Beteckning:_____

HÖGSKOLAN
I GÄVLE

Akademin för teknik och miljö

Rendering Methods for 3D Fractals

Rickard Englund
June 2010

Bachelor Thesis, 15 hp, C
Computer Science

Computer Science program
Examiner: Anders Hast
Co-examiner: Stefan Seipel
Supervisor: Anders Hast

Rendering Methods for 3D Fractals

by

Rickard Englund

Akademin för Teknik och Miljö
University of Gävle

S-801 76 Gävle, Sweden

Email:
gefle.rickard@gmail.com

Abstract

3D fractals can be visualized as 3D objects with complex structure and has unlimited details. This thesis will be about methods to render 3D fractals effectively and efficiently, both to explore it in real-time and to create beautiful high resolution images with high details. The methods discussed is direct volume rendering with ray-casting and cut plane rendering to explore the fractal and an approach that uses super sampling to create high resolution images. Stereoscopic rendering is discussed and how it enhance the visual perception of the fractal.
Keywords: 3D fractals, ray casting, super sampling, stereoscopic rendering, Mandelbrot, 3D Mandelbrot , cut plane, off-axis rendering

Contents

1 Introduction

A 3D fractal is a set of 3D points which can be visualized as an object with a finite volume but contained within a surface with infinite area. This is similar to a 2D fractal which is a set of 2D points that creates a surface with finite area but with an infinite contour.

A Commonly used example used when explaining 2D fractals is the length of a coastline. Since a coastline never is 100% straight we cannot just measure from point A to point B. We could use a map and place a wire along the coastline and then measure the wire and multiply it with the scale of the map get the length of the coastline. But when we do this again but on a map with a lower scale we will get a result that is longer. This is because a map with higher details contains more bays, headlands and peninsulas. Repeating this in infinite number of steps and will always make the coastline "longer". [1]

When understanding fractals in 2D it is easy to understand fractals in 3D. Example of 3D fractals in nature is a mountain where the surface is infinite but the volume can be defined.

1.1 Problem definition

A new method to generate three dimensional fractals has been discovered in an ongoing research [2]. This method is similar to the Mandelbrot Set but has a self-defined algebra that is similar to Complex Numbers. The Mandelbrot Set uses Complex Numbers with one real part and one imaginary part. The new method uses one real-part and two imaginary parts which gives us three dimensions instead of two. The algorithm for creating the fractal is relatively simple but computational heavy and is therefore hard to visualize in an effective way.

1.2 Aim

The aim with this project is to research which method that can be used to render this fractal in effective and efficient way. An application in which the user can interact in term of zooming, rotating and translate in real time to explore the fractal should be implemented and evaluated. A possibility to create high resolution images with high details is also to be implemented. An evaluation on how the stereoscopic rendering can enhance the visual perception should also be done and if result is positive the application should have possibility to render stereoscopic images.

1.3 Questions at issue

What methods and optimizations exist for volume rendering and which methods suits best for fractal rendering?

What illumination model can be used to illuminate the fractal to create the visualization?

What method can be used to produce a high resolution image with many details for 3D fractal?

Which methods exist to create stereoscopic rendering and will stereoscopic rendering enhance the visual perception?

2 Theoretical background

In this section theory about 2D and 3D fractals will be explained. Reflection on other researchers works around 3D fractals. Volume rendering will be briefly explained and optimization methods will be discussed and what illumination models that fit with volume rendering. In the end of the section stereoscopic rendering will be explained and methods to achieve a good stereoscopic result.

2.1 Fractals

A common fractal is the Mandelbrot set. The Mandelbrot set was discovered 1905 by Pierre Fatou and have since then been studied by many researchers. The picture that most people associate with the Mandelbrot set was first created in 1980 by Benoît B. Mandelbrot [3]. The definition of the Mandelbrot set is given by the complex progression defined in equation 1, where Z and C are complex numbers and Z_0=0+0i.

$$Z_{n+1} = Z_n^2 + C$$

Equation 1: Definition of the Mandelbrot set.

When generating an image of the Mandelbrot set C's real and imaginary part are set depending on the x and y coordinate. In figure 1 we see a image of the Mandelbrot set generated in a computer where the real part of C varying between about -2 and 1 and imaginary part varying between about -1 and 1. The color is set depending on after how many iterations that are needed before the absolute value of Z tends to infinity. The black area is where the absolute value tends to zero (The almost black area outside where Z tends to infinity just after very few iterations).

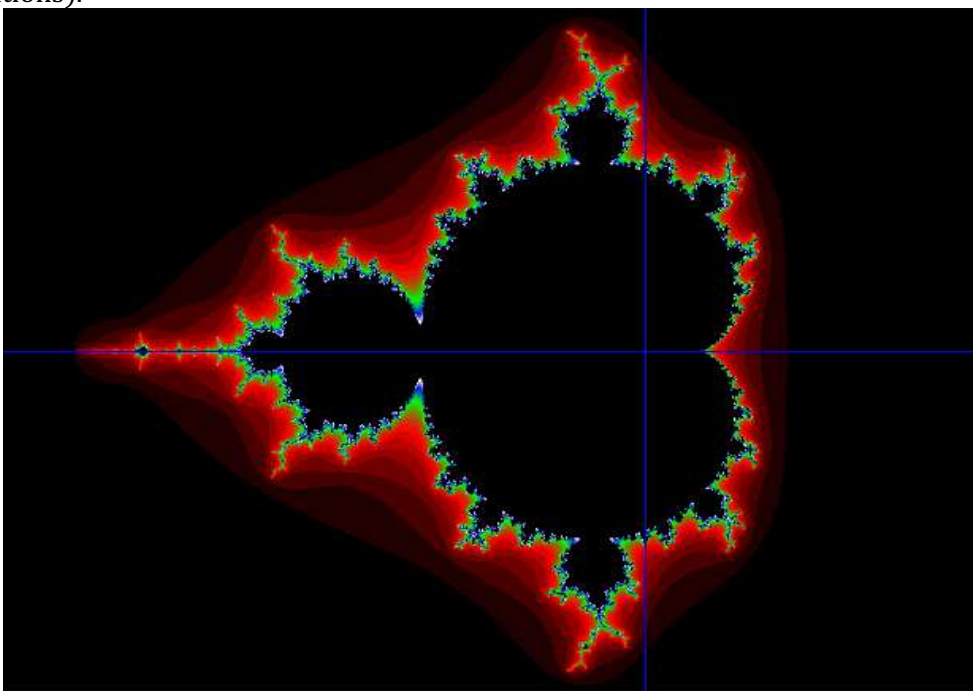

Figure 1. Computer generated picture of the Mandelbrot Set.

There exist ways to visualize the Mandelbrot set in 3D. One way is to rotate the Mandelbrot set around some axis, but that does not give a new complex structure. Another way is to set a height depending on when it tends to infinite, this though just helps exploring the 2D set and does not add a dimension to the fractal.

2.1.1 3D Mandelbrot, the Mandelbulb

There are several researchers around the world who are searching for algorithms to create real 3D fractals. The Mandelbulb is a 3D fractal that was found by Daniel White. [4]. Multiplication of to complex numbers can be seen as rotation and scaling of a point in 2D. When White discovered his algorithm he experimented with ways to extend these transformations from 2D to 3D and the equation he found is shown in equation 2.

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix}^n = r^n \begin{bmatrix} \sin(\theta * n) + \cos(\varphi * n) \\ \sin(\theta * n) + \sin(\varphi * n) \\ \cos(\theta * n) \end{bmatrix}$$

$$where:$$

$$r = \sqrt{(x^2 + y^2 + z^2)}$$

$$\theta = \text{atan2}\left(\sqrt{x^2 + y^2}, z\right)$$

$$\varphi = atan2(y, x)$$

Equation 2: The exponentiation rule that results in the Mandelbulb, founded by Daniel White.

In an interview with New Scientist White stated that no one has yet created a true 3D fractal [5]. Although he is almost there, the algorithm he discovered creates a 3D fractal with high details. But he admits that it is not quite real 3D Mandelbrot yet, "There are still 'whipped cream' sections, where there isn't detail".

2.1.2 The new fractal

The new fractal that this project is focused around is yet nameless. The fractal is defined with the same equation as 2D Mandelbrot (see equation 1) and the difference is the complex algebra.

The complex number that is used for this fractal contains 2 imaginary parts (see equation 3) and has a special way to calculate the square (see equation 4).

$$Z = a + bi + cj$$

Equation 3: Complex number with 2 imaginary parts.

$$Z^2 = \begin{bmatrix} a \\ b \\ c \end{bmatrix}^2 = \begin{bmatrix} (a^2 - b^2 - c^2) \\ 2 * sq * a\left(\frac{b^3}{|b|} - c^2\right) \\ 4 * \text{a} * \text{b} * \text{c} * \text{sq} \end{bmatrix}$$

$$where:$$

$$sq = \begin{cases} 1, & b = 0 \; AND \; c = 0 \\ \frac{1}{\sqrt{b^2 + c^2}}, & b \neq 0 \; OR \; c \neq 0 \end{cases}$$

Equation 4: Rule for calculating the square of the specific complex number for the new fractal.

When c is equal to zero in equation 4 we get the Mandelbrot set and by varying c we can see how the fractal changes from the Mandelbrot set into new complex structures.

2.2 Representing a 3D fractal as a Volume and visualize using Volume Rendering

A volume is basically several pictures stack on each other and are commonly used in applications for medical imaging [6]. A volume consist of several voxels (volume elements) [7] just like a picture consist of several pixels (picture elements). The difference is that a pixel can be addressed with two coordinates, x and y, which gives us two dimensions when a voxel is addressed with three coordinates, x, y and z, which gives us three dimensions.

In normal picture and volumes each pixels or voxels normally contains one (grayscale) or three (color) components. When working with fractals we could see these components as a representation of the part of the fractal contained in that voxel, a voxel that is completely outside or inside would have the value zero or one respectively. A value of a voxel that lays on the surface needs to be approximated depending on how much of the voxel that is inside and outside and what direction the surfaces cuts it.

Any computer can visualize a digital image without any need of installing a specific application but to visualize a volume is harder since different volumes need to be visualized in different ways. There are two main branches for volume rendering, direct volume rendering and indirect volume rendering. [8]

2.2.1 Indirect Volume Rendering (IVR)

IVR algorithms are used to extract ISO-surfaces from a volume. Marching Cubes [9] is an IVR algorithm that crates triangle models depending on a threshold value. Marching Cubes uses a cube that marching through the volume. The cubes 8 corners are 8 voxels from 2 adjacent slides in the volume. If the surface cuts through the cube one to four triangles will be created depending on how the surface cuts the cube. When the cube has marched through the volume and visited all possible sets of 8 neighboring voxels we have an array of triangles that can be render using normal rendering techniques. Since the algorithm creates up to four triangles for each cubed that is partially inside the number of triangles will increase heavily as the size of the volume is increased.

This is an algorithm that creates a triangle model depending on a threshold value. When the threshold value is changed, we need to run the algorithm one more time. This is very time consuming when exploring a volume with different threshold values.

2.2.2 Direct Volume Rendering (DVR) using Ray Casting

The main difference between IVR and DVR is that DVR does not create any polygons. Ray Casting (RC) is a DVR technique that effectively and efficiently can be implemented on the GPU. For each pixel on the screen a ray is cast, this ray cuts different voxels in the volume. Depending on the information in the sampled voxels a color is set to the pixel.

One way to implement RC is to use a cube as proxy geometry [10]. The proxy geometry is used to locate entry and exit point in the volume for each ray. Hadwiger et.al uses a cube where the color for each vertex is the same as the coordinate. The colors of the cube then represent the texture coordinate for the entry/exit point. When implementing this we need to use three rendering passes.

1. Render Front Faces
2. Render Back Faces
3. Ray-Casting.

The first two passes render the front faces and the back faces of the cube into two textures. The third pass is the actual RC algorithm. The two textures

containing the entry and exit points from the first two passes are sent to the GPU together with the volume. The last two passes can be done simultaneously and in that case we only need to render the front faces or back faces to a texture and render the other faces directly to the volume renderer.

2.2.3 Calculate gradient/normal for shading in Volume Rendering

To get 3D objects look as real as possible on 2D projections we need to shade the object depending on position of the light source and the surface normal. The most commonly used illumination method is Phong illumination [11]. In Phong illumination three types of reflections are used for each light, ambient, diffuse and specular (see equation 5). Diffuse reflection is calculated depending on the angle between the surface normal and the light direction and gives a clue about the shape of the object. Specular reflection depends on the angle between the lights reflection vector on the surface and the viewing direction and gives the object a shinier surface. The ambient reflection is a constant to prevent parts of the object to be all black which will appear where the angle between the light direction and surface normal is greater than 90 degrees.

$$C = C_a + C_d + C_s$$
$$C_a = L_a M_a$$
$$C_d = L_d M_d * \max(0, N \cdot L)$$
$$C_s = L_s M_s * \max(0, R \cdot V)^{\alpha}$$
$$where:$$
$$C = final\ color$$
$$C_a, C_d, C_s = final\ colors\ \ in\ terms\ of\ ambient, diffuse\ and\ specular$$
$$L_a, L_d, L_s = ambient, diffuse\ and\ specular\ color\ of\ the\ light$$
$$M_a, M_d, M_s = ambient, diffuse\ and\ specular\ color\ of\ the\ material$$
$$N = Surface\ normal\ vector$$
$$L = Light\ direction\ vector$$
$$R = Lights\ reflection\ vector$$
$$V = viewing\ direction\ vector$$

Equation 5: Phong illumination model

In volume rendering we need to calculate a normal for each voxel to perform Phong illumination. There are different ways of doing this depending on what rendering method we are using. When we have a static volume as described in section 2.2.1 a voxels normal can be created by calculating the gradient along each axis as shown in equation 6, where f(x,y,z) is the value of the voxel at position [x y z].

$$G = \left[G_x\ G_y\ G_z\right]$$
$$G_x = f(x+1, y, z) - f(f-1, y, z)$$
$$G_y = f(x, y+1, z) - f(f, y-1, z)$$
$$G_z = f(x, y, z+1) - f(f, y, z-1)$$

Equation 6: Gradient calculation on a static volume.

Since the voxels are either inside or outside the gradient on each axes can take three values, zero, negative or positive one which will not give us a very smooth shading. If we do a low-pass filtering on the volume before calculation the gradients we will get smoother normals but with the cost of loss of detail.

Another way to calculate the normal is to create a set of points $[x_i\ y_i\ z_i]$ that lay on the surface inside each voxel and calculate the least square plane using the linear model for multiple regressions defined in equation 7 [12]. By creating a

design matrix A and an observation vector $\overline{y}$ (see equation 8) from the points we can calculate the parameter vector $\overline{p}$.

$$A^T A\overline{p} = A^T\overline{y}$$

Equation 7: Linear Model for multiple regression.

$$A = \begin{bmatrix} 1 & x_1 & y_1 \\ 1 & x_2 & y_2 \\ \vdots & \vdots & \vdots \\ 1 & x_n & y_n \end{bmatrix}, \qquad \overline{y} = \begin{bmatrix} z_1 \\ z_2 \\ \vdots \\ z_n \end{bmatrix}$$

Equation 8: Design matrix and observation vector for Linear Model for multiple regression.

When we have the parameter vector $\overline{p}$ we can create a function for a plane (see equation 9). When calculating a normal we can set $\overline{p}_0$ to zero to get a parallel plane that passes through origin and easily calculate the cross product of two vectors as in equation 10.

$$f(x, y) = \overline{p}_0 + x\overline{p}_1 + y\overline{p}_2$$

Equation 9: Function for a plane for a parameter vector.

$$N = \begin{bmatrix} -\overline{p}_1 & -\overline{p}_2 & 1 \end{bmatrix} \quad \begin{cases} \mathrm{N} = \overline{\mathrm{v}}_2 \times \overline{\mathrm{v}}_1 \\ \overline{\mathrm{v}}_1 = [1\ 0\ \overline{\mathrm{p}}_1] \\ \overline{\mathrm{v}}_2 = [0\ 1\ \overline{\mathrm{p}}_2] \end{cases}$$

Equation 10: Normal calculation for a plane using function in equation 8 and $\overline{p}_0$ equals to zero (a parallel plane)

2.2.4 DVR optimization using Early Ray Termination and Space Leaping

When implementing GPU based Ray Casting as explained in section 2.2.2 we sample many voxels that contains non-important information which gives us a bottleneck effect in the fragment shader in the graphic pipe-line. Ruijters and Vilanova describe a couple of techniques to handle this and some other bottlenecks [13]. Two of these are early ray termination and space leaping using Octree. Early ray termination means that we only sample the ray until some criteria is reached, most common criteria is when certain opacity is reached.

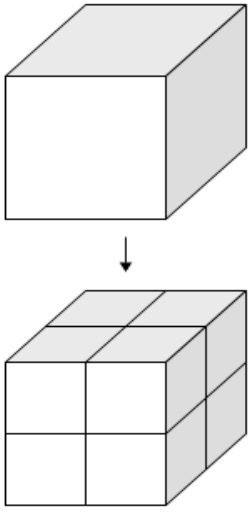

Figure 2: Subdivision of a Cube used in the Octree structure.

When using early ray termination we reduce the number of sampling points at the end of the ray. We also want to minimize the sampling points in the beginning of the ray. This can be accomplished by subdividing the proxy cube into an Octree structure. This subdivision splits a cube into 8 new cubes by dividing each side of the cube in half (see figure 2). By subdividing the proxy cube several times we can skip to render the Octree Nodes that does not contain any important voxels. This means that rendering passes 1 and 2 explained above will take a bit more time but we save time on GPU.

2.3 Stereoscopic rendering

Stereoscopy is a way to perceive real depth in planar images and displays [14]. The principle of stereoscopy is to make each eye to see its own picture of a scene, just like in the real world. When looking in the real world, the eyes see two slightly different pictures, if you would overlap these two pictures it would be a blurry image, but the brain can interpret the differences in the pictures and from that perceive depth. When looking on a normal picture like a photograph or a computer screen, there is just one picture for both eyes. If we instead would show two pictures simultaneously on the screen and in some way separate those pictures for each eye we could fool the brain to perceive depth in the picture. There are several ways to achieve this and some of these methods will be explained here.

2.3.1 Anaglyph Color Coding

Anaglyph is one of the oldest and cheapest techniques to accomplish stereoscopy [15]. The technique requires the viewer to use a pair of glasses shaded with different colors for each eye, commonly red and blue or red and cyan. The picture for the first eye is colored only with first color and the picture for the other eye is only in the other color. Since we need to color code the two pictures we will lose much of the color information.

2.3.2 Liquid Crystal Shutter Glasses

These glasses have a filter for each eye and when a filter is given a voltage it becomes opaque so no light comes through. These filters is synchronized with the frame rate of the screen and for each frame the glasses swaps between the eye, in other word each eye sees every second pixel [16]. This means that the frame rate needs to be twice as fast compared with other techniques, a frame rate above 100 Hz is preferred

2.3.3 Wavelength Multiplex Imaging

Wavelength Multiplex Imaging is a method developed by the German company Infitec [17]. The method uses two projectors with two different spectral filters. The spectral filters used filters away all light except light in specific spectral intervals. Each filter lets through some amount of blue, green and red light but from slightly different intervals. In figure 3 we see how these intervals could be defined.

When rendering you render the separated eyes pictures to separated projectors in full color. Using this method we do not lose any color information like with anaglyph and we do not need twice the frame rate like with Liquid Crystal Shutter Glasses, though we need to render the scene twice for each fram. A drawback is that it is much more expensive than Anaglyph or Liquid Crystal Shutter Glasses.

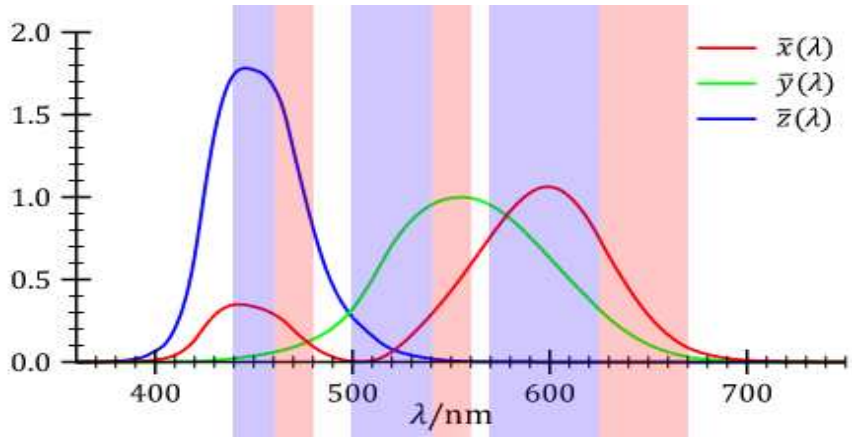

Figure 3. Intervals for Wavelength Spectral filter used in Infitec's Wavelength Multiple Imaging. (Blue intervals are for first eye and red is for second eye)

2.3.4 Off-axis rendering

When rendering the pictures for left and right eye with two cameras next to each other with one focus point we get two viewplanes. These two view plane are orthogonal to the viewing vector (see figure 4) which will result in some projection errors. To solve this problem we need to use off-axis rendering. Off-axis rendering enables the use of non-orthogonal viewplanes [18]. In OpenGL this can be done using the function `glFrustum(left,right,bottom,top,near,far)` where the first four arguments is the borders of the near clipping plane and the last two the distance from the camera to the near and far clipping plane.

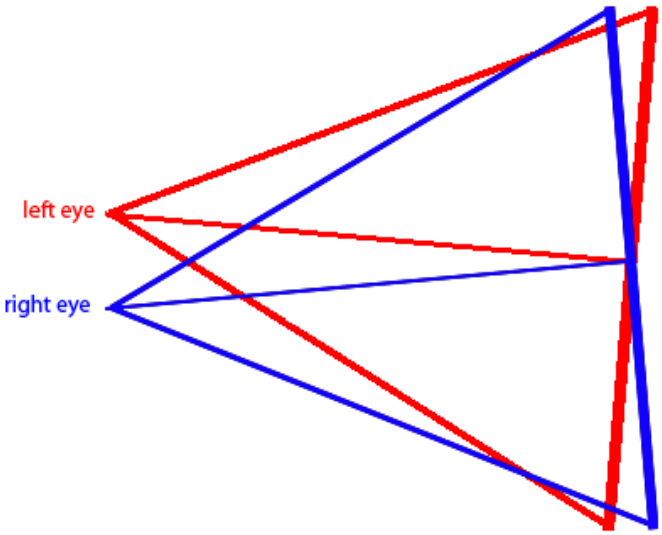

Figure 4: Viewplanes in a scene with two cameras pointing at the same point.

3 Method

During the project a solution with two applications was created. Both are developed under windows. A laptop with an AMD Turion X2 Ultra-Dual-Core Mobile ZM-82 2.80 GHz processor and 3.00 GB RAM has been used for development and testing. The graphics card that has been used is an ATI Radeon HD 3200, which is far from the best today. The first application, the real-time volume renderer (see section 3.2.1 and 3.2.2) are using OpenGL[19] and GLSL[20] to explore the fractals while the second application, the super sampler

(see section 3.2.3), is using c++ for creating an image buffer and DevIL[21] to save it to disk.
In the sections bellow I will explain these two applications and what methods are used to explore the fractal and create high resolution images.

3.1 Fractal calculations

Both applications that has been created are using a function to check if a point in 3D space is inside or outside of the fractal. This has been implemented so it is quite easy to add a new method. For each fractal there is a namespace which contains the function `inside([x y z],maximumIterations,power)`. In the function we simply check if the fractal is inside or not after that number of iterations and return a boolean value. If we want to explore another fractal we just have to create a new namespace that implements that function and use that namespace instead of the first.

3.2 Rendering methods

Like for all volumes there are several ways to visualize them and find interesting parts and information. Below I will explain those methods that has been implemented in the applications and what advantages and disadvantages there are with each method.

3.2.1 Real-Time Volume Render

A good start to get a good understanding of a dataset with volumetric data is in many cases to use a real-time volume renderer, as for the case of 3D fractals. A drawback with this method is when you want to display a volume with high details it affects the rendering time and since a fractal has infinite level of details we need to limit the details to display.

At the start up of the application a volume with a size around 500^3 voxels is created. For each voxel we calculate if it is inside or outside, if it is inside we set the value of the voxel to 1 or if it is outside we set it to 0. When having a volume like this we can calculate a normal for each voxel using the method explained in section 2.2.3. This volume is used as a 3D texture. The RGB value of the texture corresponds to the normal and since the normal vary between -1 and 1 and the RGB component can take values between 0 and 1 we have to add one and then divide each of the normal components. The alpha component of the texture corresponds to if that voxel is inside or not.

To render the volume we need to know the entry and exit point for each ray. This is done by creating an octree structure of the volume. In section 2.2.4 it is explained how to create an octree. For each node in the octree we save its coordinates, the color and the voxel coordinate. The top node is as big as the volume and consists of all colors. The color of the corner is set to the corresponding voxel coordinates and spans between [0 0 0] in one corner to [1 1 1] in the opposite corner. The color of the octree corresponds to the entry and exit point of the ray. To get the back faces to the volume renderer the octree must be rendered to a texture using a frame buffer object. The back faces will naturally be occluded by the front faces so we need to change the depth test function in OpenGL to greater-than instead of less-than.

In the volume renderer we sample along each ray with a stepsize around one divided by the size of volumes side (in this case 1/500). The length of stepsize affects the quality of the rendered image and the rendering time. A larger stepsize might introduce some sampling artifacts but the rendering time will

decrease. Depending on the speed of the GPU and the wanted quality the user can choose a step size that gives a satisfying result.

Since zooming is an important part in exploring 3D fractals and the volume is only around 500^3 voxels big it will not be many details if we do not calculates new voxels. Therefore a method has been implemented for zooming that uses a wireframe cube which you can scale and translate to fit an area of interest and then recalculate the voxels and gradients for the volume.

3.2.2 Cut plane

In section 2.1.2 it stated that we will get the Mandelbrot set when $c = 0$ (z-coordinate $= 0$). To prove this and explore it further a cut plane renderer has been implemented in my application to visualize different cut planes. The window is split into two equal sized viewports, in the left viewport a render the volume using the volume renderer described above. A transparent rectangular plane is added to the rendering from the volume render which represent our cut plane. In the right viewport a quad is drawn that is filling the whole viewport. Each vertex gets an attribute that corresponds to the associated corner of the cut plane. In the shader these coordinates are bilinearly interpolated for each fragment and are checked if it is inside or outside. If it is inside we set the pixel to black, else we set the pixel using a one dimensional lookup table using the number of iteration it took to tend to infinity divided by the maximum number of iterations.

On start up the cut planes corner coordinates are set to cut the volume at $z = 0$, which will result in the Mandelbrot set. The user can interactively zoom into the plane by scaling the corner points towards the center of the plane or translate it along the normal, side or up vector. The user can also rotate the plane using the mouse to explore many different cuts. This is done by translating the plane so the midpoint on the plane is in the origin and then we rotate each point using a simple rotation matrix for each axis.

3.2.3 Super Sampler to achieve high resolution images

As stated above when using a volume renderer for exploring fractals in real time we do not see so many details. Super sampling is a technique where several points is sampled for each pixel and then the average of this points is calculated as a pixel value [22][23]. Since we want to show beautiful high resolution images of our fractals a super sampler application has been created. This application uses a ray caster with orthogonal rays. To define the direction of the ray we can export a file from the volume renderer with viewing settings. The super sampler application creates a grid of 3x3 points for each pixel that moves along the rays towards the fractal. For each step the points are moved we check whether the center point is inside or not. When the center point is inside we need to finalize the position of each point, this is done by binary search for the surface. In other words we move each point in the direction of the ray if the point is outside and in the opposite direction if the point is inside, to come closer to surface we divided the stepsize by two for each step. From these nine points there are different ways of calculating the shading for that pixel. In the application four different methods of calculating the shading is implemented. The first is using the three points to create one big triangle (red triangle in figure 5) and calculates a normal by taking the cross product of the directional vectors. The first direction vector you get from subtraction the left point from the upper point and the second to subtract the right point from the lowest point. The second and third method uses eight triangles (blue triangles in figure 5) and calculates a normal for each triangle just like in the first method. The difference between method two and three is that for

method two the shading is calculate for each triangle and the average intensity is used as a pixel value when in method three a average normal is calculated and used for the shading. The fourth method using the least square method explained in section 2.2.3.

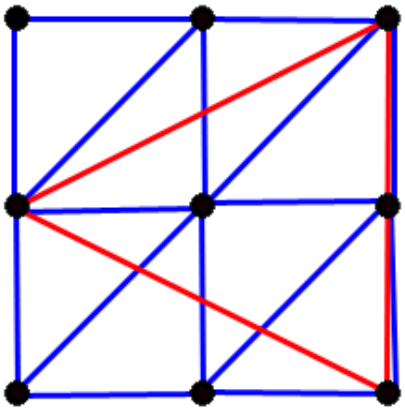

Figure 5: Triangle subdivision for normal calculation.

When doing the methods above we get high resolution images with very high details and to make these pictures look even more realistic I have added some shadows to the object. To achieve this I iteratively move the center point towards the light source, if we find a point that is inside we make that picture darker. By doing this the parts of the surfaces that are occluded from light will get darker.

3.3 Stereoscopic rendering

For the volume renderer there is an alternative for stereoscopic rendering. There are two methods for stereo, one for anaglyph rendering and one for horizontal split for two projectors.

3.3.1 Off axis-rendering

In section 2.3.4 the problem with orthogonal viewplanes when we want to render a scene in stereo is explained. To achieve correct stereo using non orthogonal view planes a function has been implemented that calculates a view frustum from a given position and in-world screen. To do this we need some parameters, the width, height and center of the view plane in scene coordinates and its normal and up vector. We also need the viewing position in the scene coordinates. The last two values we need is the front and back clipping distance measured from the view plane. When having all these values we can calculate the borders for the frustum using following steps: [24]

1. Calculate the view planes side vector by taking the cross product of its up and normal vectors. Also assure that these three vectors are normalized.
2. Calculate a vector from the center of the view plane to the view position.
3. Calculate the shortest distance from the view position to the view plane by taking the dot product between the vector from step 2 and the view planes normal vector.
4. Project the view position on view plane by subtracting the normal times the distance from the view plane from the view position.
5. We need to know the coordinates of the viewplanes top-right and bottom-left corners. To calculate top-right corner we add the viewplanes up and side vectors to the view planes center. The amount of the up and

side vectors that should be added is half the height and width of the viewplanes respectively. For the bottom right you use the same approach but use subtract instead of adding

6. Now we need to calculate the vectors from the projected view position to the two corners from step 5. This is done by just subtracting the coordinates for the projected view position from each corner. Let us call them v2bl and v2tr.
7. When having the two direction vectors from step 6 we can calculate the borders of the view plane.
    a. left = dot product between v2bl and the side vector of the view plane
    b. right = dot product between v2tr and the side vector of the view plane
    c. bottom = dot product between v2bl and the up vector of the view plane
    d. top = dot product between v2tr and the up vector of the view plane
8. We need to assure us that the near clipping plane is not behind the view point, this is done by comparing the front clipping distance with the distance from step 3
9. Since the borderer from step 7 is for the view plane and the values sent to the glFrustum function should be the borders of the near clipping plane we need to scale down the borders from step 7. To do this we multiply each value with the distance between the viewing position and the near clipping plane divided by the distance from step 3.
10. Last step that needs to be done is to make the view to be in the origin, this can be done by create an attitude matrix and invert it, see equation 11. Side, up and normal refers to the orientation vectors for the view plane and the viewPos refers to the original view position.

$$A = \begin{bmatrix} side & up & normal & viewPos \\ 0 & 0 & 0 & 1 \end{bmatrix}^{-1}$$

Equation 11: Attitude matrix to translate view position to origin

To set the view position at a correct distance from the view plane we need to measure the size of our screen and the distance our eye are from the screen. We do this by calculating a relation value by dividing the view plane width with screens width and multiply it with the distance from the screen. Now we can set the viewing position by adding the normal multiplied with the relation to the viewplanes center point. This will place the viewing position in-between our eyes. To make one view position for each eye we can use the relation value calculated above multiplied with half the interpupillary distance and move the view position for each eye that much to the side. The interpupillary distance is the distance between our pupils and the mean distance is 63 mm [25]. It does not matter what units we use to when we measure in real world as long as we are consistent. If you measure the screen in meters you have to give the viewing distance and interpupillary distance in meters as well.

3.3.2 Anaglyph

To separate the two pictures from each other and display them at the same time on the screen a chessboard pattern is used. When the scene is rendered for the first eye we check if the sum the pixels x and y coordinate, if it is even the pixel is color coded and if it is odd the pixel is discarded, for the second eye we do the same thing, but color code if odd and discard if even. The color for each eye is

determined by a uniform variable and are as a standard set to red (RGB = [1 0 0]) and cyan (RGB = [0 1 1]). To make the pixel color coded we just multiply the output color with the color code for that eye component wise.

3.3.3 Horizontal split for dual projectors

To render the scene for horizontal split we just need to divide the viewport into two equal sized viewports and place them next to each other horizontally. In the left viewport we render the scene from the right eyes view position and in the right viewport we render the scene from left eyes view position. (this depends on how the projectors is installed)

To display this correctly we need to set our computer to use extended desktop on the two projectors. We also my need to tell OpenGL to create a window that is the size of the extended desktop or else we might get problems when trying to maximize the window, normal windows is maximized on just one screen and so is also the case for OpenGL applications.

4 Result

The two applications that have been created both generate really nice and satisfying pictures. The Super Sampler application is rather slow but gives really nice pictures and the volume renderer has a quite low fps on my computer (see section 3 for hardware details). Below I will explain and show some pictures from different renderings done with the applications

4.1 Screenshots from the real-time volume renderer

The real-time volume renderer can create nice pictures of deep zooms. Figure 6 shows how the rendering looks when no zooming is preformed, it got a complex structure but do not have as much details as a fractal is expected to have. The application can handle zooming to magnification to around 2 million before the floating points loose to much precision. This can be seen in figure 7 where a magnification around 2.22 million is achieved. The center is located on the “right side” of the fractal. At this magnification we can see some details but some is lost due to floating point precision errors. If we try to zoom any deeper there will be errors in the depth buffer and the wire frame box will change its form. In figure 8 we can see a zoom with magnification around sixteen thousand with a center around [-1.7 0 0], which would be far to the left on the Mandelbrot sets real axis (x-axis) seen on figure 1. Here we can see more fractal structure than in figure 7 and also guess that there are some similarities along the z-axis.

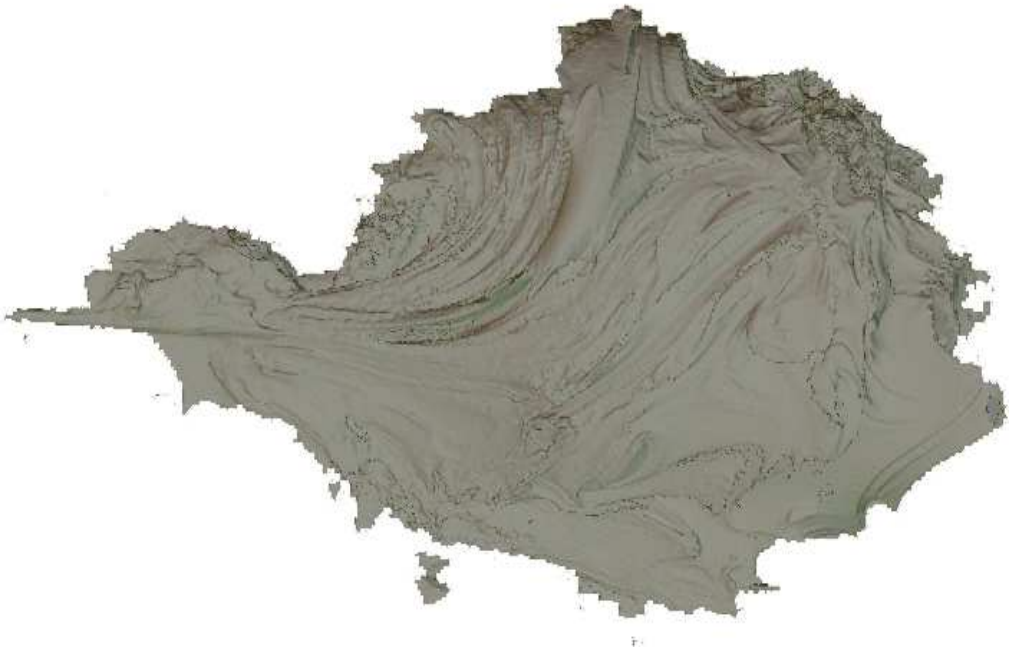

Figure 6: Rendering result from the real-time volume renderer

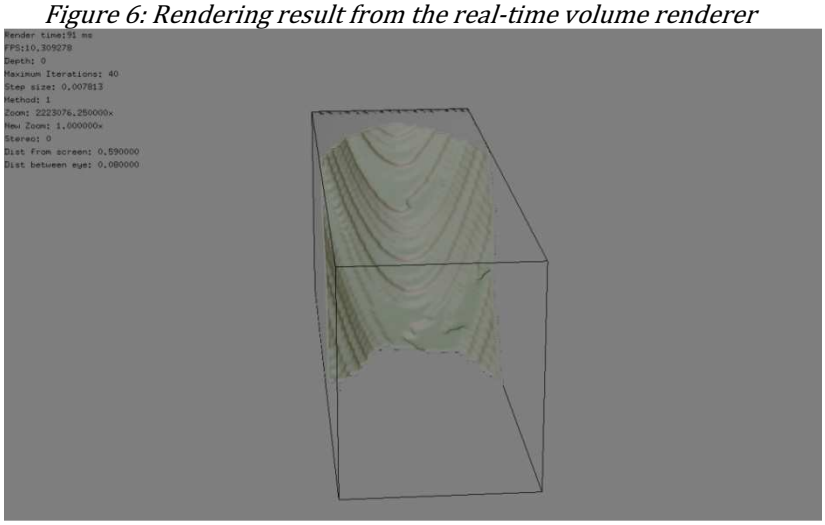

Figure 7: Successful zoom with a magnification around 2.22million.

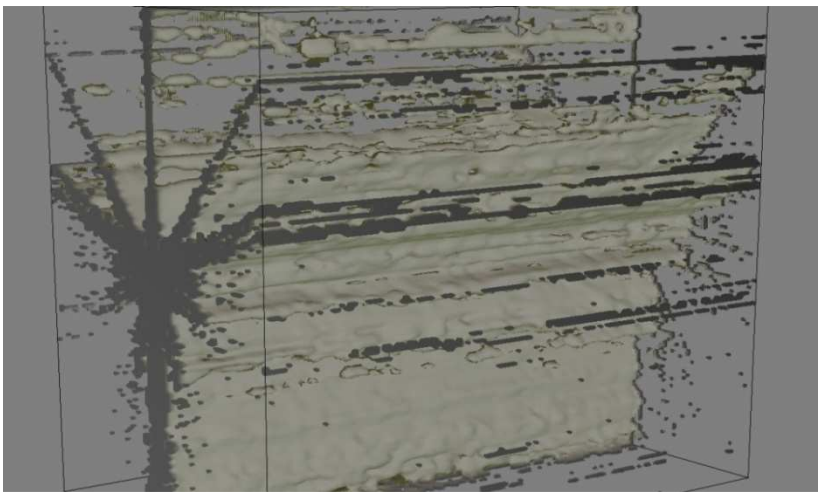

Figure 8: Deep zoom with a center around [-1.7 0 0] and a magnification on about 16 000.

4.2 Screenshots from the cut plane renderer

In figure 9 we can see the starting view for the cut plane renderer. On the left part of the picture we see the volume rendered with the ray caster with the cut plane added as transparent blue quad. On the right side we see the Mandelbrot set. The black part inside the colored areas is considered to be on the inside of the fractal, the colored area is colored depending on how many iteration that is needed before the point is tending to infinity, this number is mapped to a color using an 1D lookup table. The black area outside the colored areas is where the point tents to infinity really quick. By moving, rotating or scaling the plane we will interactively explore different cut planes in the fractal. In figure 10 we have placed the scaled cut plane at the surface on the "right side" of the fractal and can see as a small transparent quad representing the cut plane on the left side of the picture.

From figure 8 in section 4.1 we guessed that there are some similarities on the z-axis and figure 11 strengthen that guess. Figure 11 shows us two parallel cut planes with fixed y value, the top cut plane has its y value fixed at a positive y value and the bottom cut plane has a y value fixed to a negative value. Vertically we have the z-axis, horizontally we have the x-axis and the normal goes along the y-axis. Since we can see that the cut planes looks the same on both side of the x-axis and it still does that when moving along the normal we can believe that the fractal is mirrored over the x-axis.

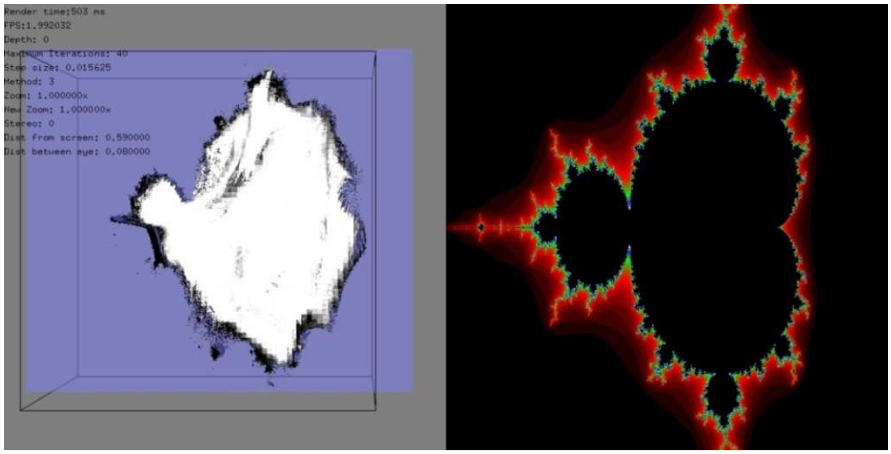

Figure 9: Start picture for the cut plane renderer

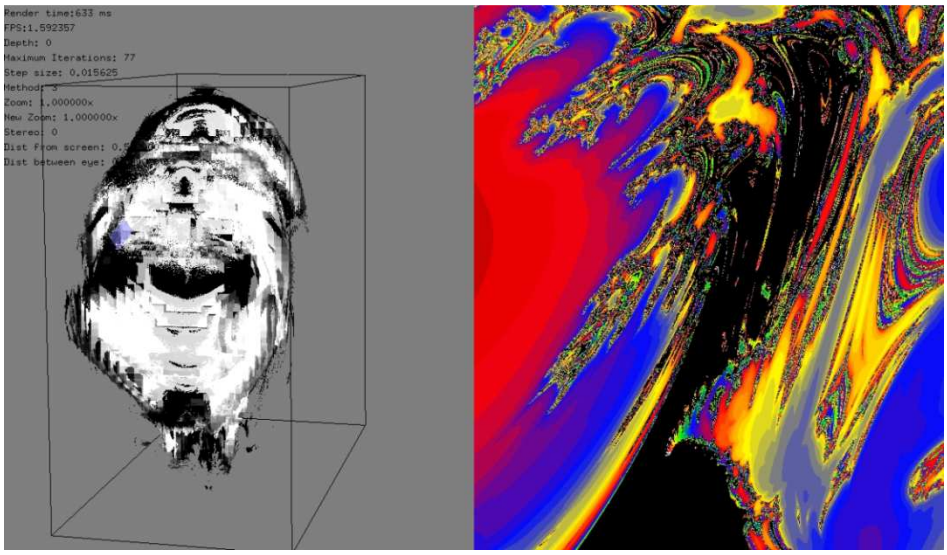

Figure 10: Zoomed cut plane, the transparent blue cut plane can be seen on the left side slightly above the middle.

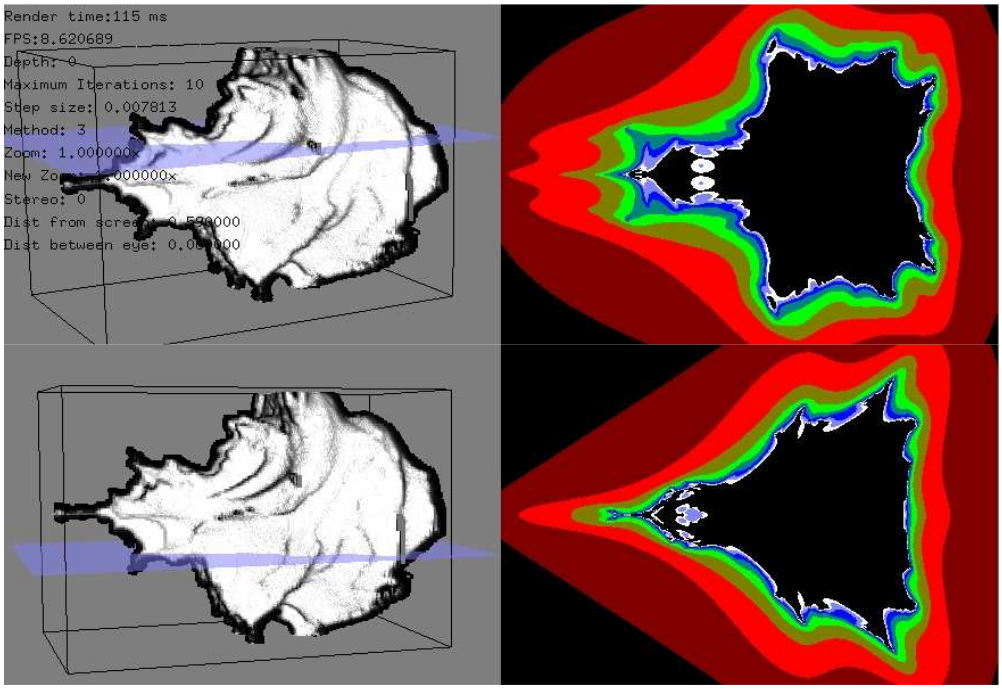

Figure 11: Two parallel cut planes with its y value fixed.

4.3 Screenshots from the Super sampler application

In the pictures above the level of details is not as good as we want to, but the rendering from the super sampler application gives a much more satisfying results. As explained in section 3.2.3 four different methods are used to calculate the shading for each pixel, in figure 13 these four methods are displayed. The two upper renderings are using method 1 and 2 and the two lower renderings are using method 3 and 4. In this report the difference is not that clear, but in the high resolution images we can see that there are some differences. Method 4, the least square plane method, produces an image slightly sharper than the other 3 methods, but we cannot declare that method as the best. The best methods would be the method that creates the nicest pictures but since people have different taste an image that is the nicest for one person might not be the nicest picture according to someone else.

In figure 14 we can see a rendering of the fractal rotated and seen from above and in figure 15 some coloring depending on the direction of the normal has been added. In figure 16 the formula has been slightly changed to use the power of eight instead of 2 (see equation 12). This can be done since taking power of 8 is the same as taking the power of two 3 times.

$$Z_{n+1} = Z_n^8 + C = ((Z_n^2)^2)^2 + C$$

Equation 12: Definition of a complex progression using the power of 8 instead of 2

As explained in section 3.1 it is easy to add new fractal formulas. To illustrate this I have added the formula for the Mandelbulb (see equation 2) and done some rendering with the power 8. Figure 12 show the result rendering.

4.4 Results from stereoscopic rendering

As explained in earlier sections there is an alternative for stereoscopic rendering in the application for real-time volume renderer. It creates nice stereo renderings of the 3D fractals when using the horizontal split method and off-axis rendering method as explain in section 3.3.1. When zooming in on the fractal when using stereo the details of the fractal becomes clearer and are easier to spot. It becomes easier to distinguish what parts that is closer or farther from the viewer. When rendering for anaglyph glasses the result is not as good as for the horizontal split.

To make the results as good as possible the user can interactively change the distance from screen and the interpupillary distance variables. This makes it easy to get a correct projection.

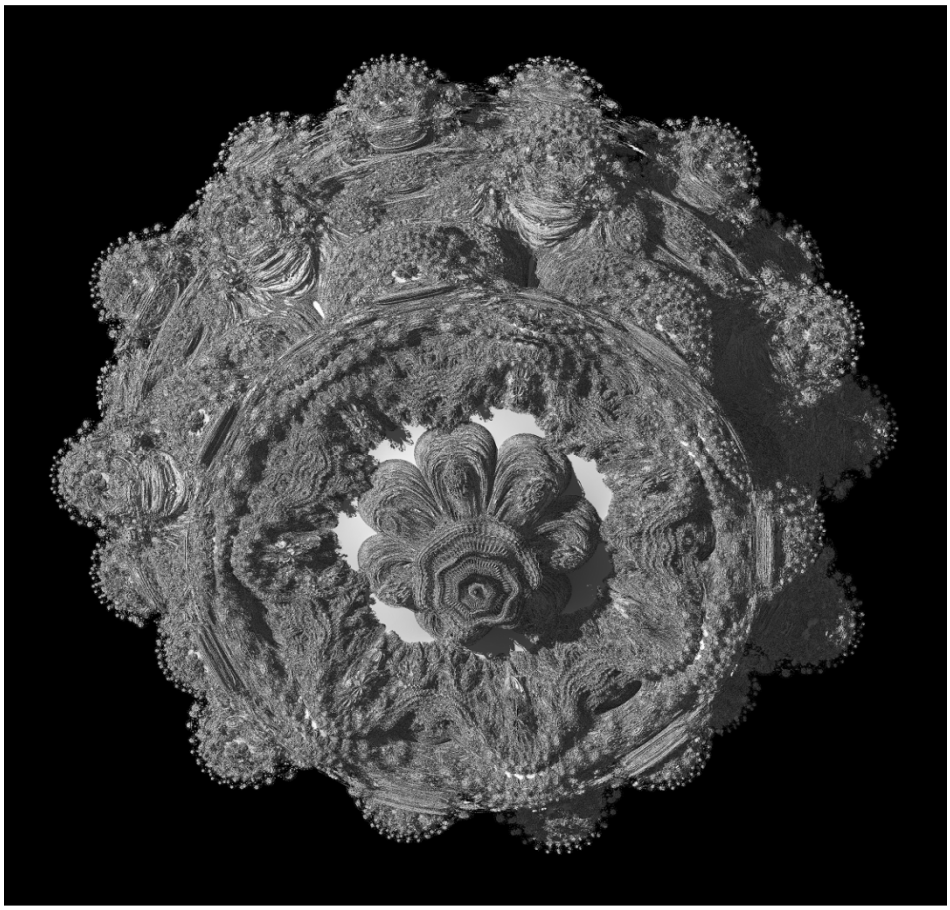

Figure 12: A super sampled rendering of the Mandelbulb using power of 8 (n = 8 in equation 2)

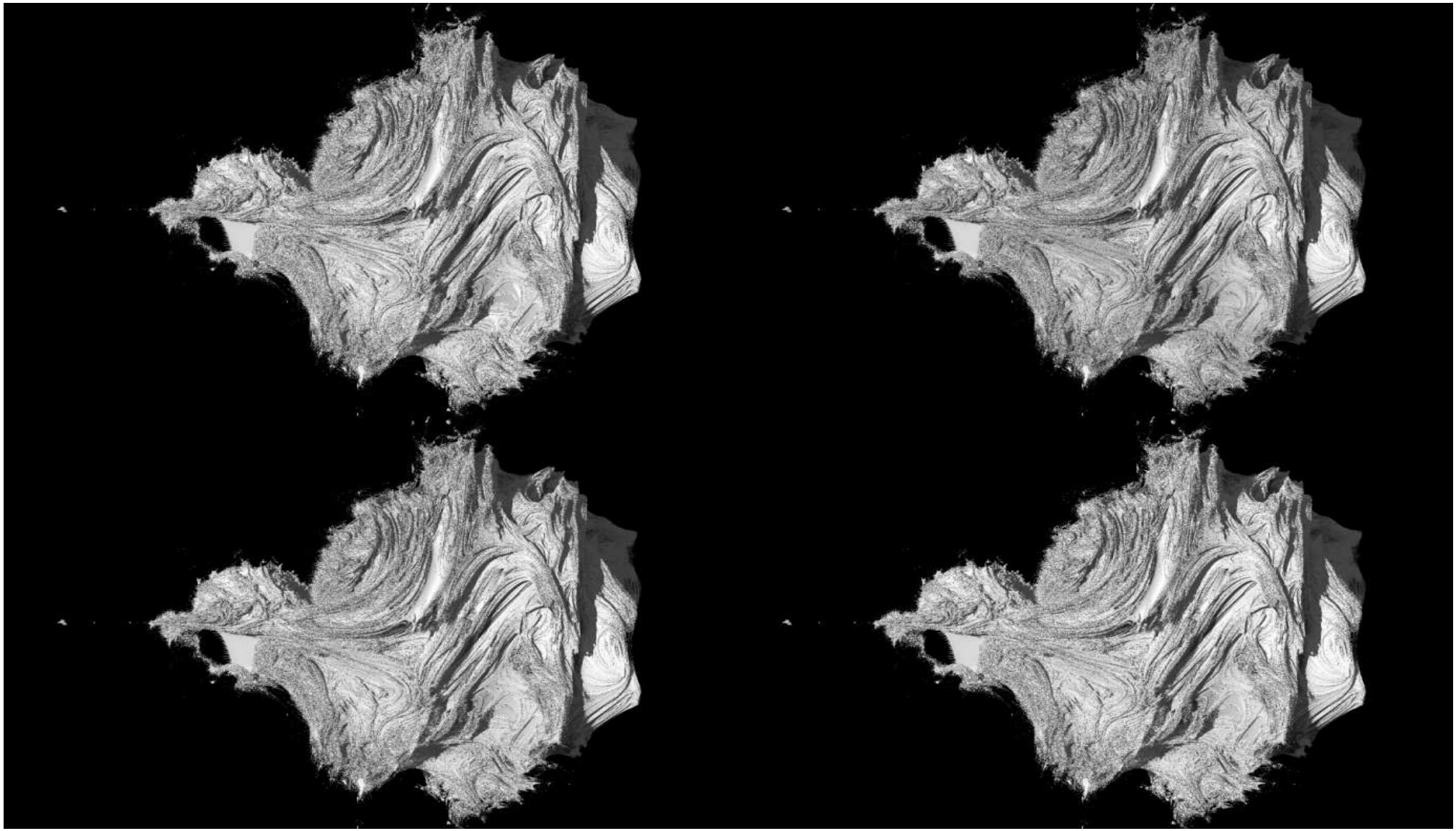

Figure 13: The new fractal with different methods for calculating normals.The upper left picture uses one triangle per pixel. The upper right picture uses eight triangles with normals to calculate averge light. The bottom left picture calculates the averge normal out of 8 triangles and the bottom right uses the least square methods to calculate best fit plane from nine points.

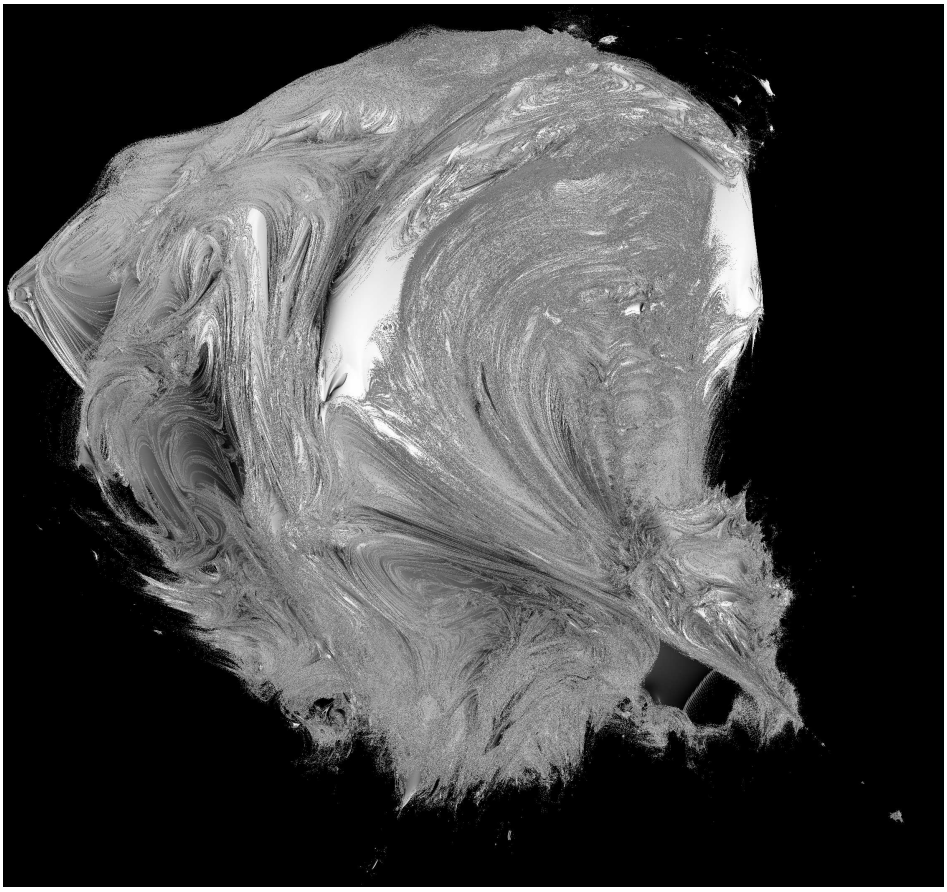

Figure 14: The new Fractal rotated and seen from above.

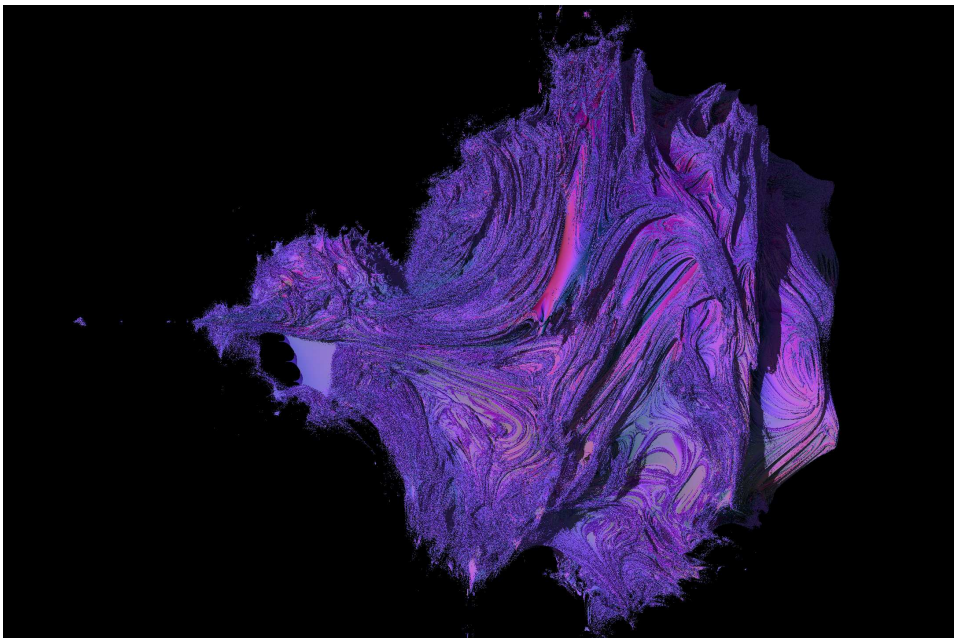

Figure 15: The new fractal with some coloring.

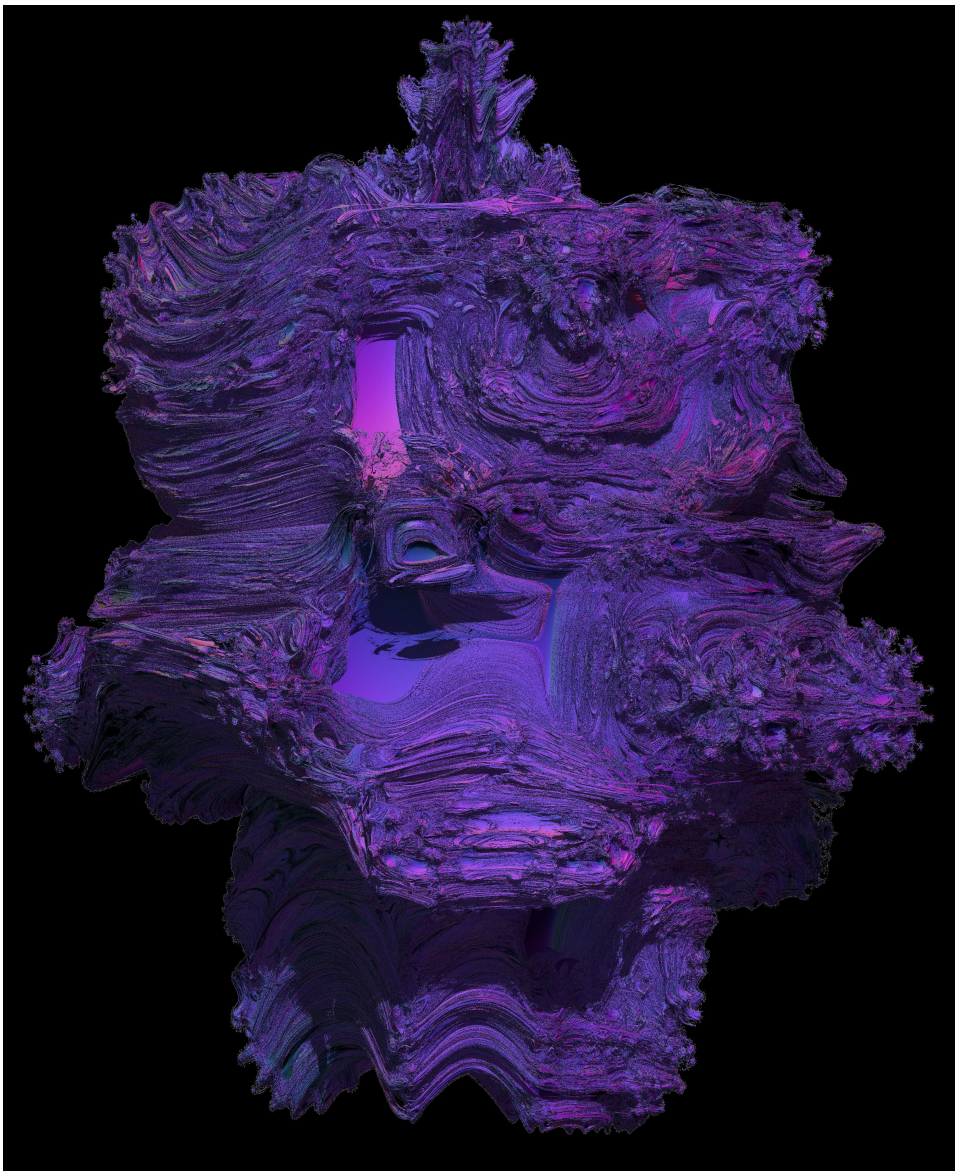

Figure 16: The new fractal calculated with the power of 8 instead of 2

5 Discussion

In the following sections I will discuss the results, what drawbacks there are with the applications and what improvements that can be done and ways to extend the applications.

5.1 Drawbacks, future work and possible improvements

The pictures created in the super sampler application looks really nice and have a very high level of details. But the time it takes to render them is quite high. An image with a resolution around one megapixel the rendering time is about 30 minutes. This is since the super sampler does not have so much optimizations. If we would want to create a animation where we are zooming in we would need hundreds or even thousands of images to be rendered it would take very long time. For 1000 renderings it would take about 500 hours, which is about 40 days. If we would want this movie to be in stereo, we would need to render twice as many images. Possible improvements to lower the super sampling time would be to use GPU based rendering instead on CPU based on a faster computer than the

one I used, and if we would want stereo we need to change from orthogonal projection to correct off-axis perspective projection to achieve good results.

It was decided to implement the super sampler as a standalone application since it takes so long time to render, so it could run on a secondary computer while working and testing the real-time volume render on the first computer. A good improvement to the applications would be to merge them into one application. By having only one application we could easier get super sampled pictures from the directions we want.

The super sampled images look very nice but they can become better, one way to improve would be to use more than nine points for each ray when calculating the least square plane, and use a stochastic spread of the points instead of the structured grid that is used now. But a drawback with adding more points is that it will take longer time to render.

In section 3.2.3 it is explained that shadows has been implemented, this is an important key to get perception of the depth of the super sampled images. The difference in depth perception between the renderings with and without shadows is quite big.

In the pictures (example figure 14) from the super sampler we can see two minor errors, the first is the "nose" to the left. To the left of the fractal we can see a couple of small objects that look like they are not connected with the fractal, but as we can see on the cut planes they should be connected, but the connection is so small so it gets missed. The other error is the hole in the left "globe" where the fractal is very narrow so it gets missed. One way to prevent this is to first render the Mandelbrot set to the image, this would fill out the empty space. But this would only work for the fractals that have the Mandelbrot set as a cut plane and with renderings with a view plane orthogonal to that cut plane.

Other improvements that could be done are the possibility to add more lights with colors and specific intensities. We could also place the object in some environment and implement environment mapping to make it look like a fractal shaped mirror.

A drawback with the real-time volume render application is that all interaction is done with the keyboard, except the rotating that is done by clicking and dragging the mouse. Since all actions have their own key it can be hard to remember what all keys do. One way to solve this would be to use some graphical interface like sliders, menus, textboxes etc.

The rotation of the cut plane is done by mouse and it is quite nice, though to get slightly better rotation an option for rotation around the axes, both the global axes and its own axes, could be implemented. This would best be done by adding some components to a gui.

The renderings of the fractal above do not use any special coloring, except those that use the normal as a color. There might be several ways to add color to them. We could use the residual error we get from calculation of the least square error on a lookup table to give some color. That would probably result in one color where the surface is flat and another color where the object curvy. Another way could be to map the distance between the entry point of a ray to its exit point, either in the direction of the ray of the opposite direction of the normal, in case we would choose to use the direction of the ray the color of the object would change as we rotate it and if we choose to use the normal as a direction the color would be the same in all direction. Yet another way to set the color could be to map the distance from the surface to another iteration layer. This distance could be measured along the normal, view direction or reflection direction to get difference results.

5.2 Comparisons with other works

It has not been possible to test any other applications for rendering fractals in general, though an application for exploring the Mandelbulb has been found at fractal forums [26]. Rendering an image with one megapixel in that application takes almost one minute, compared to the renderings from the super sampling application that application is 30 times faster. From this we can draw the conclusion that the application can get some major speed optimization.

5.3 Advantages and disadvantages with Stereoscopic rendering

By using correct stereoscopic rendering the visual perception of the fractals gets enhanced. As explained in section 4.4 it becomes easier to spot what is in the front and in the back. When looking at some zooms in mono you have to rotate and look from different views to really see all the details on the fractal but when using stereo this you spot this details much faster.

A big drawback with the horizontal split is that the viewport needs to be twice as big which results in twice as much rays which effects the frame rate. This part of the application is not tested on the laptop specified in the beginning of section 3 but on faster computer in a VR-laboratory where a satisfying frame rate was achieved.

6 Conclusion

What methods and optimizations exist for volume rendering and which methods suits best for fractal rendering?

One of the most suited methods for exploring 3D fractals in real time is ray casting with octree optimization (see section 2.2.2 and 2.2.4). The real time ray casting method visualize very few fractal details, this can be solved by adding a cut plane that visualize a cut the fractal with much more details compared to the image from the ray caster. These two methods both support zooming and rotation which is an important key in exploring a fractal to fully understand and explore its structure.

Indirect Volume Rendering with Marching Cubes algorithm (explained in section 2.2.1) was found to be not as good as ray casting since it takes long time to run the algorithm and the large amount of triangles created can be quite heavy to render.

What illumination model can be used to illuminate the fractal to create realistic visualization?

There exist a couple different illumination models and most of them can be used in volume renderers and therefore also for fractals. In the applications create during these project phong illumination model has been used with great results.

What method can be used to produce a high resolution image with good details for 3D fractal?

The ray caster and cut plane renderer is great methods to explore 3D fractals they do not give high detailed pictures. To solve this problem an application using super sampling was implemented that are sampling 9 points per pixel to find the orientation of the surface in each pixel. The surface orientation is found by calculating a least square error plane using the model for

multiple linear regression. In figure 12-16 the results of super sampled pictures is shown.

Which methods exist to create stereoscopic rendering and will stereoscopic rendering enhance the visual perception?

In section 2.3 a couple of methods for stereoscopy are explained and in the application for ray casting there is a possibility to render in stereo and real depth can be perceived by using anaglyph glasses or horizontal splited viewports on dual projector. On the dual projectors the visual perception is greatly enhanced when exploring the fractals and its easier for the viewer to determine what is in the front and what is in the back.

7 References

1. **Alan, Flook.** Fractals. *Sensor Review.* 1996, Vol. 16, 3, ss. 42-47.

2. **Hast, Anders.** Private communication. Gävle : February 2010.

3. **Mandelbrot, Benoît B.** Fractal asptects of the iteration if z $\rightarrow \Lambda z(1- z)$ for complex Λ and z. *Annals of the New York Academy of Sciences*. 1980, ss. 249-259.

4. **White, Daniel.** The Unravelling of the Real 3D Mandelbulb. [Online] 2009. http://www.skytopia.com/project/fractal/mandelbulb.html.

5. **Aron, Jacob.** The Mandelbulb: first 'true' 3D image of a famous fractal. *New Scientist.* November 2009, Vol. 204, Issue: 3736, ss. 54-54.

6. **Drebin, Robert A, Carpenter, Loren och Hanrahan, Pat.** Volume rendering. *Computer Graphics.* 1988, Vol. 22,no 4, ss. 65-74.

7. **Wikipedia contributors.** Voxel. [Online] June 3, 2010. http://en.wikipedia.org/w/index.php?title=Voxel&oldid=365396220.

8. **Lichtenbelt, Barthold, Crane, Randy och Naqvi, Shaz.** *Introduction To Volume Rendering*: Prentice Hall, Inc, 1998.

9. **Lorensen, William E and Cline, Harvey E.** Marhcing Cubes: A high resolution 3D surface construction algorithm. *Computer Graphics.* July 1987, Vol. 21, no 4, ss. 163-169.

10. **Hadwiger Markus, Ljung Patric, Salama R. Christof and Ropinski Timo.** GPU-Based Volume Ray-Casting with Advanced Illumination. *Europgraphics*, 2009.

11. **Wikipedia contributors.** Phong Shading. [Online] May 25, 2010. http://en.wikipedia.org/w/index.php?title=Phong_shading&oldid=364161526.

12. **Lay, David C.** Charpter 6 - Orthogonality and Least Squares. *Linear Algebra and its Application*. : Pearson Education, Inc, 2006, ss. 373-446.

13. **Ruijters, Daniel and Vilanova, Anna.** Optimizing GPU Volume Rendering. *Jornal of WSCG.* 2006, Vol. 14.

14. **Hast, Anders.** 3D Stereoscopic Rendering: An Overview of implementation Issues. *Game Engine Games: Volume One.* 2010.

15. **Wikipedia contributors.** Anaglyph image. [Online] June 3, 2010. http://en.wikipedia.org/w/index.php?title=Anaglyph_image&oldid=365107651.

16. **Burdea, Grigore C och Coiffet, Philippe.** *Virtual Reality Technology.* New Jersey : Wiley, 2003.

17. **Jroke, Helmut and Fritz, Markus.** www.infitec.net. *INFITEC.* [Online] 2003.

18. **Zelle, John M och Figura, Charles.** Simple, Low-Cost Stereographics: VR for Everyone. *SIGCSE '04: Proceedings of the 35th SIGCSE technical symposium on Computer science education.* 2004, ss. 348-352.

19. **OpenGL.** [Online] June 3, 2010. http://www.opengl.org

20. **GLSL.** [Online] June 3, 2010. http://www.opengl.org/documentation/glsl/

21. **DevIL.** [Online] June 3, 2010. http://openil.sourceforge.net

22. **Rost, Randi J och Licea-Kane, Bill.** *OpenGL Shading Language*: Addision Wesley, 2009.

23. **High-Resolution Antialiasing.** [Online] June 3, 2010. http://www.nvidia.com/object/feature_hraa.html

24. **Seipel, Stefan.** Seminar slides. *Advanced Visualization Course (VT10_23203).* University of Gävle 2010.

25. **Dodgson, Neil A.** Variation and extrema of human interpupillary distance. *Proceedings of SPIE.* 2004, Vol. 5291, ss 36-46.

26. **Mandelbulb 3D v1.42.** [Online] May 30, 2010. http://www.fractalforums.com/index.php?action=downloads;cat=5.