bit depth the table dimensions are $256\,x\,256$.

2.3.5 Raycasting

Raycasting is a computationally expensive problem, since a ray needs to be cast through the volume, sampled, classified, illuminated and accumulated for every pixel of the image. Fortunately, the problem is highly parallelizable and can be efficiently implemented on the GPU. There are a few ways to optimize raycasting. With early ray termination we terminate the ray, if opacity reached a certain threshold (other structures that remain will contribute an insignificant amount). We can also employ a method of empty space skipping, because most volume datasets are sparse. One way is to use subvolumes with flags, which tell us, if the there is any useful data present in the subvolume. The subvolume technique can be further improved by hierarchical data structure, such as an octree.

2.4 Ray marching

2.5 Cinematic rendering (path tracer)

2.6 Requirements and specification

2.7 Algorithms and data structures

2.7.1 Object order empty space skipping

The ray marcher samples the whole volume at fixed delta steps. This can be very expensive, especially when we sample an empty region, since we have to march through the whole volume before we terminate the sampling.

The idea behind the implementation mentioned in the article [20] is to construct the bounding geometry, which is then used as a starting point and end point of the ray marching algorithm, thus avoiding sampling through most of the empty space regions. How efficient is this approach largely depends on the distribution (locality of the data) - volumetric cloud data can happen to be distributed across the whole volume and thus not much performance can be gained, while CT data is (usually) more localized, so this approach can lead to better results.

Another potential performance issue that does occur in object order empty space skipping, is that rays which travel (sample) near the actual object tend to traverse a lot of empty space. This affect can be observed in figure 2.9 (the top most ray in the figure) - so while other rays in the figure intersect with the cyan volume and can thus stop marching, the top most ray needs to march through the whole bounded volume despite the fact it only samples empty space. This effect however only occurs for small number of rays and can be mitigated by combining the object order skipping with image-order empty space skipping (for example in ray casting fragment shader to either skip individual samples or advance the sample position along the ray by some samples).

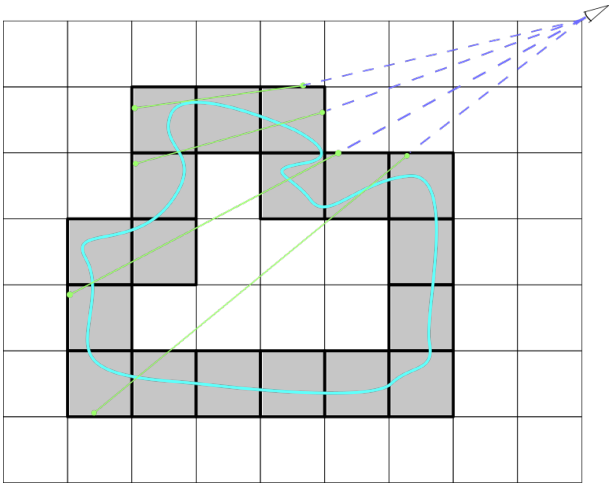

Figure 2.9: Simple sketch showing the ray start and end point in the bounding volume. The green lines are the actual rays that sample the volume. The bold grid is the bounding volume constructed by using the active grid buffer. The cyan coloured line is the 'data'.

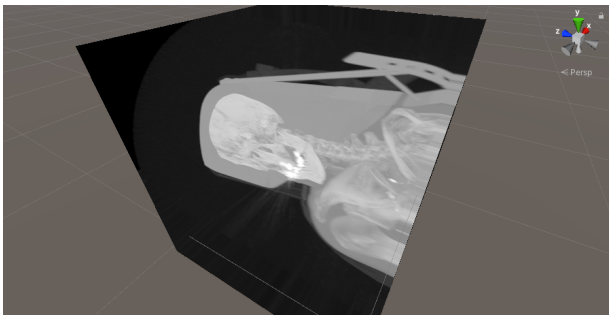

Figure 2.10: Whole volume march - bounding box is the size of the whole volume.

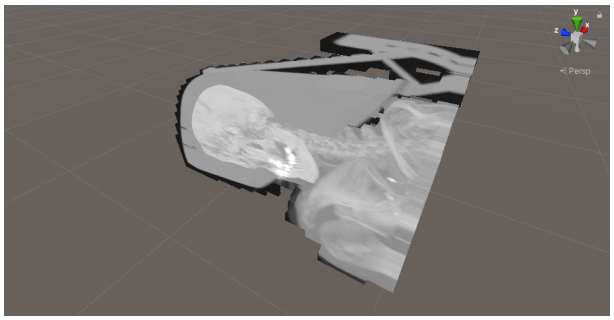

Figure 2.11: With Using the bounding box.

The following procedure describes how to compute the active grid, construct the bounding box and finally get the ray start and end positions:

1. Start with the original volume texture with ISO values and dimensions of (for example) $256x256x128$.

2. Create a grid (active grid) of lower dimensions (more coarse). A grid where a cell spans $8x8x8$ voxels from the original volume texture, which results in a grid with dimensions $32x32x16$.

3. Computing the data in the active grid:

   (a) The cells in this grid hold the maximum ISO value (found in the $8x8x8$ region) that was computed from the application of transfer function on the original volume ISO value.

   (b) The cell in this active grid is considered active if its 'mapped' (the value after applying the transfer function) ISO value is larger than given threshold.

4. Creating the bounding box:

   (a) First we create as many vertices as there are active grid cells. These are then sent of to the vertex shader.

   (b) In the shader, in the geometry pass, we use the active grid buffer to check if the vertex is in the active or non-active cell in the

```
(!(center_intensity >= iso && neighbor_intensity < iso)) return;
```

Figure 2.12: Insure we only build quads at border regions.

active grid. For vertices that are in the active cell, we check the neighbouring cells and if the neighbouring cell is inactive, only then we construct a quad - our goal is to construct the bounding volume, thus we only construct quads at the borders of the active volume. We have to check 6 sides at the current position, since if all neighbouring cells are inactive, we have to draw a quad on each side, in essence creating a cube.

(c) Let's examine the generation of the right side of the cube. All others follow the same procedure, but with a different normal (to check the neighbouring cell) and offset vector (to create the necessary vertices).

i. We first offset the position of the grid cell by the given normal, for this particular example, by normal $\vec{n} = \begin{bmatrix} +1 \\ 0 \\ 0 \end{bmatrix}$. With this normal we obtain the next grid position along the X axis: $neighbour_{position} = current_{position} + \vec{n}$.

ii. We then obtain the intensity of the neighbour and current cell and check the condition 2.12, which insures we only build the quads at border regions.

iii. After the condition check, we must construct the vertices that will form the basis of the two triangles that form the quad of the cube, the quad can be seen in figure 2.13. First we compute center of the quad: $c_{position} = current_{position} + 0.5 * \vec{n}$.

iv. Then we must compute the 4 vertices: V_0, V_1, V_2, V_3. These four vertices are computed by offsetting the center of the quad by the given offset. For this particular example the offset is

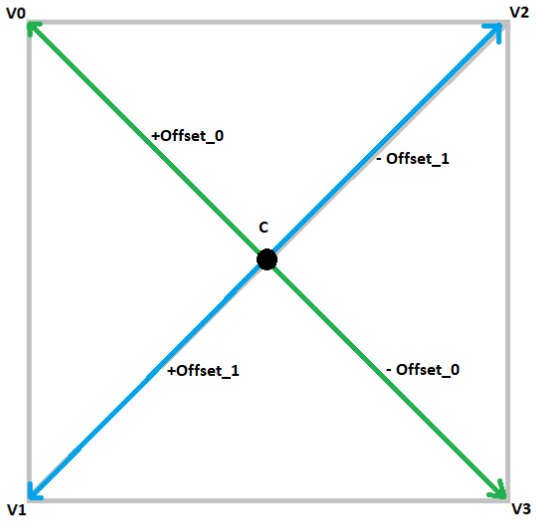

Figure 2.13: Computing the vertices example.

$\vec{o_0} = \begin{bmatrix} 0 \\ 0.5 \\ 0.5 \end{bmatrix}$. That only allows us to offset in one direction. We need to compute $\vec{o_1}$ as well, which we obtain by computing the cross product: $\vec{o_1} = cross(\vec{n}, \vec{o_0})$. Now we can construct all 4 vertices that form our quad. In the geometry shader all that is left to do, is to emit them in order for the gpu to construct the polygons:

$$\begin{aligned} \vec{v_0} &= c_{position} + \vec{o_0} \\ \vec{v_1} &= c_{position} + \vec{o_1} \\ \vec{v_2} &= c_{position} - \vec{o_1} \\ \vec{v_3} &= c_{position} - \vec{o_0} \end{aligned} \tag{2.32}$$

5. For completeness, we list here the normals and offsets for all of the

quads 2.33.

$$
\begin{aligned}
Axis_X \longrightarrow \vec{n_+} = \begin{bmatrix} +1 \\ 0 \\ 0 \end{bmatrix} \vec{o_+} = \begin{bmatrix} 0 \\ 0.5 \\ 0.5 \end{bmatrix} \vec{n_-} = \begin{bmatrix} -1 \\ 0 \\ 0 \end{bmatrix} \vec{o_-} = \begin{bmatrix} 0 \\ 0.5 \\ 0.5 \end{bmatrix} \\
Axis_Y \longrightarrow \vec{n_+} = \begin{bmatrix} 0 \\ +1 \\ 0 \end{bmatrix} \vec{o_+} = \begin{bmatrix} 0.5 \\ 0 \\ 0.5 \end{bmatrix} \vec{n_-} = \begin{bmatrix} 0 \\ -1 \\ 0 \end{bmatrix} \vec{o_-} = \begin{bmatrix} 0.5 \\ 0 \\ 0.5 \end{bmatrix} \\
Axis_Z \longrightarrow \vec{n_+} = \begin{bmatrix} 0 \\ 0 \\ +1 \end{bmatrix} \vec{o_+} = \begin{bmatrix} 0.5 \\ 0.5 \\ 0 \end{bmatrix} \vec{n_-} = \begin{bmatrix} 0 \\ 0 \\ -1 \end{bmatrix} \vec{o_-} = \begin{bmatrix} 0.5 \\ 0.5 \\ 0 \end{bmatrix}
\end{aligned}
\tag{2.33}
$$

6. Obtaining the start and end positions:

   (a) todo

   (b) todo

- Then we render this newly created bounding box with multiple passes: - we render the backfaces and store the depths. - we render the frontfaces and store the depths. - In our volume march program we then render the structure again and compute the start and end positions of our ray marching from the stored depth values obtained in the previous procedure.

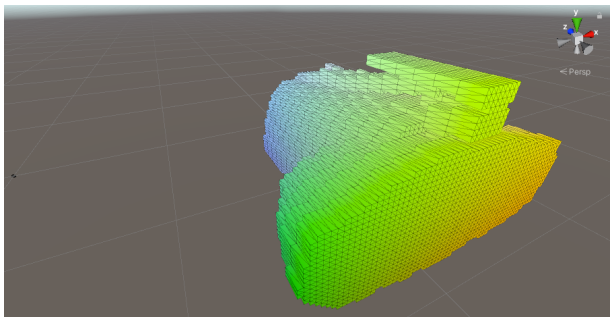

Figure 2.14: Bounding box front faces.

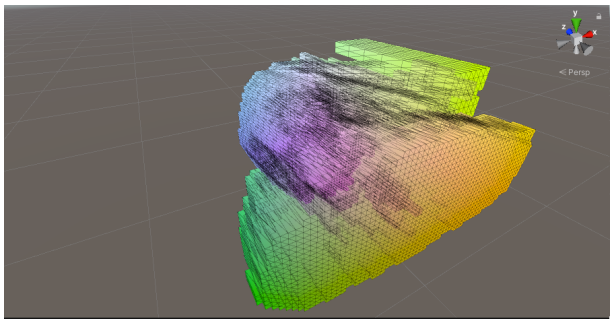

Figure 2.15: Bounding box back faces.

Chapter 3

Implementation

3.1 Architecture

3.1.1 Overview

The application architecture is very simple and designed as a prototype and does not follow any of the software design principles (such as SOLID, patterns, higher abstraction of more stable entities etc.). It consists of multiple classes that focus on providing one service that is then used by the client.

In general the application can be divided into the following parts of the functionality they provide:

1. Main menu (data loader): loading and saving of processed data.
2. Object manipulator: provides rotation, scaling and transforming of the volume box.
3. Miscellaneous tools: slicer (slices the volume box), 3D 'map', histogram.
4. Transfer function drawer: a tool for drawing, loading and storing of transfer functions.
5. Shader manipulator: exposes different shader functionalities (different lighting modes, shading modes, desired sample size etc.).

6. Shader: the drawer of the volume data.

3.1.2 Application flow

The standard flow would be as follows, user first launches the application where he or she is greeted by the main menu entry screen. There the user can either choose a new volume data (stored somewhere on the PC) that is supported by the application for processing or select an already processed data, which was stored during the previous processing. After the processing is done the user can proceed to the next screen, where he has the option to connect to the Microsoft HoloLens (uses a Microsoft library to connect to the HoloLens). If he chooses to connect, he gets notified, if the connection was established successfully. At last the user can enter the volume box render scene. In this scene the user is able to see the medical data, open various tool menus that provide all sorts of functionalities: slicing, histogram visualizations, volume object spatial transformations, transfer function drawing and shader settings.

3.1.3 Main menu data loader

Main menu provides the following functionalities:

1. data storing (loading and saving of processed data),
2. data processing (normalizing and filtering),
3. establishing a stream connection to Microsoft HoloLens.

The application uses SimpleITK library for loading the initial volume dataset and currently supports reading and converting NRRD, MRC, MINC and NIFTI to data buffer. This data buffer is then processed in the compute shaders, where we create the following processed datasets:

1. normalized volume data (values are between 0 and 1),
2. derivatives: computed with central differences method or Sobel filter,

3. Gaussian filtered normalized volume data and data with derivatives (computed with central differences method).

In total we calculate 5 datasets, which are then stored on the PC for future loading (thus this data processing step can be omitted in the future, if the user wishes to view the same volume dataset).

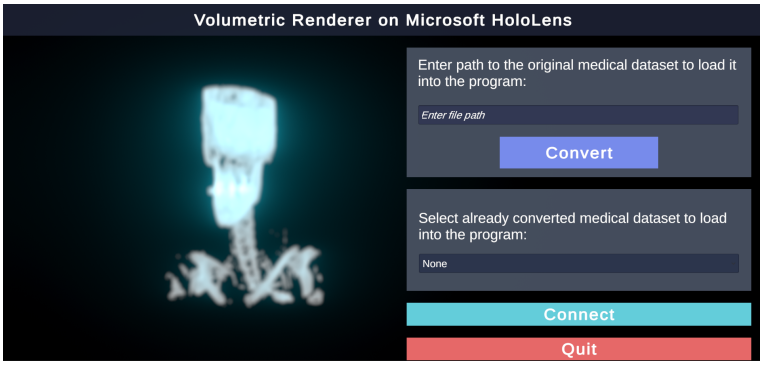

Figure 3.1: Main menu of the application.

Data storing

The data storing process - loading (deserialization) and saving (serialization) - is based on `SaveImageDataObject` class, which holds the necessary image attributes: read type, raw data array, min and max value, image dimensions, file path and the five datasets. These attributes are all primitive data types so that the class can be serialized and deserialization into the `ImageDataObject`, which holds data in Unity `Texture3D` data type. The copying between data buffers and Unity texture data types is done using compute shaders on the GPU.

User has two options in the main menu (seen in figure 3.1): they can either enter the file path of raw data (for example NRRD data) and convert it (process it) or they can enter the file path of the already processed data and load it into the application. The second approach is much faster, because no processing takes place - the deserialized image data object is serialized and cast to `SaveImageDataObject` type and then used to populate the fields of

the `ImageDataObject`, which is used throughout the application (an example of successful loading operation can be seen in figure 3.2). On the other hand, if the user enters the file path of the raw data, the raw data is first processed and after the processing is complete, it is also serialized in the location of the application persistent data path (the original file name is concatenated with *_image* tag).

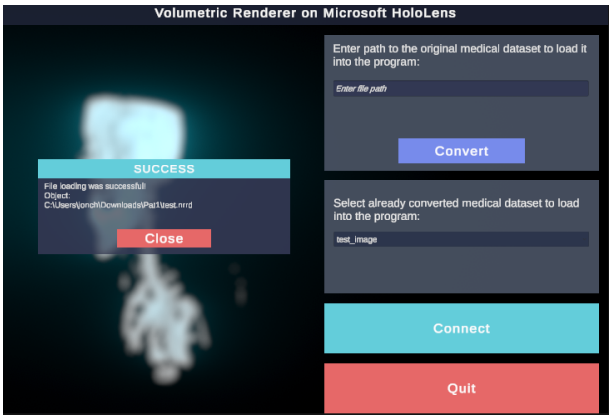

Figure 3.2: Example of successful serialized image data deserialization.

Data processing

Data processing encompasses the data normalization, gradient calculations and filtering. All processing is done within Unity compute shaders and thus paralelized on the GPU. **Derivatives computed by the central difference method** (2.1.1) and **normalization of the image intensity values** (2.1.1) are computed by the two kernels that can be found in the `ImageCalculator.compute`. The Derivative kernel uses two read write $3D$ textures - one as input and the other as output, while the normalization kernel has to use read write structured buffer of floats as input and then writes normalized data to $3D$ read write texture.
Sobel operator (2.1.1) and **Gaussian filtered gradients computed by the central difference method** (2.1.1) and **Gaussian filtered normalized data** are computed by multiple kernels for all of the three dimensions

- kernel for x, y and z dimension. There is also one additional special kernel for z direction which is used by the Sobel operator. Both Gaussian filter and Sobel operator use decomposed kernels in order to minimize the number of multiplications, but at the cost of multiple kernel dispatches (this is done by the `ImageDataObject`).

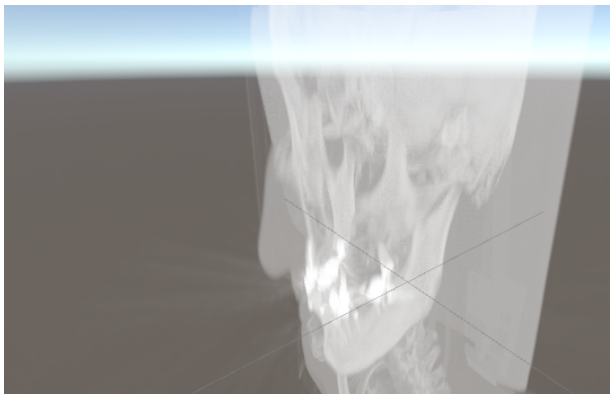

Figure 3.3: MIP image.

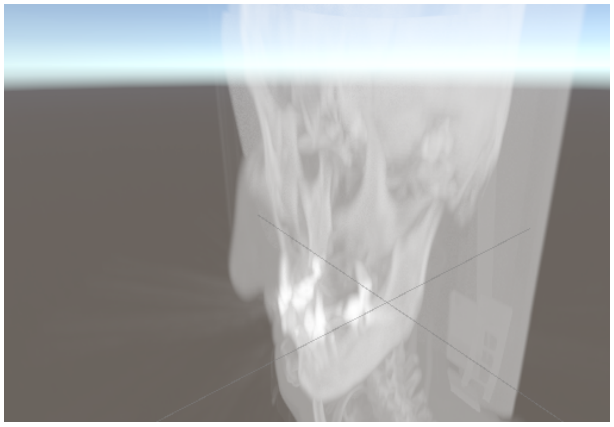

Figure 3.4: MIP image with applied Gaussian filter.

In figures 3.3 and 3.4 one can observe the normalized data compared to normalized data filtered by Gaussian filter. In figures 3.5, 3.6 and 3.7 we can see a comparison between different gradient calculations (central difference method, Sobel operator, Gaussian filtered central difference method),

when we use surface rendering (seen in figure 2.8) and Blinn-Phong reflection lighting model.

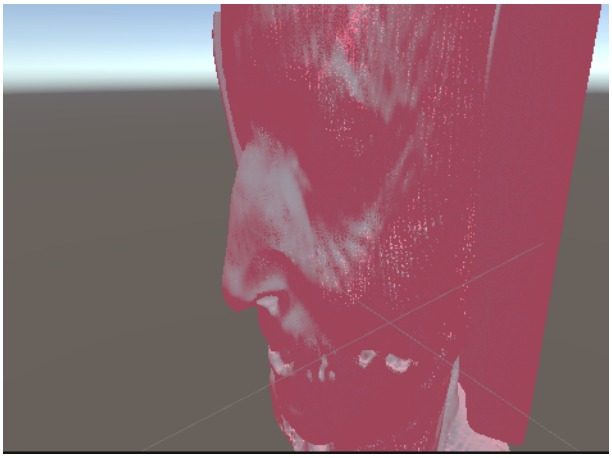

Figure 3.5: Gradient computed with central difference method.

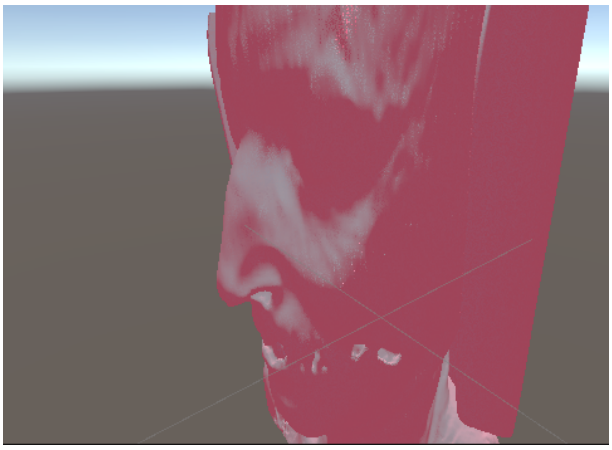

Figure 3.6: Gradient computed with Sobel operator.

At a first glance and in this relatively small dataset, the Gauss filtered gradients computed with the central difference method seem the least noisy and regular gradients with central difference method the most noise sensitive.

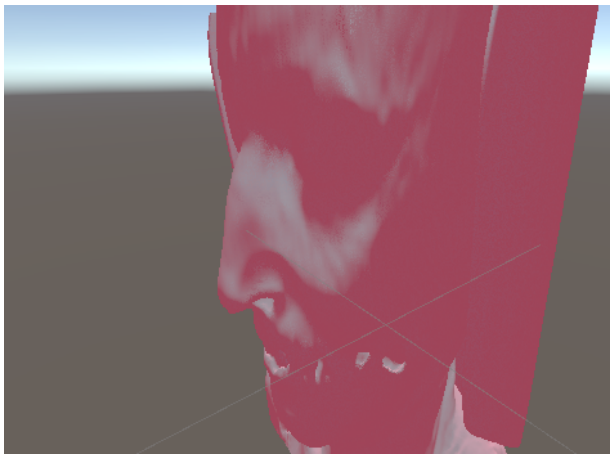

Figure 3.7: Central difference gradients with applied Gaussian filter.

Streaming

As can be seen from the figure 3.8, user can enter the `IP` address of the HoloLens and establish the connection. The application will inform the user when and if the connection was established successfully. It is important to note, that this connection step is optional and user is free to press the button `Run` and enter the main volume rendering scene. The streaming and connection establishing logic is delegated to the library `UnityEngine.XR.WSA`.

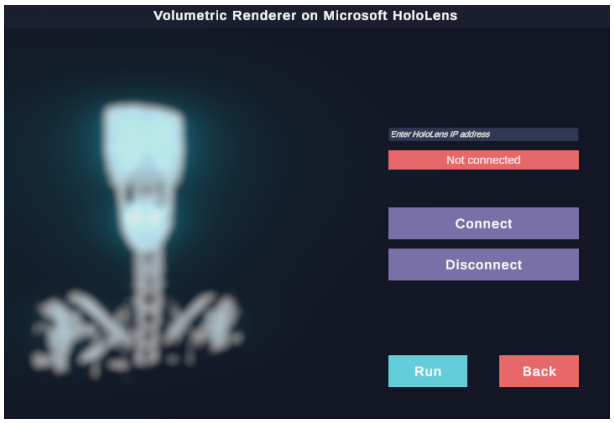

Figure 3.8: Streaming connection menu.

3.1.4 Shader manipulator

Describe the screen in detail and what it does and how.

3.1.5 Transfer function drawer

Describe what and how (also mention that it's compute shader based)

3.1.6 Histogram drawers

Describe what and why.

3.1.7 Shader

Where the magic happens. Insert images. - Maximum intensity projection - DVR blnding - Surface rendering - Volume path tracing (failed to obtain any results of value, provide some approaches and images perhaps, but dont dwell too much here since it's unsuccessful) - empty object space skipping

Chapter 4

Results

experience and questionaries in the next chapter4

Chapter 5

Interpretation and conclusion

Chapter 6

Further work

Appendix A

Title of the appendix 1

Example of the appendix.

Bibliography

[1] C. P. Botha, B. Preim, Introduction to visual computing for medicine, Morgan Kaufmann, San Francisco, United States, 2014.

[2] M. Hadwiger, J. M. Kniss, C. Rezk-salama, D. Weiskopf, K. Engel, Real-time Volume Graphics, A. K. Peters, Ltd., Natick, MA, USA, 2006.

[3] T. Sielhorst, M. Feuerstein, N. Navab, Advanced medical displays: A literature review of augmented reality, Journal of Display Technology 4 (4) (2008) 451–467.

[4] C. Gsaxner, A. Pepe, J. Wallner, D. Schmalstieg, J. Egger, Markerless image-to-face registration for untethered augmented reality in head and neck surgery, in: Proc. International Conference on Medical Image Computing and Computer-Assisted Intervention, Springer, 2019, p. to appear.

[5] L. Trestioreanu, Holographic visualisation of radiology data and automated machine learning-based medical image segmentation (2018). `arXiv:1808.04929`.

[6] M. Wieczorek, A. Aichert, O. Kutter, C. Bichlmeier, J. Landes, S. Heining, E. Euler, N. Navab, Gpu-accelerated rendering for medical augmented reality in minimally-invasive procedures, CEUR Workshop Proceedings 574 (07 2010).

[7] L. Chen, W. Tang, N. W. John, Real-time geometry-aware augmented reality in minimally invasive surgery (2017). `arXiv:1708.01234`.

[8] L. Soler, S. Nicolau, P. Pessaux, D. Mutter, J. Marescaux, Real-time 3d image reconstruction guidance in liver resection surgery, Hepatobiliary surgery and nutrition 3 (2014) 73–81. doi:10.3978/j.issn.2304-3881.2014.02.03.

[9] P. Pratt, M. Ives, G. Lawton, J. Simmons, N. Radev, L. Spyropoulou, D. Amiras, Through the hololens^{TM} looking glass: augmented reality for extremity reconstruction surgery using 3d vascular models with perforating vessels, European Radiology Experimental 2 (12 2018). doi:10.1186/s41747-017-0033-2.

[10] M. H. Eid, C. N. D. Cecco, J. W. Nance, D. Caruso, M. H. Albrecht, A. Spandorfer, D. D. Santis, A. Varga-Szemes, U. J. Schoepf, Cinematic rendering in ct: A novel, lifelike 3d visualization technique., AJR. American journal of roentgenology 209 2 (2017) 370–379.

[11] E. Dappa, K. Higashigaito, J. Fornaro, S. Leschka, S. Wildermuth, H. Alkadhi, Cinematic rendering – an alternative to volume rendering for 3d computed tomography imaging, Insights into Imaging 7 (09 2016). doi:10.1007/s13244-016-0518-1.

[12] S. P. Rowe, J. Fritz, E. K. Fishman, Ct evaluation of musculoskeletal trauma: initial experience with cinematic rendering, Emergency Radiology 25 (1) (2018) 93–101. doi:10.1007/s10140-017-1553-z.
URL https://doi.org/10.1007/s10140-017-1553-z

[13] F. Fellner, Introducing cinematic rendering: A novel technique for postprocessing medical imaging data, Journal of Biomedical Science and Engineering 09 (2016) 170–175. doi:10.4236/jbise.2016.93013.

[14] M. Elshafei, J. Binder, J. Baecker, M. Brunner, M. Uder, G. Weber, R. Grützmann, C. Krautz, Comparison of cinematic rendering and computed tomography for speed and comprehension of surgical anatomy, JAMA Surgery (05 2019). doi:10.1001/jamasurg.2019.1168.

[15] S. Rowe, E. Fishman, 3d ct cinematic rendering of mandibular odontogenic myxofibroma, Oral Surgery, Oral Medicine, Oral Pathology and Oral Radiology (01 2019). doi:10.1016/j.oooo.2018.12.022.

[16] P. Ljung, J. Krüger, E. Gröller, M. Hadwiger, C. Hansen, A. Ynnerman, State of the art in transfer functions for direct volume rendering, Computer Graphics Forum 35 (2016) 669–691. doi:10.1111/cgf.12934.

[17] C. Viehland, B. Keller, O. M. Carrasco-Zevallos, D. Nankivil, L. L. Shen, S. Mangalesh, D. T. Viet, A. N. Kuo, C. A. Toth, J. A. Izatt, Enhanced volumetric visualization for real time 4d intraoperative ophthalmic swept-source oct., Biomedical optics express 7 5 (2016) 1815–29.

[18] E. Dappa, K. Higashigaito, J. Fornaro, S. Leschka, S. Wildermuth, H. Alkadhi, Cinematic rendering – an alternative to volume rendering for 3d computed tomography imaging, Insights into Imaging 7 (6) (2016) 849–856. doi:10.1007/s13244-016-0518-1.

[19] T. Kroes, F. H. Post, C. P. Botha, Exposure render: An interactive photo-realistic volume rendering framework, PLoS ONE 7 (7) (2012). doi:10.1371/journal.pone.0038586.

[20] M. Hadwiger, P. Ljung, C. R. Salama, T. Ropinski, Advanced illumination techniques for gpu-based volume raycasting, in: ACM SIGGRAPH 2009 Courses, SIGGRAPH '09, Association for Computing Machinery, New York, NY, USA, 2009. doi:10.1145/1667239.1667241.
URL https://doi.org/10.1145/1667239.1667241