Implementation of volume rendering in C# for LightningChart

Alexey Tukalo

Bachelor's Thesis

May 22, 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	May 22, 2016	Pages/Appendices	21
Supervisor Arto Toppinen			
Client Organization/Partners Arction Oy			

Abstract

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

The company wanted to extend LightingChart's ability to render polygonal 3D 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 shader language for low level optimisation 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 Oy for offering me an opportunity to take part in the development of the project. I really like the office atmosphere and freedom in terms of my working style and schedule allowed by the company.

My special thanks go to Mr. Pasi Toummainen, the CEO of the company, who expressed interest in my idea to extend the library by the volume rendering engine, 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 UAS and especially lectures who used to teache 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, for his mentoring and support during the report writing stage of my work.

In addition, I would like to express my deepest gratitude to Karlsuruhe Institute of Technology, there 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 Tomo Ray Caster 2 and to Aleksandr Lizin, the creator of the volume rendering engine based on WebGL.

Contents

1 INTRODUCTION

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

1.1 Motivation

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

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 3 dimensions of the data via position in a 3D space and 3 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 a big role in medicine. It is used for an advanced cancer detection, visualization of aneurisms and treatment planning. This kind of rendering is also very useful for non-destructive material testing via computer tomography or ultrasound. Geoseismic researches produce huge three-dimensional datasets. Their visualisations are used in an oil exploration and planning of the deposit development.

1.2 Personal backgound

The first experience in the visualisation of volumetric data was gained by me during my internship at the Institute of 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 the development of a new methodology for early breast cancer detection. An algorithm for 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 the it was integrated into Tomo Ray Caster 2[5] and USCT's edition of DICOM Viewer.

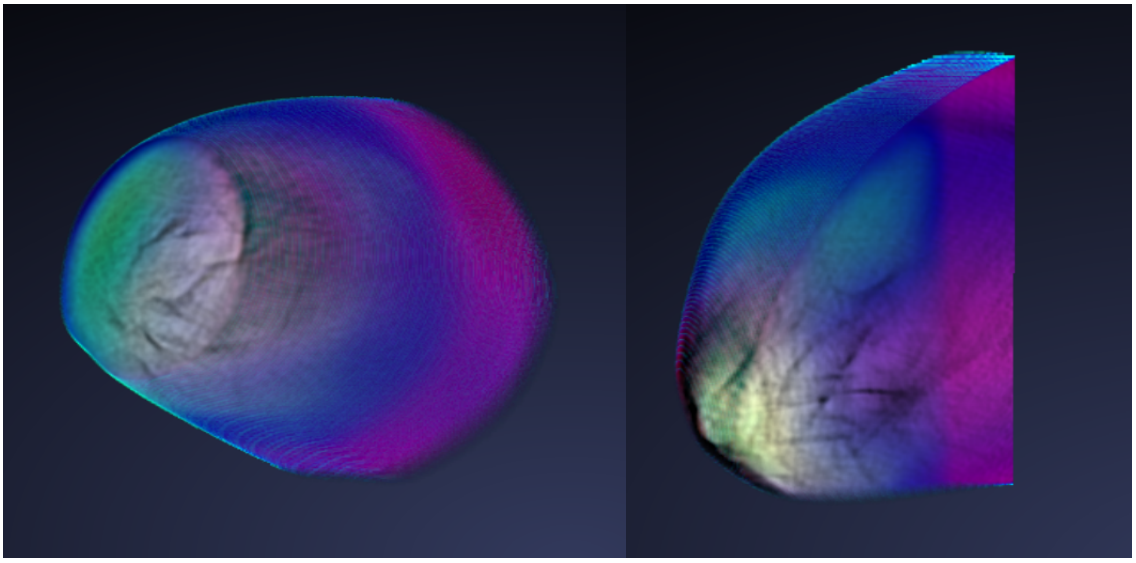

FIGURE 1.1: Volume visualisation of breast phantom made by USCT

The very first steps in modern computer graphics was made by me during the project. The first experience in work with WebGL was gained during customisation of the Tomo Ray Caster. GLSL as my first shader language was learned during the work. A lot of knowledge about image processing and scientific data visualisation, which became the basis for my thesis work was received by me at the workplacement.

1.3 Arction Oy and Ligthning Chart

Arction Oy is a Finnish software company based in Kuopio. Their team has a strong background in computer graphics and science. The main product of the company called LightningChart Ultimate. It is the fastest C# library for scientific and engineering data visualisation. The library is capable to draw 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 graphics routines developed by Arction, then the most part of competitors use graphics routines which belongs to System.Windows.Media.

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

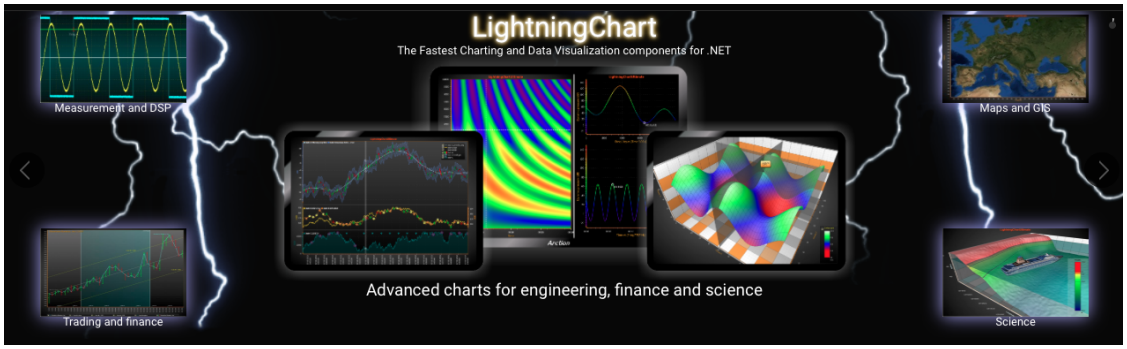

FIGURE 1.2: Example of LightningChart possibilities from the main page of Acrtion

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's why, an idea to extend it by the special rendering engine for visualisation of volumetric data was suggested by me to the CEO of the company. It will give Arction's clients the unique possibility to combine visualisation of volume datasets with a wide 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-transparent.

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

2 THEORY

This chapter explains the theory behind 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 in terms of speed, final image quality, flexibility and other implementation issues.

2.1 Rendering

Visualisation of 3D object as 2D image called rendering. Usually, 2D image is based on pixels[1]. In 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. Configuration of colored images is dependent from 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 and the channel called Alpha.

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

Multidimensional data can be represented 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 literally surrounded by the surface rendering based on polygonal mesh. The technology is used in computer games, design, cinema, science, engineering and etc. The technology is so popular that entire 3D graphic pipeline is built around the idea. That's why this type of visualization is easily accelerated by graphic cards.

[1] a shortcut from 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 triangle via list of three coordinates called vertices. The internal area of the shape filled with color during rasterization step. The color is calculated as dot product between the surface vector normal and the vector of light.

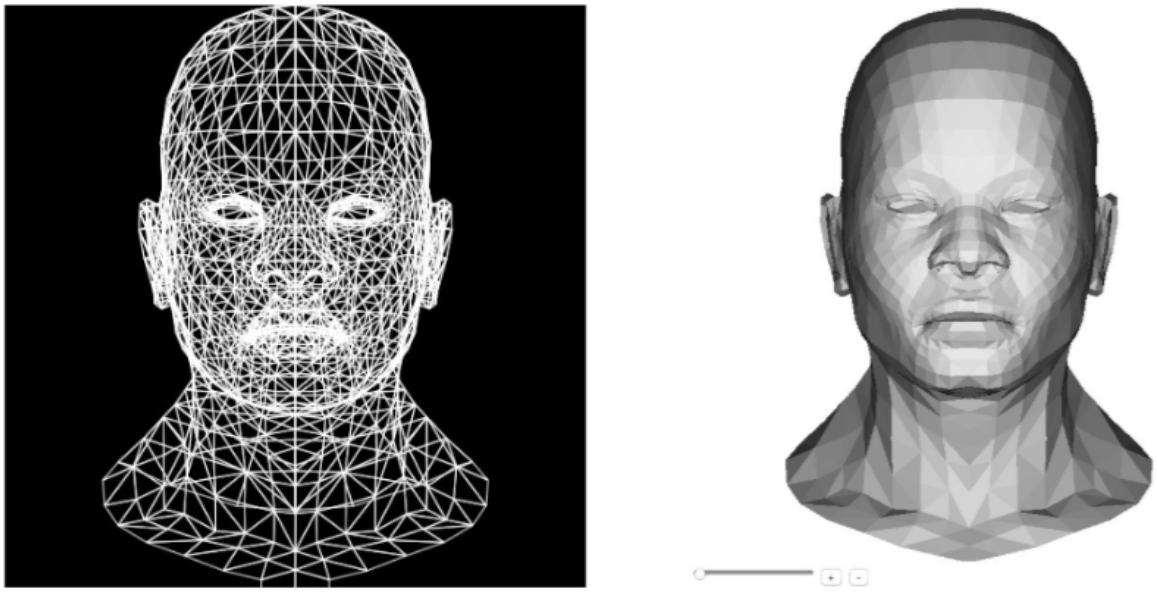

FIGURE 2.1: Wireframe and flat shading polygonal meshmodel

2.2.2 Normals

Surface normal vector of a triangle can be calculated as a cross product of two triangle's sides, but it will give an acceptable result only for very flat surfaces. That's why curve surfaces usually contain an additional normal vector for every vertex. They are able to significantly improve detailisation on the model. The information kept in them is used during shading and the tessellation of the model's geometry.

2.2.3 Textures

Very small 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 an information about normal vectors for a nice visualisation of very small structures on the surface. Textures are mapped to the surface via third vertex' parameter called texture coordinates. They keep an information about corresponding to this vector position

in the 2D space of the image.

2.2.4 Redering process

Position, scale, rotation of the object and perspective characteristics of the space is specified via matrixes 4×4.

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

2.3 Volume Rendering

Volume data are composed out of voxels[2]. Voxel is simply a point in 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 about 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 surface out of the dataset during preprocessing and render the surface as a polygonal mesh. Several algorithms are invented for this application:

- Marching Cubes
- Surface Tracking
- Fourier Transfor Rendering

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

- complex and slow preprocessing algorithms
- can be inacurate due to noise
- sometimes does not able to generate isosurface out of specific dataset, for example smoke
- lose an information about an internal structure
- need to repeat preprocessing to apply changes in a transfer function

[2] volume element

But the algorithm is very popular for generation of medical illustrations, video or other static visualisation. The main advantage of the solution is that nicely preprocessed model can be easily rendered in via wel-known 3D mesh models' rendering, even in very weak hardware.

Unfortunately, this technology is not suitable for our project, because it does not satisfy our requirements.

2.3.2 Direct

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

For an implementation of hardware acceleration for redering process a volume data has to be loaded to the grapic cards. It is usually served as 3D texture or as set of 2D textures. Several slices can be collected to one huge texture map due to optimization of a texture's buffer consumption.

Texture-based

As it was mentioned befor the graphics pipeline of modern graphic cards is optimized for rendering of polygonal mesh models. So, volume rendering algorithms are forced to use the set of tools to achieve hardware acceleration. Texture-based volume rendering is the most straight forward way to achive 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 texture mapping, which can be used to this kind of volume rendering. The algorithm creates set of planes called proxy geometry. Transfer function projects the volumetric dataset on the proxy geometry. The final image is rendered out of the planes with applied on them textures via Alpha-blending.

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 directions in 3D space. This approach leads us to sudden jumps in image quality for different camera position and does not allow to change the sampling rate of the visualisation.

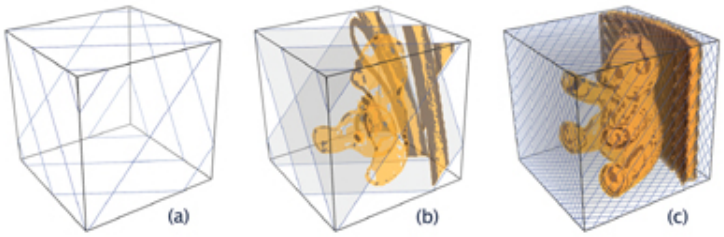

FIGURE 2.2: a) example of proxy geometry for 3D texture-based volume rendering a) Texture-based volume rendering with a lower sampling rate c) high sampling rate texture-based volume rendering

That's 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 different camera position.

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 medical software, but it was not selected for Arction's Volume rendering engine, because it is less flexible than all other methods discussed further. It is possible to use transfer function to emphasize or classify features of interest in the volume, but it is not comparable with possibilities provided by other approaches.

Ray Casting

Image is produced by the algorithm throughout sampling of the volume along tracks of the rays which travel inside the dataset. A smiple realisation of hardware acceleration of the aproach requer generation of boundaries for our volume. Usually they are represented by cube.

Volume Ray Casting includes four simple steps:

- An engine shoots a ray in a direction of observation for every point on a screen.
- The ray travels throught scene and dataset.
- A vector of the ray's track is calculated based on position there 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 in according with a Ray Function.

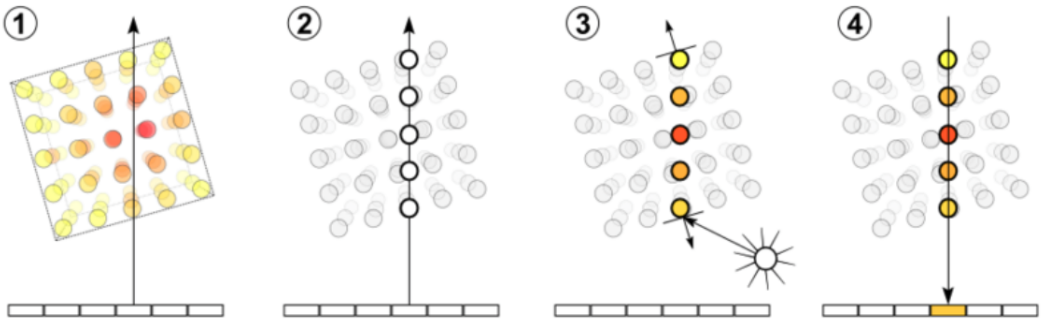

FIGURE 2.3: Raycasting steps: 1) The ray is shooted, 2) The ray travels throught scene and dataset. 3) Inlet and outlet position of the ray's track are detected 4) Downsampling and the pixel's color calculation process

Ray Function is a core of the algorithm. It possesses an entier power of the technique, because it specifes the way how the data is combined. Such a high level of flexibility is provided to the algorigth by Ray Function. It is very powerful tool for feature extraction, becuase it controls how color, opacity and gradient of isosurface are calculated by the enigene. Transfer function and classification are also performed via Ray Function.

The algorithm has very high rendering quality without any additional arifacts. Due to the reasone that every ray is calculated seperatly it is easilly implemented in modern hardware focused on parallel calculations. The main problem of the algorith is that rays usually do not hit the voxel to the center. In case of 3D space an interpolation is a very complex operation in terms of calculanal expensis, which has to be performed to get nice sampling.

Splatting

The technique was created to reduce interpolation expensis which were the main problem of Volume Ray Casting. The solution uses totally oposite aproach to reach the goal. Instead of sampling of the dataset by ray, the algorigthm projects the voxel to the image plane one by one. Of course the voxel porjections do not always fit excatly to pixels' grid of screen, but the problem is again solved by an interpolation, fortunately in this case the operation is much less expensive in term of calculation, becaus it is perfomed in 2D space.

The final pixel's color is calculated in a very similar way used by Ray Function, that is why the algorithm is as flexible as Volume Ray Casting. The main disadvanrage of the approach is that some artifacts are contributesd to the final image due to voxels overlap. It is also difficalt to change sampling rate of the approach. The issues make the algorithm less interesting for us.

Shear-warp

As Splatting, the aproach also tries to speed Ray Caster up by solving of the interpolation issue, but it uses totally different way to illuminate the problem. Shear-warp has a very similar idea

to the Volume Ray Casting. In some sence it is an optomozation of the technique which makes rays always hit excatly to the center of voxel.

The aim is achived by transformation of the volume data to sheared object space by translation and scaling of slices. After that an intermediat 2D image is produced by Ray Casting performed in the sheared object space. The transformation brings some distortion to the output image. It is fixed at last step of the algorithm called wraping. Final image is result of transformation applied to the intermideia picture.

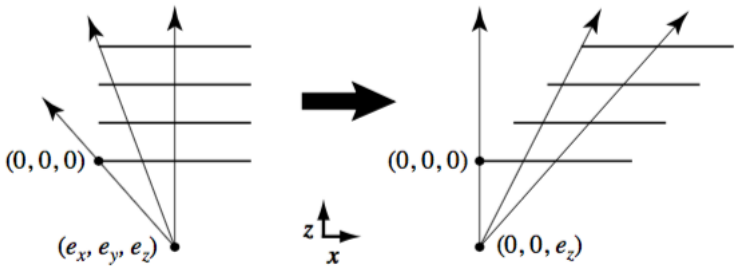

FIGURE 2.4: (left)Ray Casting in world coorinates, (right)Ray Casting in shear object coordinates

This solution gives us the fastest volume rendering process, but unfortunatly the algorthm also suffers from several types of artifacts. Some of them are result of sampling rate inconsistent along different directions, others are produced by 2D transformation at wrapping stage.

3 IMPLEMENTATION

This chapter contains detailed explanation of the product implementation. It discribes how the engine works, what kind of parts contains and which tools are used in it. Key step of the engine work are highlighted and explaned in this chapter.

3.1 Tools

This part of the chapter discibes technologies used in the implementation of the project. It also gives a short explation to thier usage in the solution. The rendering engine has to became a part of LightningChart Ultimate. An integration with the library gives some restrictions in terms of set of tools which can be used in the project and it plays a key role in technology selection.

3.1.1 C# and .NET

C# is general perpuse multi-paradigm programing language with strong types. It is created by Microsoft as native language of .NET Framework. Our days the language can be used for Web services, Windows desk-top and mobile applications, computer games with several different game engines. In addition, Xamarin created set of tools for croos-platfom C# development.

In some sence it contains constructions ispiered by an object-oriented, imperative, functional and many other programming paradigm. The language belongs to C-like family, that is why the curly-brace syntax looks very similar to the C, C++, Java. C# combines an advantages of Java and C++ in single powerful and elegant way. It is more simple 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 part of them are implemented in the last version of Java, C# still is way more flexible.

Other advanced feature of C# is Language-Integrated Query expressions or shortly LINQ, it is set of functions for strong-typed queries, which allows to write very short functional stylied code for collection processing.

.NET Framwork contains applications run in virtual execution system called the Common Language Runtime shortly CLR and more than 4000 of class which implements wide range of useful functionalities. The runtime is able to execute an Intermediate Language, which is compiled from C# or 20 other languages.

3.1.2 DirectX 11

DirectX is a C/C++ API[1] for work with multimedia resources created by Microsoft for Windows and Xbox. It contains advanced tools for rednering 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 is mainly focused on GPU accelerated rendering of polygonal meshmodels.

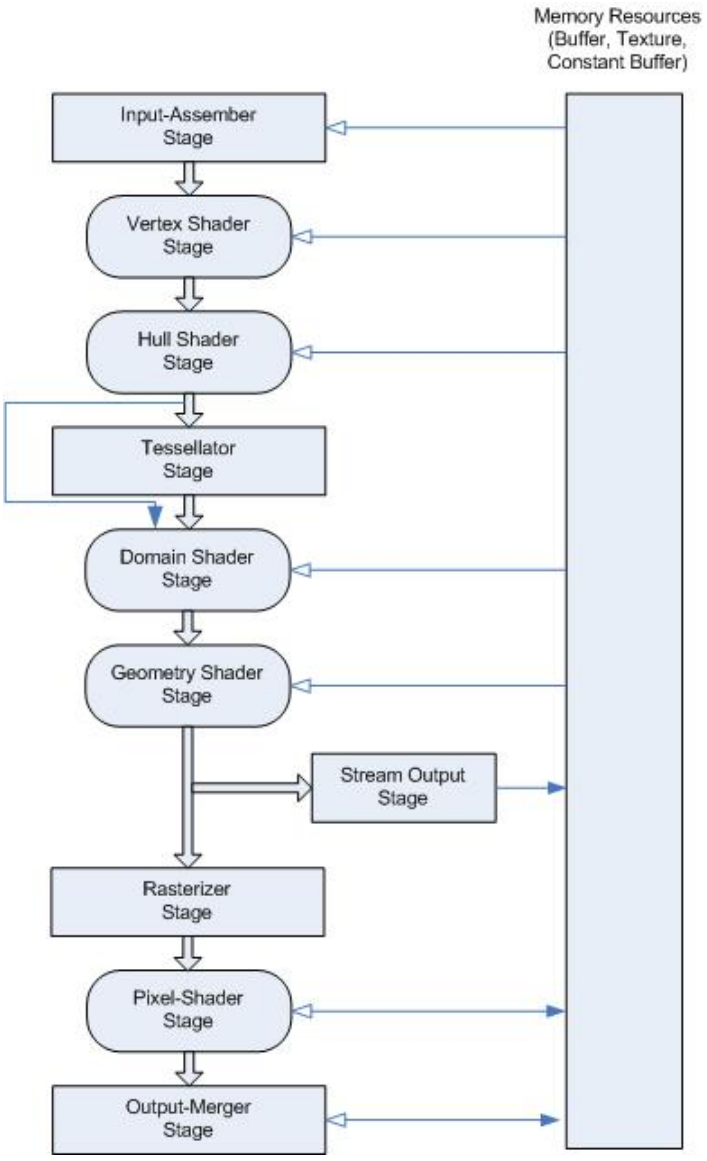

FIGURE 3.1: Flowchart of Direct3D 11 Graphics Pipeline

[1] Application Programming Interface

Rendering Pipeline

Key consept of the library is redering pipeline. It is a very general term in computer grapics. The pipeline is a siqence of stages which recieves an input data as set of polygons, textures and variables, process the information step by step and generates the final image in the end.

Direct3D's pipeline has two types of steps:

- Fixed-function - perform certain processing operation which can be customize due to some level thougth the API.
- Programmable - processing is performed by so called shader program. It is function which is created by developer and satisfies input and output parameters of the stage.

The graphics pipeline is shown on Figure 3.1. Boxes with rounded angles represent programable stages and other ones are fixed-functions. Let me exaplain every step of the pipeline more detailed:

- Input-Assembler Stage recieve input information as set of buffers and supply future to the pipeline. There are four types of buffers in Direct3D:
  - Vertex Buffer is used to define geometry of the object by list of vertexs. Every vertex has to follow Input Layout deffined for the pipeline by developer. It discribes the vertex members like: position and texture coordinates, color value, normal etc used in this application.
  - Index Buffer conatins an order of vertexs as sequence of integer variables.
  - Constant supplies pipeline with a constant data, like transfor matrixes, position and directions of ligth sources, textures, and so on.
- Vertex-Shader Stage is the first programmable stage of the pipeline. Verteces are read one by one and processed in according with Vertex Shader program supplied by developer. For best performence the processing has to includes every operation which can be performed on every vertex indivudualy. The function is not able to destroy or create new vertex.
- Hull Shader, Tessellator and Domain Shader stage are used togher to archive tessellation[2] of the mesh.
  - Hull Shader programm includes two functions. The first one recieve primitives and calculates tessellation factors out of the data. The second function is called once for every contorl point and creates them.
  - Tessellator uses data provided by Hull Shader and one of several algorithms to choose the best point to break of current primitive into smaller polygons.
  - Domain Shader uses an original infromation from Hull Shader toughert with results of tessellator to generate new verteces list.

[2]Process of mesh model's enchantment via aoutomatic generation of additional polygons

- Geometry Shader recieves several verteces at the same times and perform operations on them. An ability to add and remove points from pipeline makes the function extimly powerful and flexible tool. For example it can easelly turn single vertex to the polygon of even set of polygons. It is able to pass geomtry future along pipeline or away to output stream.

- Rasterization projects geometry to the final image and determines which pixels are covered by the polygons. It interpolates the verteces' atributes inside the primitives to get the pixels values for the area. It also perfoms depth and stencil tests.

- Pixel shader is executed once for every pixel on the output target. It is supplied with an information from Resterizetion step and calculates per pixel data for example color of imga's points. Usually texture sampling and hight quality lightning calculations are implemented by pixel shaders.

- Output-Merge Stage uses pixel shader's output togher with depth/stencil information to generate final image and write it in appropriate way to render target.

HLSL

There are two way of rendering pipeline customization. Fixed-functions behavoure can be modified by the API and programable stages flexiblities is orgonised via special type of programming languages called Shader Languages. Direct3D's realisation of shader language called HLSL[3].

The language is basically significantly modified version of C. The lanaguage does not support features like pointer artichmetics and dynamic momory allocation, but the syntax is still understandable for C developers. It extends C syntax by classes, several new data types, buffers, sematics and huge library of useful for shader development functions. There are vector, matrix and 1D-3D texture data types which are very useful for processing of 3D primitives. As it was already mentioned buffers are used to supply pipeline with data. Sematics contains metadata which is related with data exchange among fixed-function and programable stages of the pipeline.

[3]High Level Shader Language

3.1.3 SharpDX

3.1.4 LightningChart Ultimate

3.2 Visualisation process

3.2.1 Loading and preprocessing of dataset

3.2.2 Multi-pass rendering

3.2.3 First pass

3.2.4 Second pass

Empty space skipping

Ray function

4 Conclusion

4.1 Results

4.1.1 Rotation and position

4.1.2 Settings

Windowing

Thresholding

Slice range clipping

4.1.3 Mouse picking

4.2 Disscusion

4.3 Future Development

5 REFERENCES

6 Appendix