SAVONIA
AMMATTIKORKEAKOULU

Implementation of volume rendering in C# for LightningChart

Alexey Tukalo

Bachelor's Thesis

June 3, 2016 _____

Bachelors degree (UAS)

SAVONIA UNIVERSITY OF APPLIED SCIENCES **THESIS**
Abstract

Field of Study Technology, Communication and Transport			
Degree Programme Degree Programme in Information Technology			
Author Alexey Tukalo			
Title of Thesis Implementation of volume rendering in C# for LightningChart			
Data	June 3, 2016	Pages/Appendices	41
Supervisor Arto Toppinen			
Client Organization/Partners Arction Oy			

Abstract

Arction Oy is a Finnish software company based in Kuopio. Their main product is LightningChart Ultimate. It is the fastest C# framework for the visualisation of scientific, engineering, trading and research data. The library contains a bunch of tools to draw of XY, 3D XYZ, Smith and polar graphs, 3D pie/donut views, 3D objects.

The company wanted to extend LightingChart's ability to render 3D polygonal models by volume rendering. It gives Arction an opportunity to attract new clients to use the product. As a result, the framework will provide a unique possibility to render volume and polygonal models at the same visualisation.

The project started from a literature research and comparison of different volume visualisation techniques. The best approach for the Arction's case was chosen and implemented it in the framework. The volume rendering engine is based on DirectX used together with C# via SharpDX API and HLSL for low-level optimization of complex calculations.

The final chapter of the report contains an evaluation of the results and suggestion for a future development of the engine.

Keywords
Visualisation, Ray Casting, 3D, C#, LightningChart, DirectX, HLSL, Image Processing, Volume Rendering, Rendering

ACKNOWLEDGEMENTS

I am very thankful to Arction Ltd for offering me an opportunity to take part in the development of the project. I like the office atmosphere and freedom regarding my working style and schedule allowed by the company.

My particular thanks go to Mr. Pasi Tuommainen, the CEO of Arction, who expressed interest in my idea to extend the library by the volume rendering engine. I am also thankful that he gave me permission to work on the project and guided me especially in the very early part of the development process.

I am very grateful to Savonia University of Applied Sciences and especially lecturers who used to teach me. Moreover, I would like to say thank you to my supervisor of thesis and head of my Degree Programm, Mr. Arto Toppinen, Principal Lecturer of Savonia UAS, for his mentoring and support during the report writing stage of my work.

Also, I would like to express my deepest gratitude to Karlsruhe Institute of Technology, where I got the first experience with volume rendering via Ray Casting. I am especially grateful to Nicolas Tan Jerome, who was my mentor during the part of my internship related to modification of TomoRayCaster 2 and to Aleksandr Lizin, the creator of the volume rendering engine based on WebGL which we used.

Contents

1 INTRODUCTION

This chapter contains brief information about the motivation behind a volume rendering, my personal background in computer graphics, especially in a volume rendering. It also introduces Arction as an owner of the project, explains the reasons for company's interest in the development, sets requirements for the final product.

1.1 Motivation

A volumetric data is very common our day. An importance of the type of datasets will grow shortly, because of development in the field of 3D data acquisition and possibilities to perform the visualisation of this kind of information on a modern office workstation with an interactive frame rate.

A volume rendering is a process of multi-dimensional data visualisation into a two-dimensional image which gives the observer an opportunity to recognize meaningful insights in the original information. The technology allows us to represent three dimensions of the data via position in a 3D space and three more via color of the point.

The dataset can be captured by various numbers of technologies like: MRI[1], CT[2], PET[3], USCT[4] or echolocation. They also can be produced by physical simulations, for example, fluid dynamics. The set of technologies mentioned before demonstrates that volumetric information plays an important role in medicine. It is used for an advanced cancer detection, visualisation of aneurysms and treatment planning. This kind of rendering is also very useful for non-destructive material testing via computer tomography or ultrasound. Geoseismic research produces huge three-dimensional datasets. Their visualisations are used in an oil exploration and planning of the deposit development.(Meiner, Pfister, Westermann & Wittenbrink 2000, 1-2)

1.2 Personal backgound

The first experience in the visualisation of volumetric data was gained by me during my internship at the Institute for Data Processing and Electronics, which belongs to the Karlsruhe Institute of Technology (KIT). I was a part of the 3D Ultrasound Computer Tomography (USCT) team there. Their main goal is a development of a new methodology for an early breast cancer detection. An algorithm for a visualisation of five-dimensional datasets was developed by me during

[1] Magnetic resonance imaging
[2] Computer tomography
[3] Positron emission tomography
[4] Ultrasound computer tomography

the work placement. In result it was integrated into TomoRayCaster 2[5] and USCT's edition of DICOM Viewer. An example of TomoRayCaster's output image is presented on the figure 1.1.

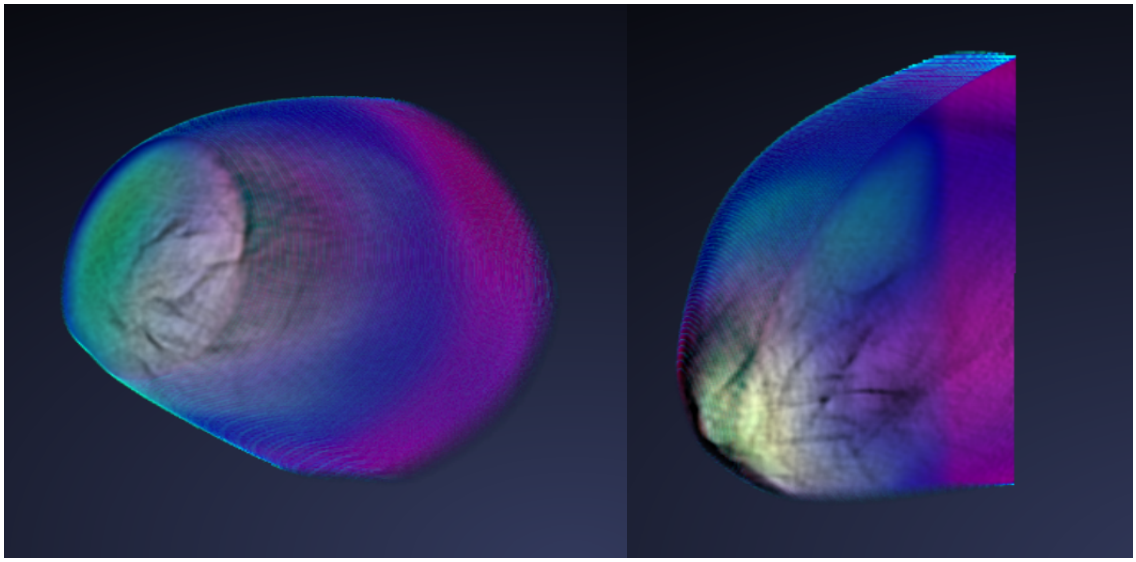

FIGURE 1.1: A volume visualisation of a breast phantom made by USCT(USCT Group, 2015)

The very first steps in modern computer graphics were made by me during the project. The first experience in work with WebGL was gained during customization of the TomoRayCaster. GLSL as my first shader language was also learned during the work. Much knowledge about an image processing and a scientific data visualisation, which became the basis for my thesis work was received by me at the job placement.

1.3 Arction Oy and Ligthning Chart

Arction Ltd is a Finnish software company based in Kuopio. Their team has a strong background in a computer graphic and sciences. The main product of the company called LightningChart Ultimate. It is the fastest C# library for a scientific and an engineering data visualisation. The library is capable of drawing massive XY, Polar, Smith and 3D XYZ graphs, polygonal mesh models, surfaces, 3D pies/donuts and Geographic information. The library has an API for .NET WinForm and WPF applications. It is also possible to use it for a traditional Win32 C++ software development. The main advantage of the library is the fact that it is based on low-level DirectX graphic routines developed by Arction, when the most of the competitors use graphics routines which belong to System.Windows.Media. Several example of the library's abilities is shown on the figure 1.2.(Arction Ltd A; Arction Ltd B)

[5]JavaScript framework for the visualisation of 3D data, prepared by Institute of Data Processing and Electronics

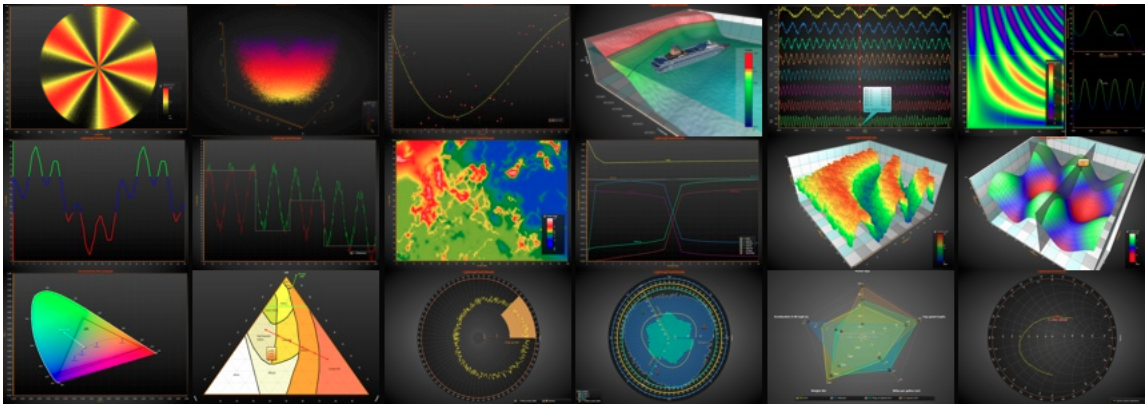

FIGURE 1.2: An example of a LightningChart's functionality

1.4 Project Goals

So, as it can be concluded from the previous section, LightningChart is a very advanced software for 3D rendering based on polygons and lines. That is why an idea to extend it by a particular rendering engine for a visualisation of volumetric data was suggested to the CEO of the company by me. The engine will give Arction's clients the unique opportunity to combine visualisation of volumetric datasets with a broad range of other 3D possibilities provided by the library.

The rendering engine must be able:

- to render large multi-dimensional volumes with an interactive frame rate.
- to move and rotate the model in the chart's space.
- to provide clients with possibilities to apply windowing and thresholding to the initial dataset.
- to render the model semi-transparently.

Basically, this tool will give end users opportunities to change the contrast and brightness of the model's visualisation for a better recognition of tiny details and make areas, which are out off a certain range, totally transparent. It will also reveal insights into the internal structure of the model to a user via semi-transparency.

2 THEORY

This chapter explains a theoretical base for the implementation of the project. It should introduce the main concepts of computer graphics, specify the difference between polygonal mesh model and volume rendering. It also contains an overview of different volume rendering techniques with their advantages and disadvantages regarding speed, a final image quality, a flexibility and other implementation issues.

2.1 Rendering

Visualisation of a 3D object as a 2D image called rendering. Usually, a 2D image is created out of pixels[1]. In the case of a grayscale picture, it is a two-dimensional array and the value of the array elements represents the brightness of corresponding pixels on a screen. The configuration of colored images is dependent on a color model. The most popular one is RGB. It represents an image as three different grayscale pictures for three different colors called channels. In case of the RGB color model the images contain Red, Green and Blue values, sometimes it also keeps an informant about opacity in the channel called Alpha.

A color model is a mathematical abstraction which allows computers to calculate the brightness of a corresponding point on the screen. An RGB is the original one for modern computer graphics because it represents a color in the way they are physically reproduced on a display. There are several other color models. They have their advantages, for example, some of them give us an advanced editing possibility while others represent physical characteristic of different types of output devices like printers.

Multidimensional data can be visualised in two different ways: as a surface and as volume. Future in this chapter, we are going to talk about these two concepts a little bit closer. We will highlight their advantage and disadvantages, common and uncommon features. Moreover, we are going to discuss an implementation detail of the techniques on modern hardware.

2.2 Polygonal Rendering

Today we are surrounded by the surface rendering based on a polygonal mesh. The technology is used in computer games, design, a cinema, science, engineering, etc. The technology is so popular that an entire 3D graphic pipeline is built around the idea. That is why this type of visualisation is easily accelerated by graphic cards.

[1] a shortcut for picture element

2.2.1 Vertexs

Traditionally, 3D surfaces are constructed out of huge amount of polygons connected as a mesh. Due to simplicity, they usually have a triangular shape. It is possible to describe a triangle via a list of three coordinates called vertices. An internal area of the shape filled with a color during a rasterization step. The color is calculated as a dot product between the normal vector of the surface and the vector of light. An example of a wireframe and a flat shading of a polygonal mesh model is demonstrated on the figure 2.1.(Frdo Durand, 2)

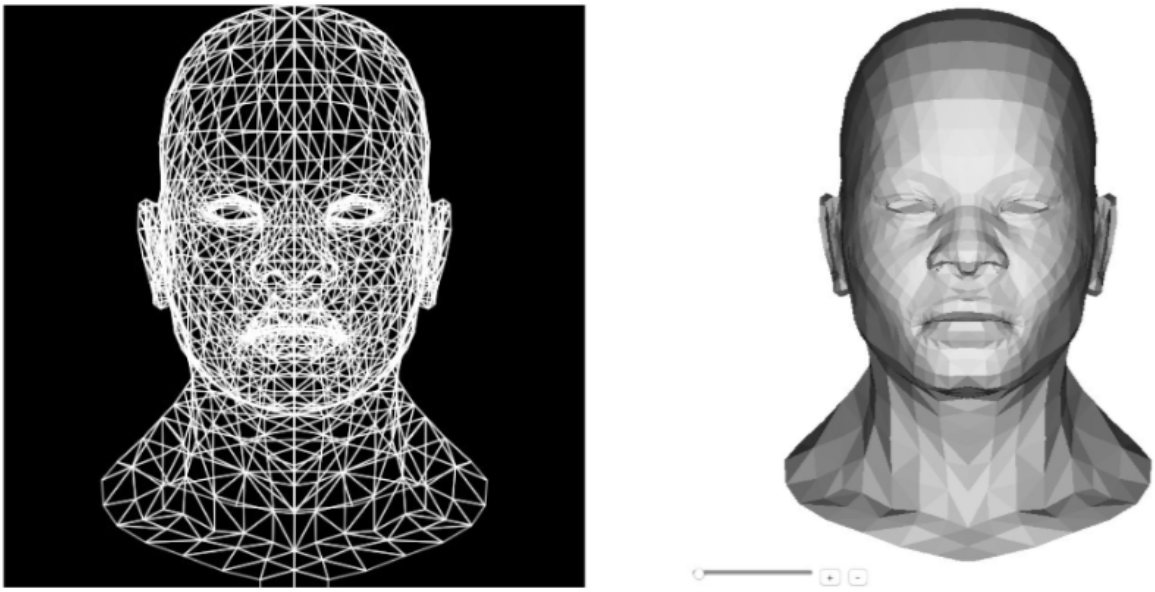

FIGURE 2.1: A wireframe and a flat shading of a polygonal mesh model provided by Vidar Rapp

2.2.2 Normals

The surface normal vector of a triangle can be calculated as a cross product of two triangle's sides, but it gives an acceptable result only for very flat surfaces. That is why curve surfaces usually contain an additional normal vector for an every vertex. They can significantly improve resolution on the model. The information kept in them is used during a shading and a tesselation of the model's geometry. (Wagner 2004, 1)

2.2.3 Textures

Tiny details can be added to the model via textures. It is specific 2D image which is used to sample high-frequency information during rendering. The picture usually contains local color

values, but it also can carry any information. For example, it can store normal vectors for a nice visualisation of very small structures on the surface. Textures are mapped to the surface via the third parameter of the vertex which is called texture coordinate. They keep any information about corresponding to this vector position in the 2D space of the image.(McReynolds 1996, 10-11)

2.2.4 Redering process

Position, scale, rotation of the object and perspective characteristics of the space is specified via matrixes 4×4. This kind of matrix allows applying any transformation possible in a 3D space. Usually, three matrixes are applied to the object. The first one transfers the object to the world space coordinates: it specifies the position, rotation, and scale of an original object in the world coordinate system. The next one is viewing matrix. It also takes in a consideration a position of a camera and transforms the models to achieve a desirable framing. The last matrix called projection. It applies perspective to the scene, in another word it defines an angle of view of the camera.(OpenGL-Tutorial)

A specific program called shader receives all the information together with a camera and light source positions, after that they can perform calculations needed to render the final image in accord with the goals of the visualisation. Usually, the calculations are produced by a graphic card in a parallel way.

2.3 Volume Rendering

Volume data are composed out of voxels[2]. Voxel is just a point in a 3D space, which has a position and a color. Together gives us an opportunity to visualise up to six scalar parameters. There are two ways to render volumes. We are going to discuss main principles, advantages and disadvantages of the technologies in this section.

2.3.1 Indirect

The first one called Indirect volume rendering. It is based on the idea that it is possible to extract a surface out of the dataset during a preprocessing and draw the surface as a polygonal mesh. Several algorithms are invented for this application:

- Marching Cubes
- Surface Tracking
- Fourier Transform Rendering

It is the oldest idea behind volume rendering, and it has plenty of disadvantages:

[2] volume element

- It is complex and slow preprocessing algorithms
- It can be inaccurate due to noise
- Sometimes it is not possible to generate an isosurface out of specific dataset, for example, smoke
- It loses information about an internal structure
- The preprocessing has to be repeated to apply changes in a transfer function

However, the algorithm is very popular for generation of medical illustrations, video or other static visualisations. The main advantage of the solution is that nicely preprocessed model can be easily rendered via well-known technique for 3D mesh model's rendering. They can be used even in a very weak hardware.(Jeroen Baert 2012, 3)

Unfortunately, this technology is not suitable for our project, because it does not satisfy Arction's requirements. First of all, it is not acceptable for us to lose the internal structure and an opportunity to runtime modify the transfer function.(Jeroen Baert 2012, 3)

2.3.2 Direct

Direct volume rendering does not require any kind of preprocessing. The data are visualised from an original dataset. It gives the algorithms an opportunity to modify a transfer function runtime. There is four most common technique for direct volume rendering. The algorithms will be discussed more detailed in the future in this section.

For an implementation of a hardware acceleration for a rendering process, a volumetric data has to be loaded to the graphics card. It is usually served as 3D texture or as a set of 2D textures. For the optimization of a texture's buffer consumption, several slices can be collected to one huge texture map. (Barbagallo 2014)

Texture-based

As it was mentioned before the graphics pipeline of modern graphic cards is optimized for a rendering of polygonal mesh models. So, volume rendering algorithms are forced to use this set of tools to achieve hardware acceleration. Texture-based Volume rendering is the most straightforward way to achieve the aim.

As it was already said textures are used to add tiny details to polygonal models. Any graphic library has an advanced set of tools for a texture mapping, which can be used to this kind of volume rendering. The algorithm creates a set of planes called proxy geometry. A transfer function maps the volumetric dataset on the proxy geometry. The final image is constructed out of the planes with mapped on them textures via Alpha-blending.(Ikits, Kniss, Lefohn & Hansen 2007)

A proxy geometry can be created in two different ways. The first one called 2D texture-based rendering. In this case, three different sets of planes are generated. All sets are generated perpendicular to a different direction of a 3D space. This approach leads us to sudden jumps in an image quality for various camera position, and it does not allow to change the sampling rate of the visualisation.(Ikits, Kniss, Lefohn & Hansen 2007)

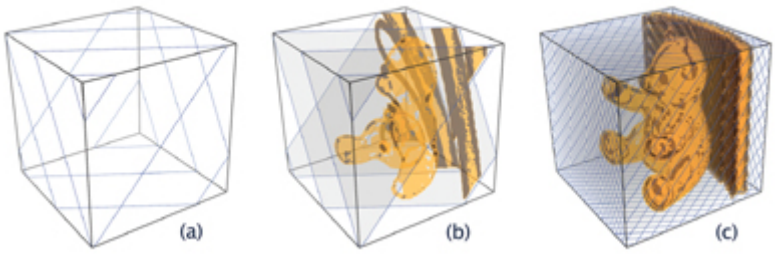

FIGURE 2.2: a) an example of proxy geometry for a 3D texture-based volume rendering b) a texture-based volume rendering with a lower sampling rate c) a high sampling rate texture-based volume rendering

That is why the second way was invented. It is called 3D texture-based volume rendering. In this solution, the algorithm creates only one set of planes which are always perpendicular to the camera and textures are mapped on them differently for the different camera position. The figure 2.2 shows an example of proxy geometry for the 3D texture-based volume rendering, a texture-based volume rendering with a lower sampling rate and a high sampling rate texture-based volume rendering.(Jeroen Baert 2012, 6-8)

It gets rid of an artifact of the first technique and allows to modify the sampling rate of the visualisation. The approach is the most popular direct volume rendering algorithm among a medical software. However, it has significant disadvantages against competitors. It can use, transfer function to emphasize or classify features of interest in the volume, while final blending is performed via Alpha-blending provided by DirectX API. It makes the approach less flexible than another method which provides more advanced tools for blending of sampled information.(Jeroen Baert 2012, 6-8)

Ray Casting

An image is produced by the algorithm throughout sampling of the volume of tracks of the rays which travel inside the dataset. A simple realization of hardware acceleration of the approach requires generation of boundaries for our volume. Usually, they are represented by a cube.

Volume Ray Casting includes four simple steps:

- An engine shoots a ray in a direction of observation for an every point on a screen.

- The ray travels through a scene and a dataset.
- A vector of the ray's track is calculated based on the position where the ray hits front and back faces of the cube.
- The volume is downsampled along the ray track and color of the pixel is calculated out of collected information by a Ray Function.

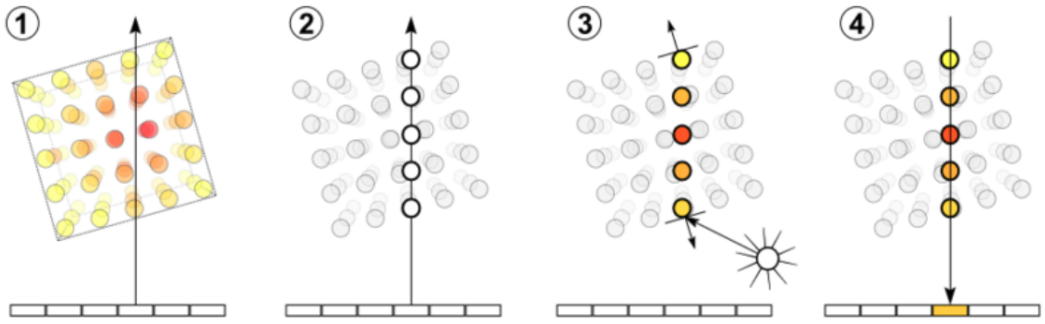

FIGURE 2.3: Ray Casting steps 1) The ray is shot, 2) The ray travels through a scene and a dataset. 3) Inlet and outlet position of the ray's track are detected 4) downsampling and the pixel's color calculation process

Ray Function is a core of the algorithm. Such a high level of flexibility is provided to the algorithm by the feature. It possesses an entire power of the technique because it specifies the way how the data is combined. That is why it is very powerful tool for a feature extraction. It controls how color, opacity, and gradient of isosurface are calculated by the engine. Transfer function and classification are also performed via the Ray Function.(Laramee, 2016)

The algorithm has very high rendering quality without any additional artifacts. Due to the reason that an every ray is calculated separately it can be easily implemented in a modern hardware which focuses on parallel calculations. The main problem of the algorithm is that rays usually do not hit the voxel's centers. In the case of 3D space, an interpolation is a very complex operation regarding calculational expenses, which has to be performed to get a nice sampling. (Jeroen Baert 2012, 18-30)

Splatting

The technique was created to reduce interpolation expenses which were the main problem of Volume Ray Casting. The solution uses totally opposite approach to reach the goal. Instead of sampling of the dataset by Ray, the algorithm projects the voxel to the image plane one by one. Of course, the voxel projection does not always fit exactly into pixel's grid of a screen, but the problem is again solved by an interpolation, fortunately, in this case, the operation is much less expensive in term of calculation, because it is performed in 2D space. (Jeroen Baert 2012, 10-11)

The final pixel's color is calculated in a very similar way used by the Ray Function that is why the algorithm is as flexible as Volume Ray Casting. The main disadvantage of the approach is that some artefacts are contributed to the final image due to an overlap of voxel's projections. It is also difficult to change the sampling rate of the approach. The issues make the algorithm less attractive for us.(Jeroen Baert 2012, 10-11)

Shear-warp

Shear-warp also tries to accelerate Ray Caster by solving of the interpolation issue, but it uses entirely different way to illuminate the problem. Shear-warp has a very similar idea to the Volume Ray Casting. In some sense, it is an optimization of the technique which makes rays always hit exactly in the center of the voxel.(Jeroen Baert 2012, 13-16)

The aim is achieved by the transformation of the volume data to a sheared object space by a translation and a scaling of slices. After that, a 2D intermediate image is produced by Ray Casting performed in the shared object space. The transformation brings some distortion to the output image. It is fixed at the last step of the algorithm called wrapping. The final image is the result of a transformation applied to the intermediate picture. The key difference between normal Ray Casting and Shear-Warp is shown in the figure 2.4.(Jeroen Baert 2012, 13-16)

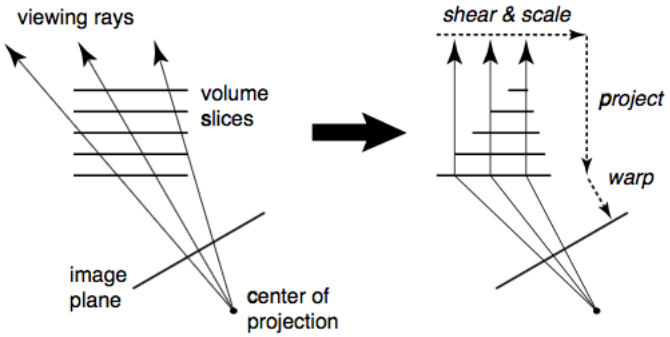

FIGURE 2.4: (left)Ray Casting in world coordinates, (right)Ray Casting in shear object coordinates

This solution gives us the fastest volume rendering process, but unfortunately, the algorithm also suffers from several types of artifacts. Some of them are caused by an unevenness of a sampling rate along different directions, and others are produced by 2D transformation on the wrapping stage.(Jeroen Baert 2012, 13-16)

3 IMPLEMENTATION

This chapter contains a detailed explanation of the product's implementation. It describes how the engine works, what kind of parts contains and which tools are used in it. The key steps of the software work are highlighted and explained in this chapter.

According to results of literature research, several the most common algorithms for volume rendering were compared, and the most suitable approach was implemented as Volume Rendering Add-On of LigthinghChart Ultimate. One of the main requirements for the LightningChart's volume rendering engine is a possibility to combine volume and other 3D visualisation at the same chart's space. Easy and accurate conversion between a volume's and a chart's coordinate systems is needed for this purpose. Only volume ray caster and texture-based volume rendering can provide us with the feature. The algorithms keep an entire visualisation inside the coordinate system of a proxy geometry. It is placed on the coordinate system of the chart. It means that the coordinates of the volume model can be easily converted to the coordinates of the entire chart. Shear-warp and Splatting do not use any proxy geometry, and that is why they are not good solution for Arction's case.

Another important thing is an implementation of rotation, scaling, and transmission of the model in the chart space. As it was already mentioned the proxy geometry also makes it much easier. However, this approach is not entirely applicable for a 3D Texture-based solution. In this case, slices always have to be perpendicular to the camera, and that is why rotation has to be implemented in the process of texture mapping.

Finally, there are only two technologies. The final choice is Ray Casting because it has better image quality than 2D Texture-based rendering. Also, Texture-based Volume rendering is less flexible than Volume Ray Casting.

3.1 Tools

This part of the chapter describes technologies used in the implementation of the project. It also gives a short explanation of their usage in the solution. The rendering engine has to become a part of LightningChart Ultimate. Integration with the library gives some restrictions regarding a set of tools which can be used in the project and it plays a key role in a technology selection.

3.1.1 C# and.NET

C# is a general purpose multi-paradigm programming language with strong types. It is created by Microsoft as the native language for .NET Framework. Our days the language can be used for development of Web services, Windows desktop, and mobile applications, computer games

with several different game engines. Also, Xamarin created the set of tools for cross-platform C# development.(Xamarin)

In some scene, it contains constructions inspired by an object-oriented, imperative, functional and many other programming paradigms. The language belongs to a C-like family, and that is why the curly-brace syntax looks very familiar for C, C++, Java developers. C# combines advantages of Java and C++ in a single, powerful and elegant way. It is more straightforward and safe than C++, but at the same times, it has advanced features like pointers arithmetics, enumerations, lambda expressions, structs, delegates and implicitly typed local variables. Even today when the most of them are implemented in the last version of Java, C# still is way more flexible.(Microsoft Development Center)

Another advanced feature of C# is LINQ[1] expressions, it is a bunch of functions for strongly-typed queries, which allows writing very short code in functional style for collection processing.(Microsoft Development Center)

A .NET Framework application runs in a virtual execution system called the CLR[2], The framework contains more than 4000 of classes which implement a broad range of useful functionalities. The runtime can execute an Intermediate Language, which is compiled from C# or 20 other languages.(Microsoft Development Center)

C# is used in the project to reach as close integration with LightningChart Ultimate as it is possible. It performs a management of visualisation process, a loading and preprocessing of the dataset.

3.1.2 DirectX 11

DirectX is a C/C++ API[3] for work with multimedia resources created by Microsoft for Windows and Xbox. It contains advanced tools for a rendering of 2D and 3D graphic and sound management. Direct3D is a part of the library responsible for hardware accelerated rendering of 3D graphics. The tool mainly focuses on a GPU accelerated rendering of polygonal mesh models.(TechTarget, 2005)

Rendering Pipeline

The key concept of the library is a rendering pipeline. It is a very general term in computer graphics. The pipeline is a sequence of stages which receives an input data as a set of polygons, textures and variables, process the information step by step and generates the final image in the end.

[1] Language-Integrated Query
[2] Common Language Runtime
[3] Application Programming Interface

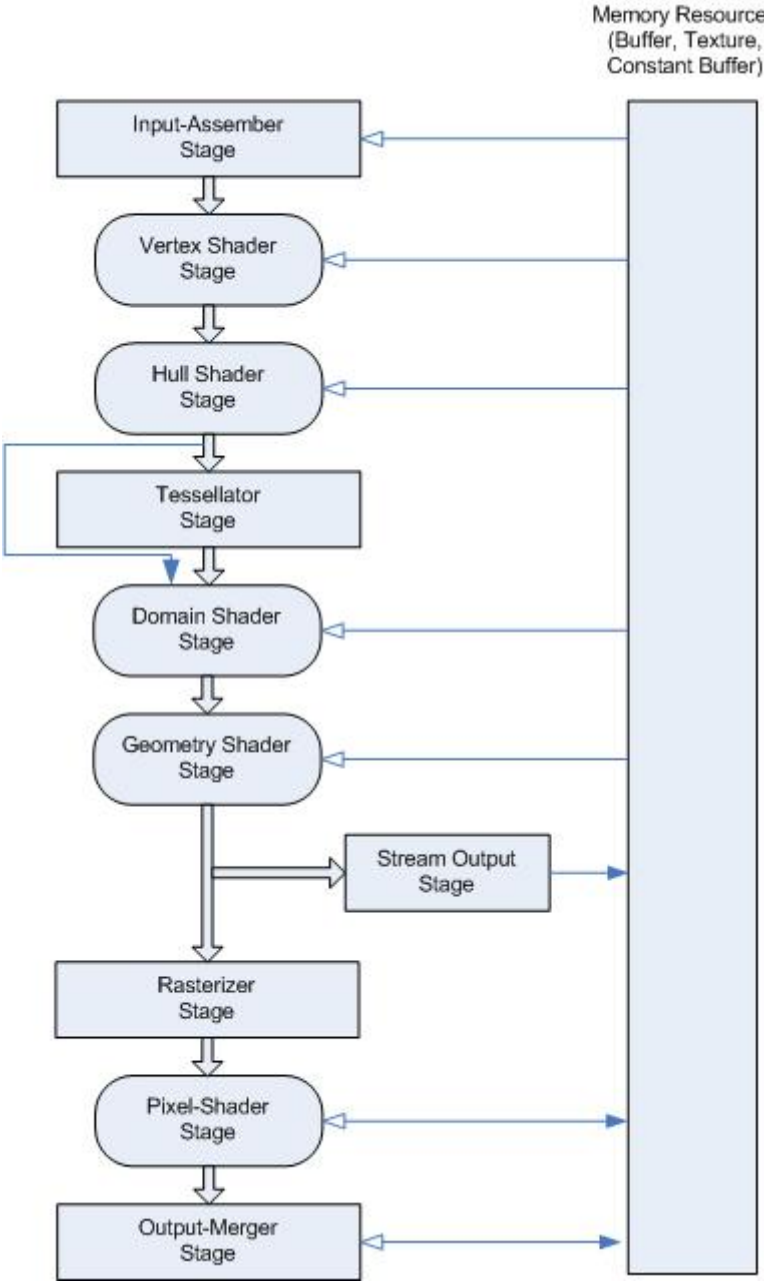

FIGURE 3.1: Flowchart of Direct3D 11 Graphics Pipeline

Direct3D's pipeline has two types of steps:

- Fixed-function - perform a certain processing operation which can be customized up to some level via the API.
- Programmable - processing is done by the so-called shader program. It is a function which

is created by the developer and satisfies input and output parameters of the stage.

The graphics pipeline is shown on Figure 3.1. Boxes with rounded angles represent programmable stages, and the other ones are fixed-functions. Here is more detailed explanation of the pipeline's steps:

- Input-Assembler Stage receives an input information as a set of buffers and supplies it future to the pipeline. There are three types of buffers in Direct3D:
  - Vertex Buffer is used to define a geometry of an object by a list of vertices. An every vertex has to follow Input Layout defined for the pipeline by the developer. It describes the vertex members: position and texture coordinates, colors, values, normals and so on. used in a particular application.(Windows Dev Center C)
  - Index Buffer contains an order of vertices as a sequence of integer variables. It can be used for an optimization of the Vertex Buffer because it allows avoiding repetition of vertexes common for several polygons.(Windows Dev Center C)
  - Constant Buffer provides pipeline with uniform variables, like transformation matrixes, position, and directions of light sources, textures, and so on.(Windows Dev Center C)
- Vertex Shader Stage is the first programmable stage of the pipeline. Vertices are read one by one and processed in accord with a Vertex Shader program which is supplied by the developer. For the best performance, the processing has to include an every operation which can be performed on an every vertex individually. The function is not able to destroy or create the new vertex.(Zink, Pettineo & Hoxley 2011, 4-8)
- Hull Shader, Tessellator, and Domain Shader stage are used together to achieve tessellation[4] of the mesh.
  - The Hull Shader program includes two functions. The first one receives primitives and calculates tessellation factors out of the data. The second function is called once for every control point and creates them.(Zink, Pettineo & Hoxley 2011, 4-8)
  - Tessellator uses data provided by Hull Shader and one of the several algorithms to choose the best point to break of the particular primitive into smaller polygons.(Zink, Pettineo & Hoxley 2011, 4-8)
  - Domain Shader uses an original information from Hull Shader together with results of the tessellator to generate the new vertex list.(Zink, Pettineo & Hoxley 2011, 4-8)
- Geometry Shader receives several vertices at the same time and performs operations on them. Ability to add and remove points from pipeline makes the function extremely powerful and flexible tool. For example, it can easily turn a single vertex to a polygon or even a set of polygons. It is able to pass geometry along the pipeline or to the output stream.(Zink, Pettineo & Hoxley 2011, 4-8)

[4]Process of mesh model's enchantment via automatic generation of additional polygons

- Rasterizer projects geometry to the final image. It determines which pixels are covered by the polygons. The vertices' attributes are interpolated inside the primitives to get the pixels values for the area. It also performs depth and stencil tests.(Zink, Pettineo & Hoxley 2011, 4-8)

- Pixel shader is executed once for an every pixel on the output target. It is supplied with information from Restriction step and calculates per pixel data. A texture sampling and Phong shading are implemented in pixel shaders.(Zink, Pettineo & Hoxley 2011, 4-8)

- Output-Merge Stage uses the pixel shader output together with depth/stencil information to generate the final image and writes it in an appropriate way to the render target.(Zink, Pettineo & Hoxley 2011, 4-8)

It is worth to mention that a render target is not always represented by a screen. Sometimes the output is stored in the texture. The image can be loaded again to the graphic card. For example, it can be used to get some pre-calculated information. The technique called multi-pass rendering.(Windows Dev Center B)

HLSL

There are two ways of a rendering pipeline customization. Fixed-functions behaviour can be modified by the API. A flexibility of programmable stages is organized via a special type of programming languages called shader languages. Direct3D's realization of a shared language called HLSL[5].

The language is significantly modified version of C. The language does not support advanced features like pointer's arithmetics and dynamic memory allocation, but the syntax is still understandable for C developers. It extends C syntax by classes, several new data types, buffers, semantics and the huge library of useful for shader development functions. There are the vector, matrix, and 1D-3D texture data types which are needed for a processing of 3D primitives. As it was already mentioned, buffers are used to supply pipeline with data. Semantics is very specific tool needed only in shaders. They contain metadata which is related to a data exchange among fixed-functions and programmable stages of the pipeline.(Zink, Pettineo & Hoxley 2011, 311-315)

3.1.3 SharpDX

As it was already mentioned DirectX is the C++ library, so it is not possible to use it directly from C#. The problem is solved by SharpDX. It is an open-source C# wrapper for the original DirectX API. It plays a role of the bridge between an API and.NET. As a result, developers are able to manipulate with the functional of the API from C#.(SharpDX)

[5]High-Level Shader Language

An every feature of the API is implemented in a very natural way. That is why a huge amount of tutorials and examples available for the original C++ API can be easily translated to C# and SharpDX. Even an original documentation is still useable for SharpDX developers.

3.1.4 LightningChart Ultimate

LightningChart Ultimate is the fastest C# library for a scientific and engineering data visualisation. It can draw massive XY, Polar, Smith and 3D XYZ graphs, polygonal mesh models, surfaces, 3D pies/donuts and Geographic information. (Arction Ltd B)

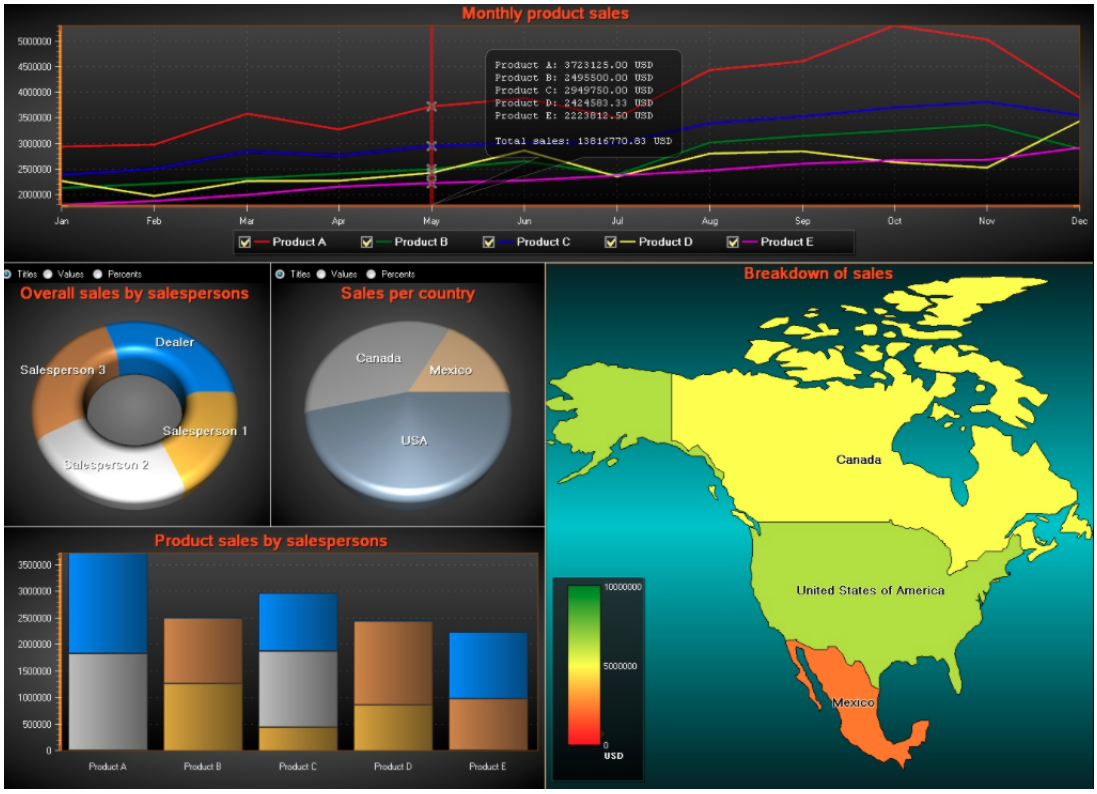

FIGURE 3.2: Some of LightningChart Examples

The library has an API for .NET WinForm and WPF applications, it is also possible to use it for a traditional Win32 C++ software development. The main advantage of the library is the fact that it is based on low-level DirectX graphics routines developed by Arction, when the most part of competitors base on graphics routines which belong to System.Windows.Media.(Arction Ltd B)

Integration with LightningChart has several advantages and disadvantages. On one hand, it plays the main role regarding tool selection for the project implementation, because deep enough integration can be achieved only via usage of an exactly same set of technologies. It also forces

the volume rendering engine to follow similar architecture. Moreover, LightningChart is a commercial software package. That is why any external dependencies have to be avoided as much as it is possible. It means that even open-source libraries cannot be used without a critical reason.

On the another hand, an ability of LightningChart to draw 3D polygonal mesh models makes the development much more simple. The library can rotate, scale and move the models around chart's space so that the functionality can be easily applied to a proxy geometry. As a result, the features are implemented without almost any development from the side of the volume rendering engine. In addition, it is a huge advantage for clients, because the volume and mesh 3D models can be visualised at the same chart in the same coordinate system. It provides them with really a unique and powerful opportunity for complex 3D visualisations.

3.2 Visualisation process

This section of the paper explains the main principles and implementation details of the project. It describes the main challenges and tells how they were solved during a development of the rendering engine.

3.2.1 Loading and preprocessing of dataset

The most common way of a volume data distribution is a collection of 2D slices. The 2D slices are just images in common formats. Sometimes they can be represented as a set of DICOM[6] files. In this case, they contain metadata of the studies together with embedded pictures. Volumes can also be distributed as a table of original values. The table usually is a part of an RAW file which also contains some meta information. Raw is several different formats with the common name. More detailed specification of an every particular realisation depends on a machine which performed the data acquisition.

Unfortunately, the current implementation of the volume rendering engine supports only the first approach, because slice-based data is easier to get. The specific software can convert DICOM slices to the picture formats supported by .NET like *. JPEG, *. TIFF and *. PNG. RAW volume data files can be converted to *.CSV files, but due to the reason that there are many different specifications of the-the extension it is tough to find the software.

.NET tools for file handling are responsible for the loading of images in extensions which are supported by the framework. The slices are loaded into the memory from specified folder as an array of bitmaps in the same order they are already stored. An example of a slice is shown on the figure . DirectX 11 also supports 3D textures, but the approach was not used in the engine because the feature is not compatible with DirectX 9. Arction wanted to have an opportunity to port the volume engine to DirectX 9 quickly if it would be needed.

Constant buffer has a limited amount of available slots for texture. Slices usually have a very low resolution, so it is not efficient to load them one by one. For an optimization of the buffers'

[6]Digital Imaging and Communications in Medicine

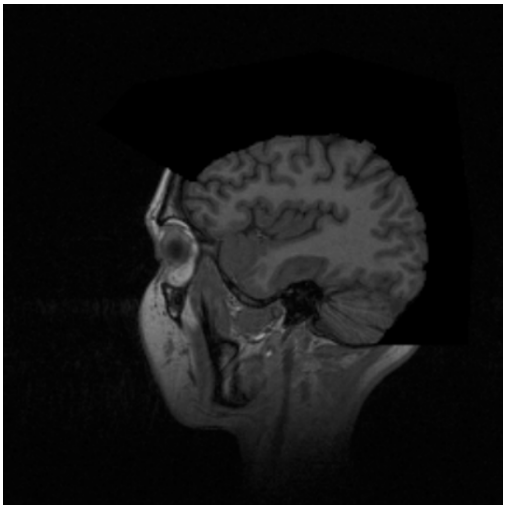

FIGURE 3.3: An example of the slice from Volume dataset.

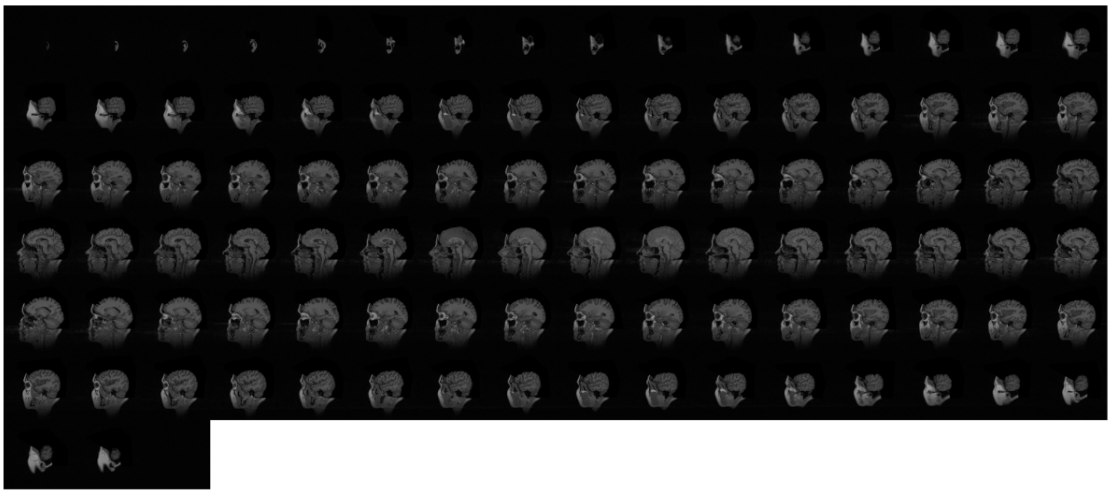

FIGURE 3.4: An example of slice map

consumption, an array of slices is mapped to a big texture map. The map contains slices mapped from left to right in accord with an array index, look the figure . A bitmap processing is accelerated via pointer arithmetic, however even in this case, it requires some time. That is why the engine is also able to load preprocessed texture map for a quicker start of visualisation.(Barbagallo, 2014)

DirectX has a limitation regarding the biggest available texture size, which can be loaded to the graphic card. The value depends on feature level of the graphic card, look table 3.1. Feature level defines the functionality supported by the graphic card. Usually, modern graphic cards have at least 11_0. It means that they support up to 16 384 pixels per aside. Such a large texture

TABLE 3.1: Maximum sizes of texture for different feature levels

Feature level	Maximum size of a texture
9_2	2048
9_3	4096
10_0	8192
10_1	8192
11_0	16384
11_1	16384
12_0	16384

can keep a volume dataset with more than 600 voxels long in an every dimension. It is possible to draw even bigger models, because several texture maps can be generated if it is not possible to fit an entire dataset into a single one.(Windows Dev Center A)

3.2.2 Multi-pass rendering

Possibility to render a scene in several steps makes rendering pipeline especially flexible. The technique is called Multi-pass rendering. Sometimes it is more efficient to render a scene in several stages. Multi-pass rendering is based on an idea that an every step except the last one outputs the result of performing a calculation to a texture. The final pass combines information from the set of textures and generates the output image.

Deferred rendering is one of the most popular realization of the idea. The technology is created for a rendering of scenes with a lot of dynamic lights. Traditionally, this kind of functionality requires a complex set of shaders. Multi-pass approach breaks the process into smaller pieces and performs the calculations separately for an every step. It allows simplifying an architecture of rendering framework. It can also give an opportunity to apply GPU acceleration to some parts of rendering algorithm which used to be previously executed on CPU.(Zink, Pettineo & Hoxley 2011, 491-495)

Our volume ray caster is also going to get some advantages from multi-pass rendering. In the case of this solution, it has only two passes. It means that entire rendering pipeline of the engine contains two typical rendering pipelines. The technology allows our algorithm to determine vectors of the sampling ray in the volume coordinate system without any additional mathematical calculations. The first pass generates texture which contains valuable information for a rendering of the final image at the second pass. Both passes use only vertex and pixel shader programs.(Kyle Hayward, 2009)

3.2.3 First pass

At this step rendering engine has to generate texture which keeps information about the exit position of sampling rays. For this reason, a front-face culling mode of Direct3D has to be turned

on. It makes the library draw only polygons which are usually not visualised because their front side looks in an opposite to the screen's direction.

An input of the first pass contains only a set of vertices of the boundaries of the dataset, constant buffers for slice range clipping and matrices for calculation of the vertex position of the chart space. The boundaries are always represented by a cube. The cube contains eight vertices calculated on C# in according with the required size of the volume model. A constant buffer is represented by two Vector3 variables, one of them contains the number of minimal slices for every direction, and another one keeps information about maximal boundaries. We also have three traditional for 3D graphic matrixes: World, View, and Projection Coordinate. This matrix has unique values for an every object and allows to calculate desirable positions, scales, and rotations of the objects in the screen coordinates.

The Vertex Shader of the first pass calculates positions of the cube vertices in according with a clipping range buffer. After that matrixes are applied to transform the geometry to the screen coordinates. Internal coordinates of the dataset are calculated out of an original position of a vertex. They are represented by Vector3, with value range from 0 to 1 along an every axis and stored as three-dimensional texture coordinates.

The Pixel Shader represents the coordinates as color. The X coordinate is stored in the red channel, Y one is saved in the green channel and Z coordinates is stored as the blue value. The final result of the first pass is demonstrated on the figure 3.5.

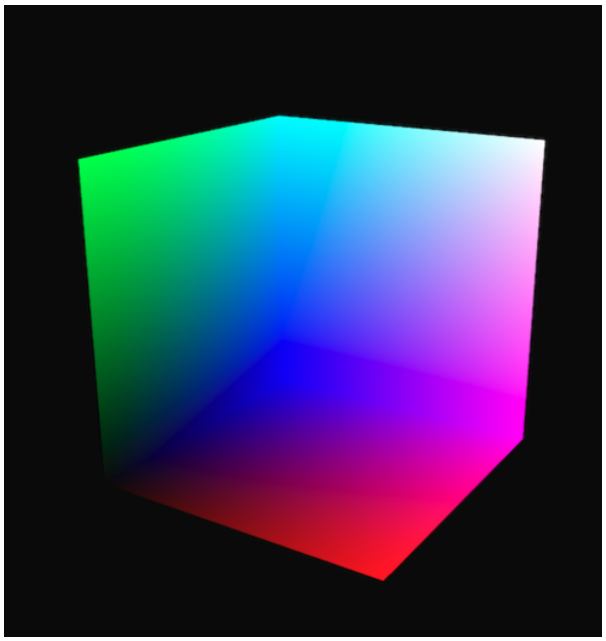

FIGURE 3.5: The first pass output

3.2.4 Second pass

At this pass, the engine has to calculate the vector of a sampling ray displacement through the dataset. The input coordinates of the rays are needed for the calculation that is why the default culling mode has to be restored. The coordinates are defined by the position where the rays hit at the front sides of the cube's planes. The input of the pipeline has a set of vertices, matrices, an original dataset represented as a texture map, texture created at the first pass, buffers with slice range, contrast, brightness and threshold settings.

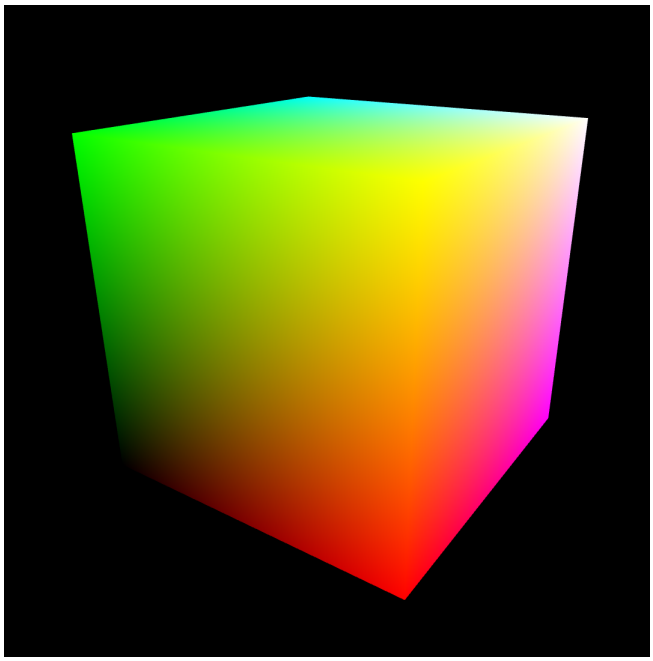

FIGURE 3.6: The coordinates of hit points.

The second pass has an exactly same vertex shader with the first one, but due to changes in the culling mode, it outputs the inlet points of sampling rays. The figure 3.6 shows them in the same way it was done on the first pass.

The pixel shader subtracts the coordinates where rays hit the front side of the cube, from the ones where the rays hit the rear aspect of the boundary. The input coordinates are calculated directly at the vertex shader of the second pass. The output coordinates are sampled from the texture generated at the first pass. The operation result is the vector of the ray's track. The tracks are visualised as colors on the figure 3.7. The vector is divided by sampling rate to get a sampling step vector of the ray. An actual volume sampling is performed by a for loop. It starts from the hit point of the ray and adds the step vector every iteration to get coordinates of the new sampling point. The data is sampled out of volume by a function which calculates the corresponding position on the texture map. The function takes into account the fact that the model

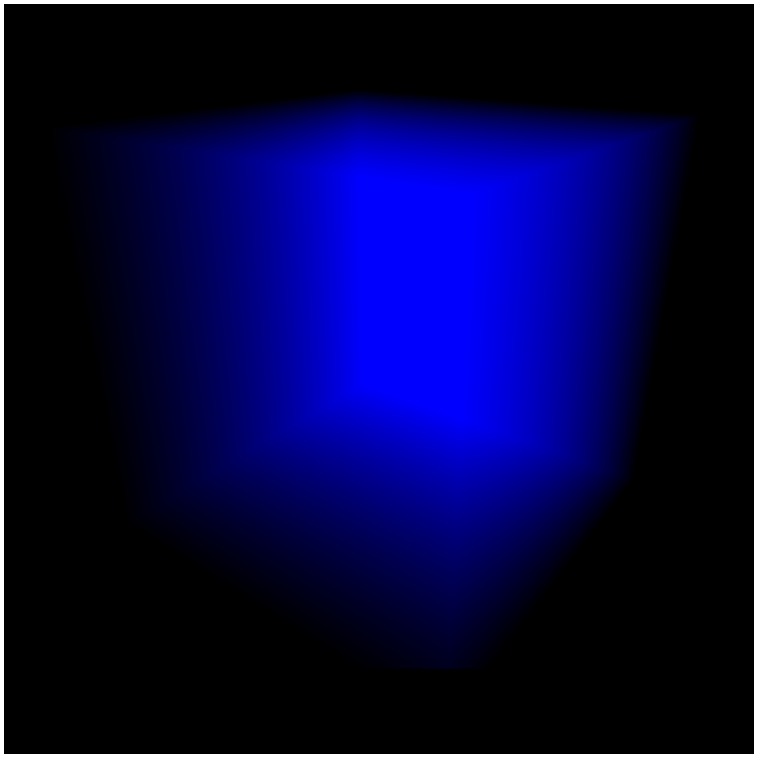

FIGURE 3.7: The visualisation of the sampling rays vectors' tracks

can have a different resolution along a different axis. It can cause a distortion of the shape. That is why the constant buffer contains special scaling factors. Multiplication of the coordinates by an appropriate factor gives right sampling position.

The function calculates the nearest slice corresponding to the Z position. After that, it checks how many full rows are included in the value and what is the column number of the rest. Values from two closest slices can be interpolated to get a smoother result. Handling of the information depends on a Ray Function.

Empty space skipping

The most part of any volumetric dataset is represented by an empty space. The empty space does not contain any relevant information. Usually, it is represented by pixels with a very low value per each channel. In our case, a voxel is classified as empty one if its values are out of an acceptable range for all of the channels. There is an enormous amount of rays which travels through arrays with only an empty space. Besides, there are areas with an empty space in front and behind the model. The optimization has to prevent high-resolution downsampling of this regions.

Due to the realization of the technique, the sampling is broken down into two steps. At the first step, ray travels through dataset with a very low sampling rate. It saves positions of the first and the last non-empty voxels which are met by the ray. After that, the second ray starts at a one low-resolution sampling rate step vector earlier the position of the first non-empty voxel. It stops at the one low-resolution sampling rate step vector after the very last non-empty voxel detected by the first ray. The process is shown on the figure 3.8. The Ray with a low sampling resolution can skip some part of non-empty voxels between sampling position. In this case, an actual border of the shape can be a little bit further or closer. Possible artifacts are prevented by the low-resolution sampling rate steps which are added in front and behind the model.

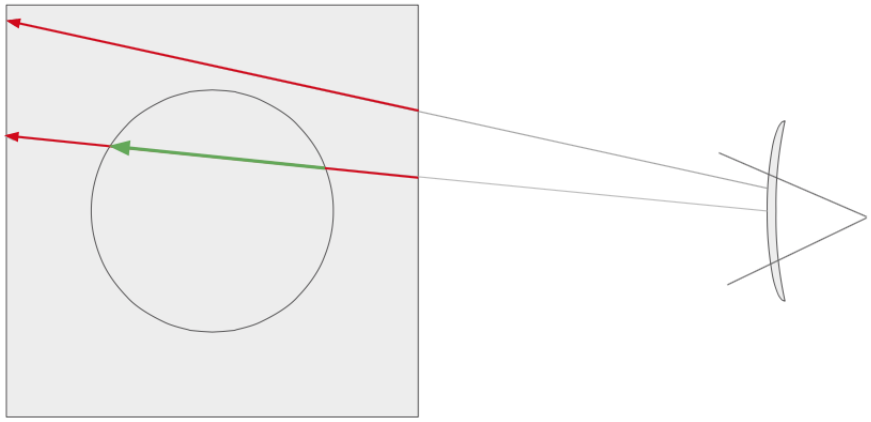

FIGURE 3.8: Red arrows represent the low sampling rate Ray. Green demonstrates high resolution sampling one.

High and low-resolution sampling rates are specified by a constant buffer. High sampling rate determines rendering quality. An optimal quality is reached, then it matches an amount of voxels of the longest dimension because it allows to perform uniform sampling of the entire dataset and collect all information stored inside. Lower resolution sampling can only be defined via experiment. Too small, low resolution of sampling rate can contribute some artifact to the final picture. It is a trade off quality and frame rate. The right value has to keep an interactive frame rate and unnoticeable level of artifacts.

Ray function

Ray function determines the way how the sampled data are combined. Different Ray functions allow extracting different features out of the dataset. Two the most common ray functions are implemented in our rendering engine.

Accumulation function uses an alpha-compositing to collect as much data as it is possible. The visualisation which is produced by this technique looks like a semi-transparent gel. The ray

function implementation is represented by a for-loop, which runs from 0 to the number of samples which is needed to perform sampling of the region in according with the desirable sampling rate. An every iteration contains several steps:

- The voxel is sampled out of the texture map.
- The validation of voxels value is performed by an if-statement. It checks that an every channel is in its personal threshold range. If the voxel is empty next four steps, have to be skipped.
- The alpha is calculated in accordance with an equation:

$$\alpha_{output} = \alpha \times opacity \times \frac{1}{sampling\ rate} \tag{3.1}$$

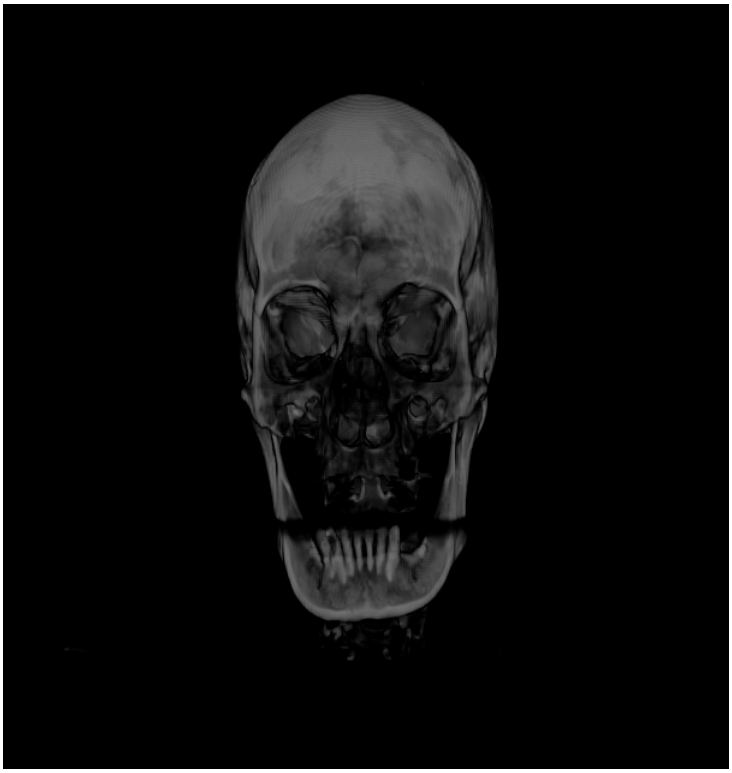

FIGURE 3.9: Visualisation of the female's head CT-scan with a Z coordinate scaling, a thresholding of soft-tissues which reveal a skull and a contrast correction, 512x512x230 voxels with and an accumulation function

- The colors are calculated in accordance with an equation:

$$color_{output} = (1 - \alpha_{accomulation}) \times (color + brighness) \times contrast \times \alpha \tag{3.2}$$

$\alpha_{accomulation}$ contains previously accumulated alpha values. The brightness and the contrast are supplied by constant buffers and used to perform dynamic range correction.

- After that the colors and alpha values are added to the corresponding variables which keep a previously accumulated information.
- If-statement checks that alpha is not oversaturated. The ray has to be terminated in this case, due to performance optimization.
- Step vector is added to reach next sampling position.

An example of images produced by an accumulation function is demonstrated in the figure 3.9.

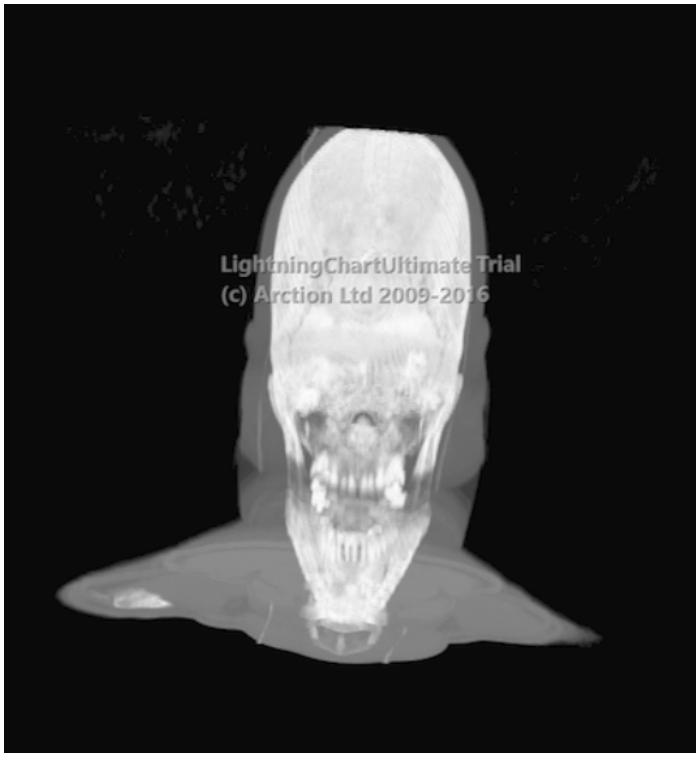

FIGURE 3.10: Visualisation of the female's head CT-scan with a Z coordinate scaling, thresholding of soft-tissues which reveal a skull and a contrast correction, 512x512x230 voxels with and a maximal intensity function

Maximum intensity function visualises only the brightest value sampled by the ray. It gives very similar results to the X-ray images. The ray function is very similar to the previous one. It is also used a for-loop and step vector to sample the dataset, but it does not accumulate the

information as it is done in the previous example. Instead, it keeps the biggest value which the ray meets during the travel and put it to the pixel in the end. It allows an end user to identify specific structures inside the volume. An example of the ray function's result is shown in the figure 3.10.(Ronald Peikert, 2007)

4 Conclusion

4.1 Results

The volume rendering engine developed for the thesis project successfully became a part of Lightning Chart Ultimate as the Volume Rendering Add-On. It is going to be distributed by Arction's brand new licensing system which allows clients to purchase components of Lightning Chart separately.

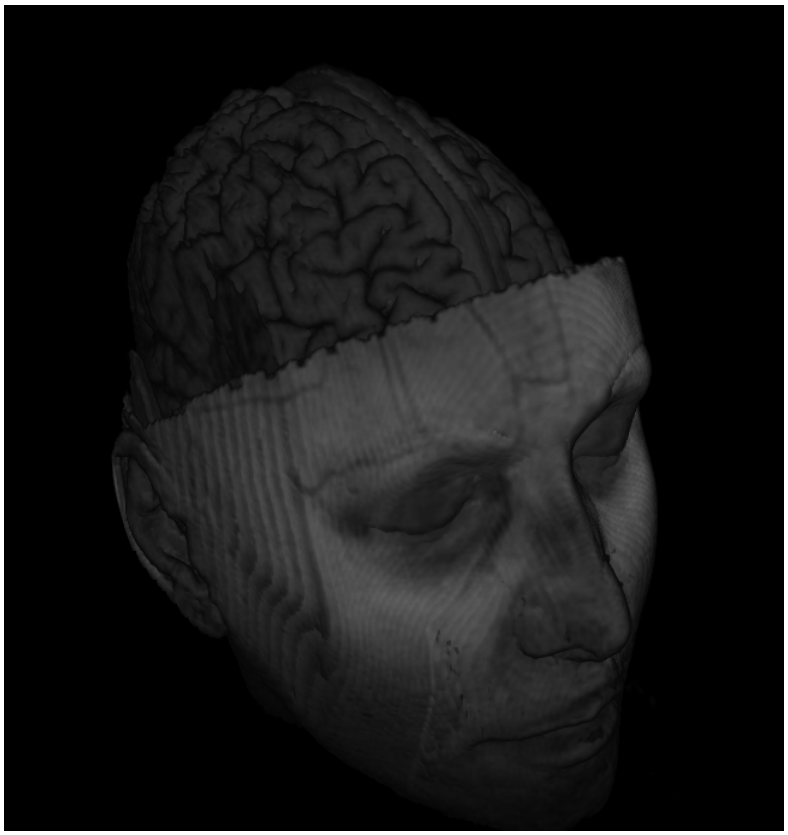

FIGURE 4.1: A semi-transparent rendering of a rotated model of male's head with a contrast brain.

The Ray Caster is able to visualise large multi-dimensional volume datasets. Theoretically, the size of the dataset is only limited by the total size of textures which can be loaded into the single texture array. In according with a documentation of DirectX 11, modern graphic cards can load more than 2048 textures with 16 384 pixels per a side on each. It is more than 549 billions of

voxels or cubic volume with more than 8000 of voxels along each axis.(Windows Dev Center D)

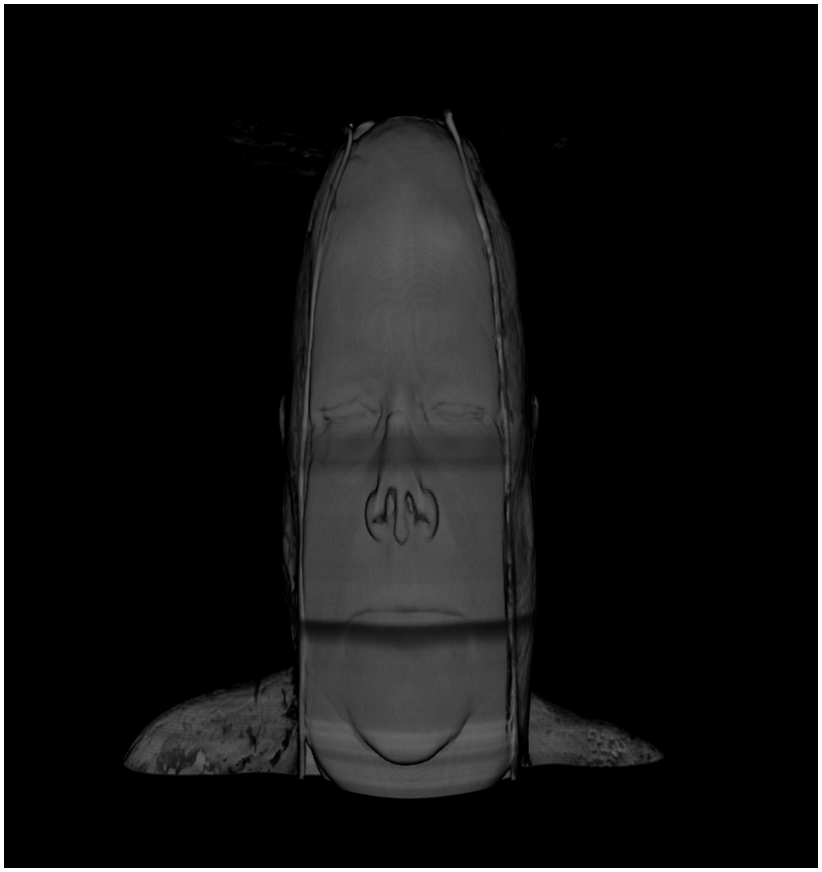

FIGURE 4.2: A visualisation of the female's head CT-scan, 512x512x230 voxels.

Of course, it is not realistic to say that can be possible to reach an interactive frame rate for such a huge volume model of modern hardware. Also, it is entirely useless to render volumes with a much larger amount of voxels per side than the resolution of the visualisation itself, because if a projection of a single voxel is smaller than a pixel, it would be too small to contribute any significant impact to the final image. That is why such a big dataset is useless even for a full-screen visualisation at 4K displays. Too large datasets can be carefully downscaled to get desirable resolution with the best performance.

Moreover, such a huge dataset is a very exotic case. A typical dataset does not contain more than 268 million of the voxel. It is even possible to fit this amount of information into a single texture map. It is about 600 of voxels per dimension in the case of a cubic shape of a volume. Our engine is able to render this amount of voxels with an interactive frame rate even in a very

weak graphic card. An example of the engine output is visualised on the figure 4.1. A part of the skull was removed during preprocessing. The segmentation was made to visualise the model with a technique which does not have a tool for a brightness thresholding.

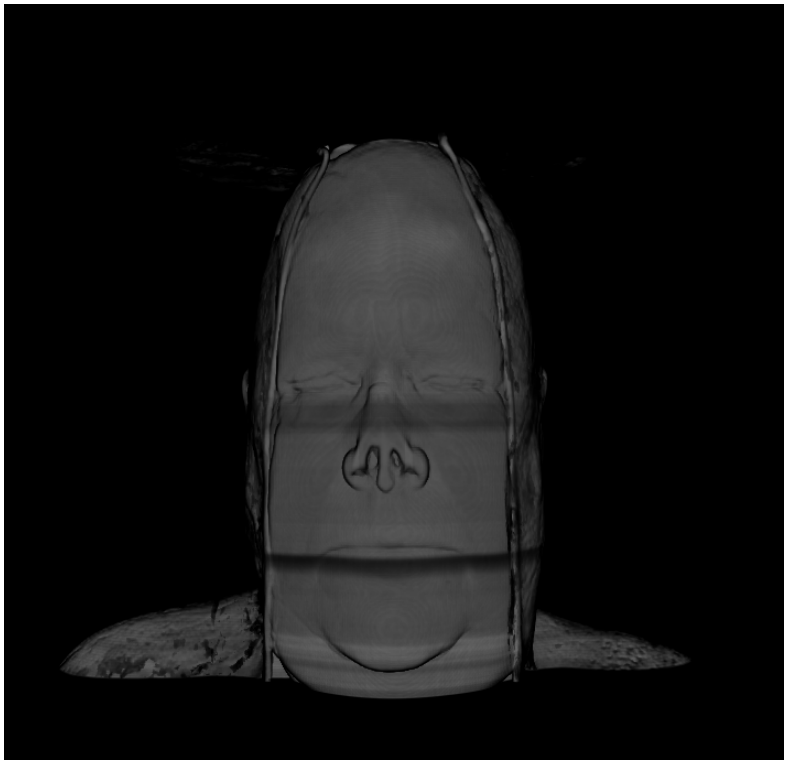

FIGURE 4.3: A visualisation of the female's head CT-scan with a Z coordinate scaling, 512x512x230 voxels.

Another example is shown on the figure 4.2. It is a female head captured by CT. The dataset is 512x512x230 voxels. The dataset has only 230 voxels along a Z - dimension. Since the volume is visualised as a dataset with a cubic shape, it makes the final model be two times stretched in this direction. The figure 4.3 demonstrates the object with a correction of the stretching via scaling of the Z coordinate.

There are two main types of tissues in the model: soft ones and bones. The skull and muscles are in different parts of the histogram, and that is why the engine can visualise them separately by a brightness thresholding. The thresholded skull is demonstrated on the figure 4.4.

The bone has a very low dynamic range. That is why a contrast and a brightness correction is very important in this case because it makes the skull allocate a full possible dynamic range. As a result, even tiny details of the bone are well recognizable. The initial shape of the model without threshold is demonstrated in the figure 3.9.

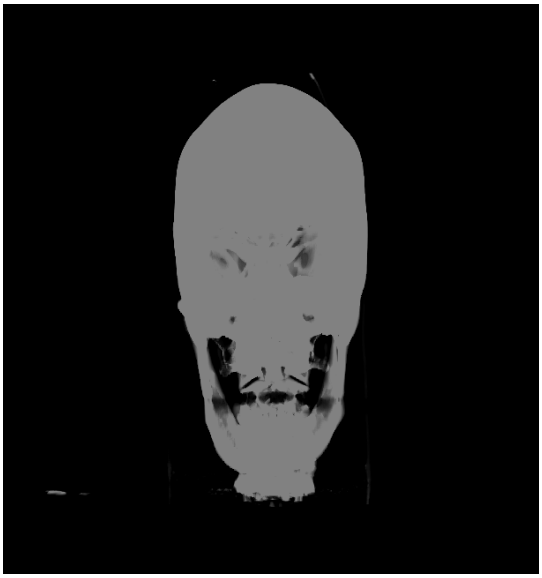

FIGURE 4.4: A visualisation of the female's head CT-scan with a Z coordinate scaling and a thresholding of soft-tissues which reveal a skull, 512x512x230 voxels.

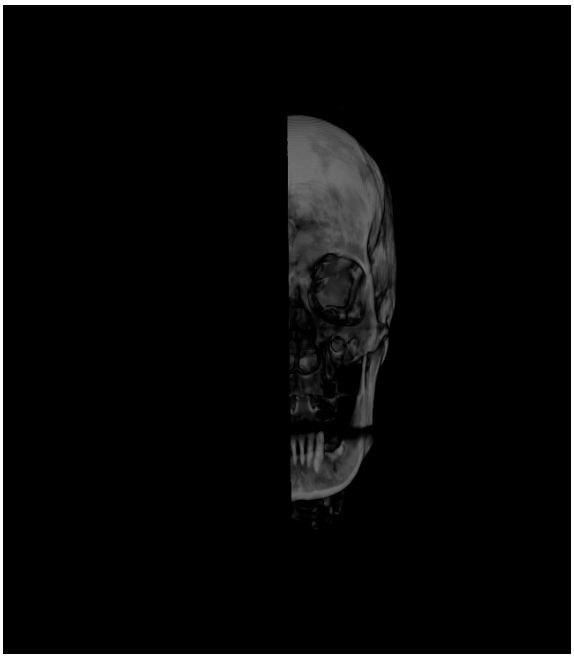

FIGURE 4.5: A visualisation of the female's head CT-scan with a Z coordinate scaling, a thresholding of soft-tissues which reveal skull, a contrast correction and a slice range clipping, 512x512x230 voxels.

A slice ranges clipping is also a very powerful tool for the engine. The figure 4.5 shows the skull with a cut away part on the left side. At certain angles, this kind of settings allows taking a look directly inside of the object, the figure 4.6. The model can be rotated, scaled and moved, because it is hosted by a cube. The cube is rendered by LigthningChart, and it allows to apply to the volume any transformation.

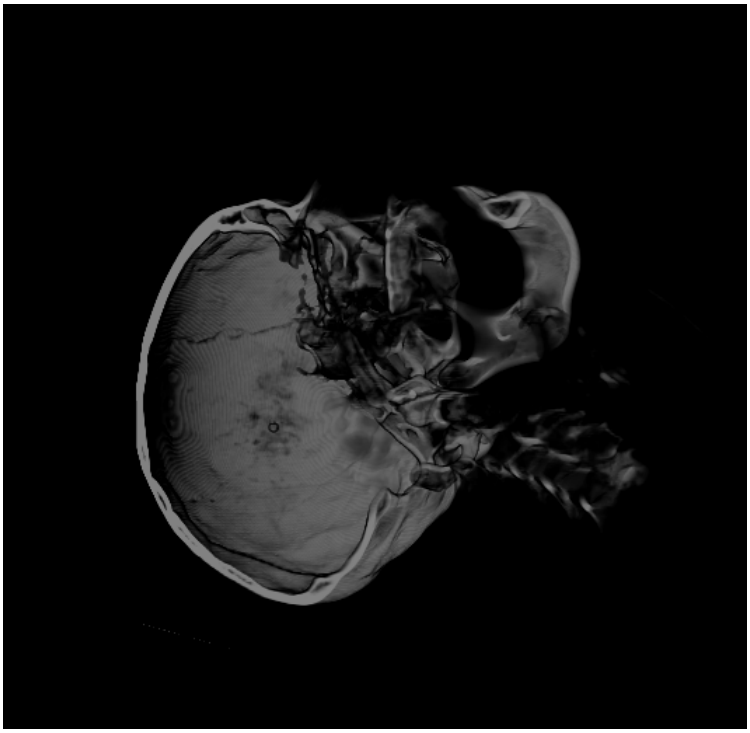

FIGURE 4.6: A visualisation of the female's head CT-scan with a Z coordinate scaling, a thresholding of soft-tissues which reveal the skull, a contrast correction, a slice range clipping and rotation, 512x512x230 voxels.

Besides, it is possible to display an internal structure via semi-transparent rendering. The example of the semi-transparent rendering is shown on the figure 4.7. The picture 4.8 contains an exactly same visualisation with a little bit different rotational settings. It does not have any thresholding. Contrast settings are similar with skull's visualisations. The result is achieved by a very low opacity. This kind of visualisation allows getting better understanding of the internal structure of the object.

There are two more examples of the engine output on the figures 4.9 and 4.10 . They show how all different settings can be combined with the single picture. This kind of visualisation can be used in a researching, planning or teaching process.

FIGURE 4.7: A visualisation of the female's head CT-scan with a Z coordinate scaling, a low opacity and a contrast correction, 512x512x230 voxels.

FIGURE 4.8: A visualisation of the female's head CT-scan with a Z coordinate scaling, a low opacity, a contrast correction and rotation, 512x512x230 voxels.

4.2 Future Development

The result of my thesis project is a fast, but at the same time a simple realisation of a volume rendering engine. There are a lot of different features, which can be added in future versions of the product. There are two main areas of growth: quality of rendering and performance for an extremely large datasets.

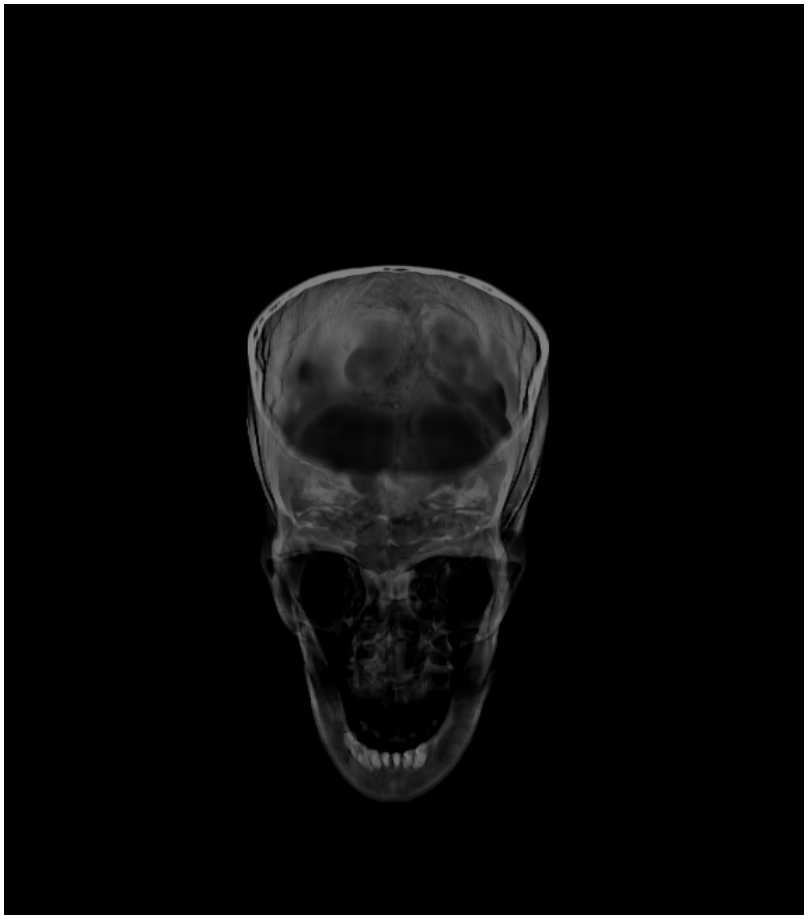

FIGURE 4.9: A visualisation of the female's head CT-scan with a Z coordinate scaling, a thresholding of soft-tissues which reveal skull, a contrast correction, a slice range clipping, a low opacity and a rotation, 512x512x230 voxels.

4.2.1 Picture quality

There are two main ways in terms of a quality improvement. Light interaction and isosurface rendering ray function were out of the thesis scope, but they have to be implemented in the engine in the next version of the library.

Light and shadow is a very important part of 3D visualisation. It takes a major part in the transmission of the 3D object's shape. The depth is usually evaluated by a combination of shadows and perspective. There are several techniques for a shading task. Some of them combine volumetric data with a polygonal model, and others do not. Not an every solution for the lightning of volume objects supports semi-transparent shadows. A shading significantly improves a quality of the final image but has very negative impact on the final performance. Performance is the main requirement for Arction's solutions, that is why the development will be done only after careful investigation of the issue.(Hadwiger, Ljung, Salama & Ropinski 2008, 56-61)

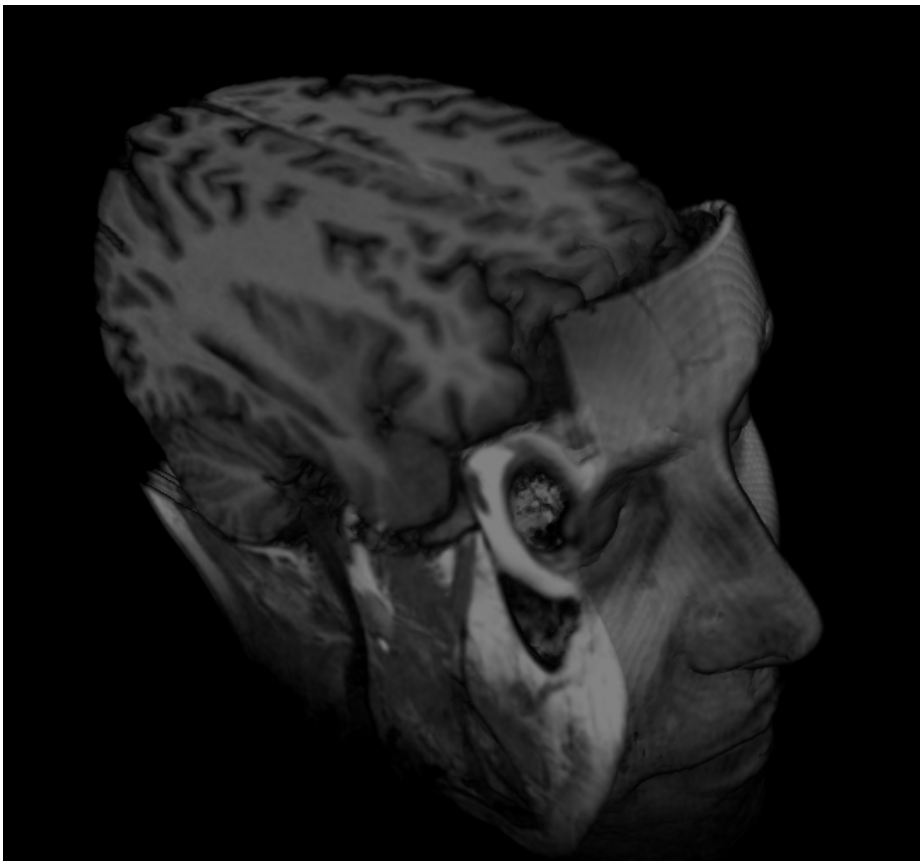

FIGURE 4.10: A semi-transparent rendering of the rotated model of a male's head with a contrast brain with a slice range clipping along two different axes.

An isosurface rendering draws the surface of the model in the way that it looks like a rendering of a polygonal model. The final result is very similar to the one which is produced by an indirect volume rendering. The visualisation seems much more impressive than the output of the accumulation function which is the main mode of LightningChart's Volume Rendering Add-On. An example of an isosurface and the accumulation rendering are shown on the figure 4.11, the comparing is not entirely fair, because the accumulation ray function does not implement an

interpolation while Arction's one use it to fill gaps between slices. Also, both of the functions have a very similar performance. However, an isosurface ray function has the same main problem with an indirect volume rendering. It is not able to visual use any information about an internal structure of the dataset. It makes the scientific application of the approach much more limited than the current solution has. It is the main reason why an implementation of the feature was not a high priority task for me on this project.(Hadwiger, Ljung, Salama & Ropinski 2008, 24-28)

FIGURE 4.11: A rendering of an accumulation ray function without an interpolation (left) and an isosurface ray function (right)(Hadwiger, Ljung, Salama & Ropinski 2008, 24-28)

The technique detects a voxel which satisfies the thresholds. Takes into account several pixels around the point and calculate a gradient vector. It is used as a normal vector. The brightness of the corresponding pixel is calculated in a very similar way with a traditional Phong approach for polygon-based 3D graphic. It is a dot product of the normal vector and the direction of light.

4.2.2 Sampling rate manager

In the future, a more complex ray function can become a reason for a performance problem. It can be solved by a sampling rate manager. The tool will add to the rendering engine an additional option which will turn on a dynamic control over the sampling rate. In the case of low frame rate, it will be able to drop the sampling rate during a transmission to make interactions with the model smoother. For a short period of time, the quality of the model will drop down to achieve more responsive rotation and transition of the object in the 3D space, but the general shape will remain to be recognisable. Right after that, the manager will set the default sampling rate, and the quality will be back.

Another application is a rendering a huge data set on a very weak hardware. Significant changes in the sampling rate should not have a dramatic effect on the quality of the final image, in the

case of models with a very high resolution. A relatively low sampling rate of the models will be at the same level as the average sampling rate of a less detailed volumetric object. That is why they will have to keep a good quality even in the low sampling rate mode.

4.2.3 API

Future development also has to be related to the extension of the API provided by the engine. The engine will have to be able to load data as a *.CSV files and receive the data created by a client's code. Additionally, Arction wants to implement a some interactive modification of the dataset runtime. It can be used for a visualisation of electromagnetic fields or another scalar datasets inside an object. A visualisation of a brain activity monitoring or a stress distribution inside a detail is excellent examples of applications for the approach.

The main problem is that it is very computationally expensive to modify the entire dataset on a CPU side for an every frame. That is why the most realistic approach requires the creation of an additional low-resolution dataset. An API will contain at least four functions. Two of them are a getter and a setter. The functions will give an opportunity to modify voxels of the dataset separately. It is more efficient to use pointer arithmetics for the modification. It requires locking of the bitmap and extraction of an array with raw pixel values. So, two more functions will be responsible for preparation the dataset for processing with a pointer arithmetics and a submission of the changes to the original bitmap. An every frame the low-resolution data will be updated and supplied to a graphic card.

Scalar fields do not usually have tiny details inside, also traditionally they are represented by a hue component of a color. It is hard for humans to reveal tiny gradations of a hue. That is why it would be difficult to recognize small details, and the dataset can have a much lower resolution. The final picture will be created by a fusion of this to datasets. The hue of the color will be sampled from a small dataset; an original one will be the source of the brightness. After that, the color will be converted from HSV color model to RGB one.

REFERENCES

Arction Ltd A*About us page* [webpage][accessed June 3, 2016]
Available from: http://arction.com/about_us

Action Ltd B*LightningChart Ultimate SDK* [webpage][accessed June 3, 2016]
Available from: http://arction.com/products_lc_ultimate_sdk

Jeroen Baert, 2012 *Rendering Large volumetric datasets: Overview, Using Sparse Voxel Octrees & GPU ray casting*[web document][accessed June 3, 2016] University of Leuven.
Available from: http://www.forceflow.be/wp-content/uploads/2012/02/vr_overview.pdf

Leandro R Barbagallo, 2014 *WEBGL VOLUME RENDERING MADE SIMPLE* [webpage][accessed June 3, 2016]
Available from: http://lebarbacom.ipage.com/blog/

Fredo Durand *A Short Introduction to Computer Graphics* [web document]MIT Laboratory for Computer Science[accessed June 3, 2016]
Available from: http://people.csail.mit.edu/fredo/Depiction/1_Introduction/reviewGraphics.pdf

Markus Hadwiger, Patric Ljung, Christof Rezk Salama & Timo Ropinski, 2008*Advanced Illumination Techniques for GPU-Based Volume Raycasting* [web document][accessed June 3, 2016] NACM SIGGRAPH Asia 2008
Available from: http://www.voreen.org/files/sa08-coursenotes_1.pdf

Kyle Hayward, 2009 *Volume Rendering 101*[web publication][accessed June 3, 2016] Available from: http://graphicsrunner.blogspot.fi/2009/01/volume-rendering-101.html

Milan Ikits, Joe Kniss, Aaron Lefohn & Charles Hansen, 2007 *GPU Gems* Chapter 39. *Volume Rendering Techniques* [web publication][accessed June 3, 2016] NVIDIA Corporation.
Available from: http://http.developer.nvidia.com/GPUGems/gpugems_ch39.html

USCT Group, 2015 *3D visualisation of USCT data* [webpage] Karlsruhe Insitute of Technology, Institute of data processing and electornics [accessed June 3, 2016]
Available from: http://ipepc57.ipe.kit.edu:10002

Philippe G. Lacroute, 1995 *FAST VOLUME RENDERING USING A SHEAR-WARP FACTORIZATION OF THE VIEWING TRANSFORMATION* [web document][accessed June 3, 2016] Computer Systems Laboratory Departments of Electrical Engineering and Computer Science Stanford University
Available from: https://www.youtube.com/watch?v=1PqvwOjnKJw

Robert Steven Laramee, 2016 *Ray Functions for Volume Rendering: A Quick and Convenient Review* [video][accessed June 3, 2016] Swansea University
Available from: https://graphics.stanford.edu/papers/lacrou_thesis/lacroute_thesis.pdf
Tom McReynolds, 1996*Programming with OpenGL: Advanced Rendering* [web document] Silicon Graphics, SIGGRAPH 96 Course [accessed June 3, 2016]
Available from: http://www.dcs.ed.ac.uk/teaching/cs4/www/graphics/Web/advanced_ogl.pdf

M. Meiner , H. Pfister , R. Westermann & C.M. Wittenbrink, 2000 *Volume Visualization and Volume Rendering Techniques*[web document][accessed June 3, 2016] The Eurographics Association 2000.
Available from: http://www.labri.fr/perso/preuter/imageSynthesis/02-03/papers/volvistut.pdf

Microsoft Development Center *Introduction to the C# Language and the .NET Framework* [webpage][accessed June 3, 2016]
Available from: https://msdn.microsoft.com/en-us/library/z1zx9t92.aspx

OpenGL-Tutorial *Tutorial 3 : Matrices* [webpage][accessed June 3, 2016]
Available from: http://www.opengl-tutorial.org/beginners-tutorials/tutorial-3-matrices

Ronald Peikert, 2007 *Raycasting* [web document] SciVis[accessed June 3, 2016]
Available from: https://graphics.ethz.ch/teaching/former/scivis_07/Notes/Handouts/03-raycasting.pdf

SharpDX [webpage][accessed June 3, 2016]
Available from: http://sharpdx.org

TechTarget, 2005 *DEFINITION: DirectX* [webpage][accessed June 3, 2016]
Available from: http://searchwindowsserver.techtarget.com/definition/DirectX

Max Wagner, 2004 *Generating Vertex Normals* [web document][accessed June 3, 2016]
Available from: http://www.emeyex.com/site/tuts/VertexNormals.pdf

Windows Dev Center A*Direct3D feature levels* [webpage][accessed June 3, 2016]
Available from: https://msdn.microsoft.com/en-us/library/windows/desktop/ff476876(v=vs.85).aspx

Windows Dev Center B*Graphics Pipeline* [webpage][accessed June 3, 2016]
Available from: https://msdn.microsoft.com/en-us/library/windows/desktop/ff476882(v=vs.85).aspx

Windows Dev Center C*Introduction to Buffers in Direct3D 11* [web document][accessed June 3, 2016]
Available from: https://msdn.microsoft.com/en-us/library/windows/desktop/ff476898(v=vs.85).aspx

Windows Dev Center D*Resource Limits* [webpage][accessed June 3, 2016]
Available from: https://msdn.microsoft.com/en-us/library/windows/desktop/ff819065(v=vs.85).aspx

Xamarin *Xamarin Planform* [webpage][accessed June 3, 2016]
Available from: https://www.xamarin.com/platform

Jason Zink, Matt Pettineo & Jack Hoxley, 2011 *Practical Rendering & Computation with Direct3D 11*
CRC Press, Taylor & Francis Group