

University of Derby

College of Engineering & Technology

A project completed as part of the requirements for the

BSc (Hons) Computer Games Programming

Entitled

Dynamic deformations to volumetric data

By

Tomas Volkovickas

[tvolkovickas@gmail.com](mailto:tvolkovickas@gmail.com)

2014-2015

# Abstract

This paper investigates how dynamic interactions with volumetric data can be achieved using the latest 3D graphics library. In the past years the advances in graphics processing units made it possible to achieve a highly efficient volumetric rendering. However, the methods of the interactions with volumetric data are still relatively unexplored in terms of efficiency and how the programmable hardware can be used to speed up such process. One of the possible dynamic interactions is the deformation of the rendered volume. Such interaction will be the main focus of this paper. The results are analysed to measure the efficiency and visual quality of the proposed method of deformations. Finally, the paper concludes with what was found during the research and recommendations for further research are provided due to certain limitations of the explored method.

# Table of Contents

[1. Abstract 2](#_Toc417461371)

[2. Table of Contents 3](#_Toc417461372)

[3. List of Figures 6](#_Toc417461373)

[4. Introduction 7](#_Toc417461374)

[4.1. Volumetric Rendering 7](#_Toc417461375)

[4.2. Project Rationale 7](#_Toc417461376)

[4.3. Project Aim and Objectives 8](#_Toc417461377)

[5. Literature Review 9](#_Toc417461378)

[5.1. Introduction 9](#_Toc417461379)

[5.2. Smoke and Fog 9](#_Toc417461380)

[5.3. Medical Usage 10](#_Toc417461381)

[5.4. Data 11](#_Toc417461382)

[5.5. Volumetric Rendering Techniques 13](#_Toc417461383)

[5.5.1. Slice Rendering 13](#_Toc417461384)

[5.5.2. Marching Cubes 14](#_Toc417461385)

[5.5.3. Ray Casting 15](#_Toc417461386)

[5.6. Collision detection 17](#_Toc417461387)

[5.7. Collision responses 17](#_Toc417461388)

[5.7.1. Deformations in model space 18](#_Toc417461389)

[5.7.2. Deformation in Texture Space 18](#_Toc417461390)

[5.8. Real-time Deformations 19](#_Toc417461391)

[5.8.1. Geometrically based deformations 19](#_Toc417461392)

[5.8.2. Physically based deformations 19](#_Toc417461393)

[5.9. Conclusions 21](#_Toc417461394)

[6. Research Methodology 23](#_Toc417461395)

[6.1. Introduction 23](#_Toc417461396)

[6.2. Research Strategy 23](#_Toc417461397)

[6.2.1. Collision Responses 24](#_Toc417461398)

[6.2.2. Engine Structure 25](#_Toc417461399)

[6.3. Data Generation Methods 26](#_Toc417461400)

[6.3.1. 3D Chain Mail algorithm 27](#_Toc417461401)

[6.3.2. Data 27](#_Toc417461402)

[6.3.3. Modified 3D Chain Mail algorithm 28](#_Toc417461403)

[6.3.4. Limitations 28](#_Toc417461404)

[6.4. Conclusions 28](#_Toc417461405)

[7. Findings and Analysis 30](#_Toc417461406)

[7.1. Introduction 30](#_Toc417461407)

[7.2. Analysis 30](#_Toc417461408)

[7.2.1. Enhanced Chain Mail algorithm results 30](#_Toc417461409)

[7.2.2. Modified Chain Mail algorithm results 31](#_Toc417461410)

[7.2.3. Accuracy 32](#_Toc417461411)

[8. Conclusion and Evaluation 34](#_Toc417461412)

[8.1. Limitations 35](#_Toc417461413)

[8.1.1. Accuracy 35](#_Toc417461414)

[8.1.2. Collision detection 35](#_Toc417461415)

[8.2. Future work 35](#_Toc417461416)

[8.2.1. Collision Textures 35](#_Toc417461417)

[8.2.2. Interaction with other objects 35](#_Toc417461418)

[8.3. Closing Statement 36](#_Toc417461419)

[9. Bibliography 37](#_Toc417461420)

# List of Figures

[Figure 1 Structured volume data 11](#_Toc417461421)

[Figure 2 Unstructured volume data 11](#_Toc417461422)

[Figure 3 Slice Rendering 13](#_Toc417461423)

[Figure 4 Slice Rendering Example 14](#_Toc417461424)

[Figure 5 Marching Cubes Algorithm 15](#_Toc417461425)

[Figure 6 Ray Casting example 16](#_Toc417461426)

[Figure 7 Deformation in model space 18](#_Toc417461427)

[Figure 8 3D Chain Mail Algorithm 21](#_Toc417461428)

[Figure 9 Back and front case culling 24](#_Toc417461429)

[Figure 10 Ray Casting algorithm 24](#_Toc417461430)

[Figure 11 Texture coordinates offset algorithm 25](#_Toc417461431)

[Figure 12 Engine structure 25](#_Toc417461432)

[Figure 13 Rendering process 26](file:///G:\unrealsvn\Tomas%20Volkovickas1.docx#_Toc417461433)

[Figure 14 Deformation vectors 27](file:///G:\unrealsvn\Tomas%20Volkovickas1.docx#_Toc417461434)

[Figure 15 Enhanced Chain Mail tests 30](#_Toc417461435)

[Figure 16 Fps difference 31](#_Toc417461436)

[Figure 17 Modified Chain Mail test 31](#_Toc417461437)

[Figure 18 Fps difference 32](file:///G:\unrealsvn\Tomas%20Volkovickas1.docx#_Toc417461438)

[Figure 19 No deformations 33](file:///G:\unrealsvn\Tomas%20Volkovickas1.docx#_Toc417461439)

[Figure 20 Using 16x16x16 texture 33](#_Toc417461440)

[Figure 21 Using 32x32x32 texture 34](file:///G:\unrealsvn\Tomas%20Volkovickas1.docx#_Toc417461441)

# Introduction

In computer graphics the 3D objects can be represented using polygonal meshes. The usual data representation for such meshes is the list of points in three dimensional coordinates that are used to draw polygons. Compared with such representation of 3D objects, the volumetric rendering is using 3D scalar data to rendered a more complex real life objects (Engel et al.,2006). The importance of volumetric rendering relies within its possibility to render something that is difficult to model using modelling software. Some of the examples of such complex data includes scans of human body interiors or smoke effects as seen in real life.

## Volumetric Rendering

Volumetric rendering is the rendering technique to display a three dimensional scalar data as two dimensional projection (Engel et al.,2006). The models used for such rendering are usually difficult to create and certain techniques must be used to obtain data required. There are two main ways of obtaining such data: scanning and data generation. The scanning example usually involves computerized tomography (CT) or magnetic resonance imaging (MRI) (Engel et al.,2006). This way real life objects can be scanned and their depth values sampled and turned into volumetric data used for rendering. Data generation examples includes data from computational fluid dynamics (Engel et al.,2006). This type of data is particularly difficult to model. Real life examples of such data include such effects like smoke or dust clouds.

Once the data has been acquired there are quite a few ways of actually render it. Some of the example includes “Slice rendering”, “Marching cubes” and “Ray Casting” techniques. The main idea behind slice rendering is creating a number of slices from sampled data and then render it using alpha blending (Engel et al.,2006). The marching cubes algorithm relies on creating a polygonal mesh from a scalar data (Sio et al., 2011). Ray casting is one of the latest techniques to render the volumetric data and it uses fragment shader to cast rays and sample the colour of each pixel. All of these techniques are further discussed in 5.5.1, 5.5.2 and 5.5.3 sections.

## Project Rationale

Volumetric rendering is a very important technique to render complex objects such as scans of human body or effects like smoke and fog. There are numerous techniques to render such data and a lot research has been done to improve the efficiency of such techniques. However, the collision detection and responses remain a relatively untouched area and certain methods must be explored in order to improve this process. The importance of such dynamic interactions can be found in computer games industry where volumetric smoke effects can add an additional experience to the player. Another example would include the use in medical applications. Volumetric rendering has already proved to be of great importance in research of various diseases and training (Meißner et al., 2000). However, the efficient collision detection and responses could further improve it’s the benefits.

## Project Aim and Objectives

The aim of this work is to develop efficient and realistic dynamic deformations of volumetric data. In order to achieve this aim, the following objects will be met:

1. An extensive literature review will be conducted into volumetric rendering techniques to determine the limitations and successes of prior work.
2. The existing methods of volume deformation will be reviewed and possible improvements will be considered.
3. A demo application will be designed and implemented to test and capture data
4. The data gathered from running the application will be analysed to draw the final conclusions of this work.

# Literature Review

## Introduction

This literature review is the review of existing applications for volumetric data. The two most popular examples are presented in sections 5.2 and 5.3. The first example reviews how volumetric rendering can enhance the quality of the visual effects in computer games or other 3D graphics applications. The second example reviews how such rendering has been used in medical applications. In section 5.4 the two methods of data gathering are discussed and analysed. The following sections 5.5.1, 5.5.2 and 5.5.3 describes some of the volumetric rendering techniques. Each technique is analysed and potential disadvantages are being addressed. The last few sections review the current ways to check for collisions and the possible implementations of their responses.

## Smoke and Fog

In computer graphics the effects such as smoke or fire are being used to create a more realistic 3D environment. In order to visually represent the smoke a system capable of rendering millions of small particles is required to mimic the way smoke is created in real life. One of the earliest techniques to achieve this was billboarding (Larsson, 2010). Billboarding has been used extensively in older computer games due to it being less computationally demanding. However, the visual effect achieved using such technique is usually of a poor quality and not very convincing. To achieve a more convincing effects soft particles, mega particles or fluid simulation must be used.

To render a smoke using soft particles a large number of sprites is being rendered in the 3D scene. However, once the particles start to intersect with the other solid objects in the scene the unnaturally sharp edges start to be visible and creates a very fake looking effect similar to billboarding technique (Lorach, 2007). To remove such edges a multiple pass rendering must be used. In the first pass the 3D scene is being rendered without smoke particles and its depth values are being stored in the texture. Once the smoke particle is being rendered its depth is compared to the value saved in the depth texture to decide the alpha value of the final colour value for the smoke particle. The final effect can be quite convincing but it does not look as realistic as fluid simulation. The lack of density in smoke makes it ideal for environmental fog or less visible smoke effects where realistic looking effect is not required.

Mega particles can be used to achieve the volumetric looking smoke without the volumetric rendering technique (Larsson, 2010). Instead of rendering the particles to the scene, they are rendered into the texture which is then applied to the scene. However, the biggest drawback of such technique is the effect is still being just the texture. It does not have any volume to it and stepping inside into the smoke would require addition post processing to make it look more believable.

Fluid simulation is one of the main techniques to achieve a realistic looking smoke. In order to do it, a volumetric smoke can be rendered onto the scene. Such smoke occupies a volume and can accurately represent the desired effect similar to one in the real life. To make the volume more realistic raycasting technique can be used to shoot rays through the cloud and help to decide the final colour value for each particle (Stopford, 2006).

## Medical Usage

Volumetric rendering can also be used for applications such as cancer detection and surgical planning (Meißner et al., 2000). To correctly display such data it needs to be sampled first. The main techniques for doing so involves MRI, CT or ultrasound. Once the certain object is being scanned the data can be stored as three dimensional arrays and then used for volumetric rendering. The key element during this process is the ability to view the inner objects or surfaces of the scan on two dimensional image.

X-ray Computer Tomography or CT scanners can provide data that is normally used to show a scan of parts of human body. Such scans can be imagined as a stack of two dimensional images that all share the same resolution and can be displayed one at the time using the conventional rendering methods. However, the volumetric rendering allows to display such data as a single volume exposing the whole scan in a real time and give the ability to view it as a three dimensional image. This also allows the rotation being applied to the image making the user to be able to adjust the viewing angle of the scan.

## Data

The data used in volumetric rendering can be given in a well-structured Cartesian rectilinear grid or in a completely unstructured grid. The structure of data is highly dependent on the type of data source. The data sets from computed tomography (CT) are well structured and can be imagined as a cube containing multiple voxels (Rezk-Salama, 2001).

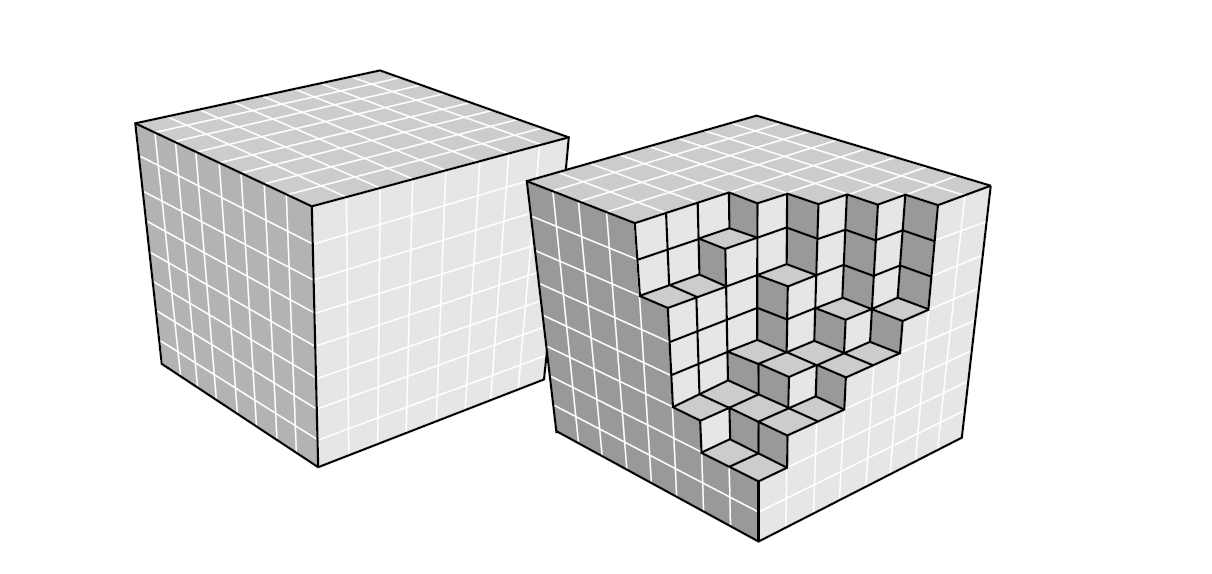


Figure 1 Structured volume data (Rezk-Salama, 2001)

Each voxel is a data unit in three dimensional array and may contain a grey value of the CT scan. One of the drawbacks of CT scans is that they only hold a single value for object visualization. Therefore, such data set is normally provided together with transfer functions. Transfer functions can be a one or multidimensional data used to map values of volume data in order to add some optical data such as colour or opacity to the voxel (Kindlmann, 2002). Transfer functions (TFs) can be represented as one dimensional textures that hold RGB or RGBA values of the scan and by mapping the coordinates of volume data the final colour value can decided while rendering the data set.

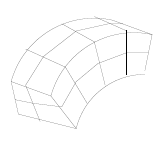


Figure 2 Unstructured volume data

However, some datasets are not obtained by sampling real world object, but can be pre-generated. These data sets are normally used for rendering smoke, clouds or even fur. Such objects are quite difficult to model and normally requires a certain noise function applied to them to simulate how they look like in real world. One of the biggest advantages of precomputed volume data is its ability to contain multiple constant values in each voxel. During the CT scans the only value available is the density of the given point. However, the precomputed data can hold data such as density, temperature or opacity in each element (Gorkin, 2009). This is a very important feature when rendering effects such as smoke, because they are normally not static and can be affected by other elements such as wind direction. Once the volumetric data has been generated, it can be adjusted using noise functions such as “windowed noise” or “Pyroclastic” (Wrenninge and Bin Zafar, 2011).

Figure 4 Pyroclastic noise (Wrenninge and Bin Zafar, 2011).

Figure 3 Windowed noise (Wrenninge and Bin Zafar, 2011).

The sample application used with this paper will be using a data provided by CT scan. Data sets can downloaded from <http://www.gris.uni-tuebingen.de/edu/areas/scivis/volren/datasets/datasets.html> and are all in binary format (Meissner, no date). The RAW file is the most common format for such data. It normally contains density values in three dimensional coordinates.

Once the data has been loaded into the memory in can be visualised as a stack of 2D pictures. Using the two dimensional textures approach the correct slice would need to be used for texture sampling. However, the sample application has taken advantage of 3D textures available in Direct3D API. The 3D textures can be created using the data that has been read from data set provided and then x, y and z values can be used for sampling it.

## Volumetric Rendering Techniques

### Slice Rendering

Slice rendering is one of the easiest ways to render volumetric data. To render in such way a stack of two dimensional textures are being rendered every frame in back to front order. The application also needs to use alpha blending in order to see through all slices and create volumetric effect (Engel et al., 2006).

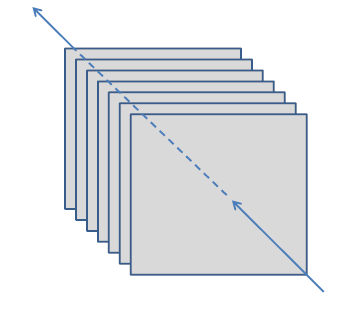


Figure 3 Slice Rendering (Engel et al., 2006)

This way of volumetric rendering is easily achievable but suffers from a one major flaw. The image only appears to be correct if rotation is not being apply to the object. Because the image is just a tack of multiple textures, it does not have any depth to it and cannot be viewed from different angle. To solve such issue the constant spacing can be applied to each texture to create gaps and make the volume to appear tree dimensional.

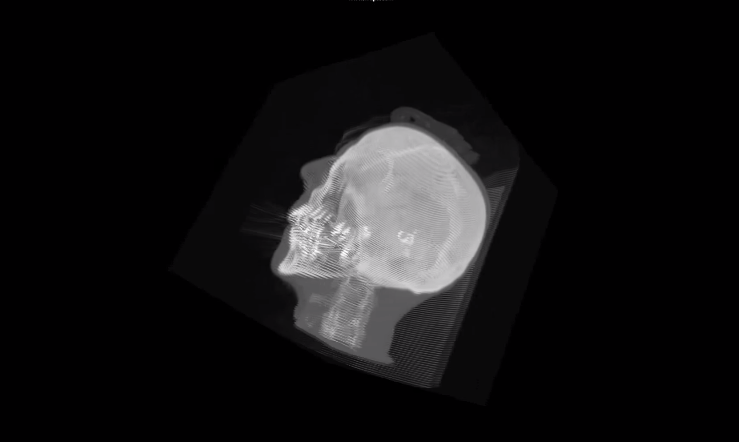


Figure 4 Slice Rendering Example

However, the final image still suffers from tiny amount of spacing and the gaps between individual layers can be easily seen.

### Marching Cubes

Marching cubes algorithm is one of the oldest techniques to render the volumetric data. It was proposed for the first time in 1987 by Lorensen and Cline (Sio et al., 2011). The main idea behind this algorithm is that the data is being used to create polygons inside predefined three dimensional space. The model can be rendered using a cube as a bounding volume for model. At first step the bounding volume needs to be divided into required number of voxels. Each voxel then acts as a smaller bounding volume where each vertex can hold the volume’s data. By going through each of 8 vertices the polygon can be constructed depending on data each vertex holds.

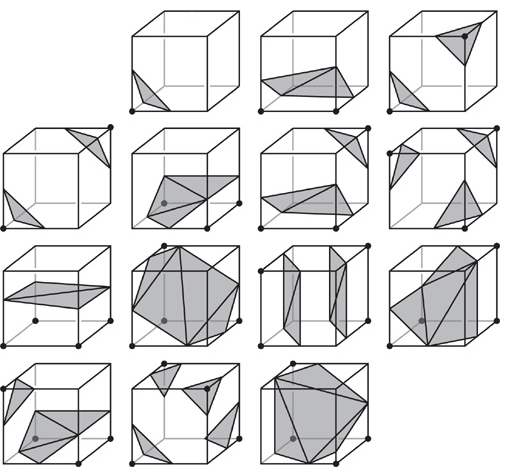


Figure 5 Marching Cubes Algorithm

The main problem with this algorithm is the actual creation of polygons. During the creation of polygons the position are being calculated using binary decisions and interpolation, and the final image might not represent the scanned object to the tiniest detail. Also using marching cubes algorithm is impossible to render a real thickness of different object layers therefore highly transparent objects can look confusing without rotating the object first. (Bartz and Meißner, 1999).

### Ray Casting

The main idea of ray casting technique is to trace rays that are shot from camera point trough volume and sample the final colour of each pixel (Engel et al., 2006). The advantage of ray casting technique is that it solves a lot of problems discussed in sections 5.5.1 and 5.5.2. Comparing with marching cubes algorithm there is no need to deal with generating triangular meshes and the object is rendered accurately by shooting rays at every pixel. Ray casting is also highly flexible in terms of optimizations. When rendering model using slicing based approach, the implementation is limited to rasterization process, where in ray casting technique numerous ways of optimization such as early ray termination is available (Engel et al., 2006).

The actual ray casting technique has been around since 1980s (Engel et al., 2006). However it was limited to CPU implementations and only in 2003 the first implementation of GPU based ray casting has been published. One of the reason of such move to GPU implementations was the advances of fragment shader in programmable hardware. Before the programmable graphics hardware the applications were relying on fixed function pipeline. Using such architecture the volumetric rendering was not possible on GPU. However, the introduction of vertex and especially fragment shaders gave a possibility to perform a ray tracing. The reason why fragment shader is so important is because the final colour of the pixel is calculated at this stage. The fragment shader allows to perform crucial operations such as texture sampling and for loops required by ray casting technique. The following example is the screenshot of model rendered using ray casting method:

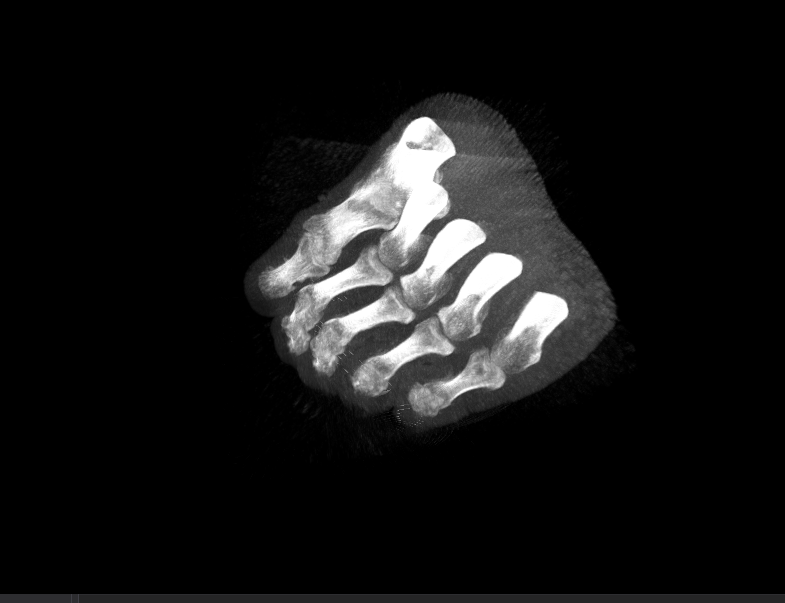


Figure 6 Ray Casting example

## Collision detection

The main collision detection techniques can be divided into two approaches: object space and image space (Jang, Jeong and Han, 2007). Object space collision detection is normally performed using bounding volumes and algorithms like AABB overlap test to detect the occurring collisions. However, due to the potentially large amount of voxels in volumetric data, such approach would be very inefficient.

Another approach for detecting the collisions is the image space approach. This way the depth and stencil buffers are used to measure the volumetric distances between the potentially colliding objects. Iain Cantley has proposed a technique to detect the collision detection for large number of particles (Cantlay, 2007). Effects like a large volumes of smoke that has a lot of density to it can be a very expensive to render, so testing for collision in image space can results in a lot more efficient processing.

The whole process can be broken down into three main stages: rendering scene without effect, rendering smoke into off-screen buffer and displaying the final result after the depth test. During the first step the scene is being rendered and the depth values are being stored in depth buffer. The smoke volume can then be rendered into a custom geometry buffer and collision test can be performed with the values from depth buffer. At the final step the volumetric effect can be composited back to the main frame buffer with the collisions being resolved. The proposed technique has been described using a large particle system, but the same can be applied to the effects produced using volumetric rendering.

## Collision responses

Collision response occurs after one or more objects collide in the rendered scene. The actual type of response is highly dependent on the nature of the colliding object. If the object is solid the response depends on its current velocity and mass. Based on those two values the object can remain stationary or be moved. However, in some cases the colliding object needs to be deformed when collision occurs. One of the examples is the smoke or dust colliding with solid objects in 3D scene. When such collision occurs the volume needs to be deformed because the particles in smoke or dust cloud gets only partially absorbed. To create such deformations there are two main methods: deformation in model space and deformation in texture space (Engel et al., 2006).

### Deformations in model space

When performing the deformation in model space the actual changes needs to be performed to the geometry of the model. One of the ways to do it is by displacing the vertex coordinates of the model to change its shape (Engel et al., 2006). When model is being rendered in the scene, one or more textures are applied to each polygon. This process is called “Texture Mapping” (Heckbert, 1989). During the texture mapping each vertex in polygon holds uv coordinates of the texture and during the rasterization process the colour value of texture is applied to the polygon. To perform a deformation one of the vertices can be displaced in model space while the texture coordinates remains the same. The following picture illustrates such deformation (Engel et al., 2006):

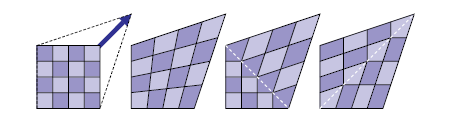


Figure 7 Deformation in model space (Engel et al., 2006)

Such deformation approach can be very suitable for slice rendering volumetric rendering technique. During the slice rendering the 3D texture is being sliced into multiple planes depending on the data sets depth. The resulting planes are then mapped onto the same number planar polygons. To perform a deformation one or more of these planes can be deformed by moving their vertices in 3D coordinates. However, one of the problems of such approach is slices becoming non-planar after the vertices movement (Engel et al., 2006). Also such deformation is dependent on how each slice is built. There are two main approaches of building slices: quads and triangles. Using a single quad or just two triangles greatly limits the options for deformation. To solve such issue the slice could be tessellated into additional triangle therefore allowing the more accurate deformation.

### Deformation in Texture Space

The alternative to deformation in model space is the deformation in texture space. Using this method the geometry of the rendered object remains the same and only the texture coordinates are being displaced (Engel et al., 2006). This deformation is possible because the 3D texture used to look up the depth values actually defines the shape of the object and by changing those values, the object can be deformed. The main disadvantage of such method is the requirement of additional layer of complexity in order to achieve a greater results. To create a more realistic deformation the geometry model used for volumetric rendering needs to be subdivided into a smaller parts (Engel et al., 2006). Once the subdivision is performed it creates a larger number of vertices therefore allowing a greater diversity in choosing the deformation points. One of the ways to improve such process is to use fragment shader for offsetting the texture coordinates (Engel et al., 2006). The method uses an additional texture to hold the offset values that could be used when sampling the 3D texture for final pixel colours.

## Real-time Deformations

Sections 5.7.1 and 5.7.2 has reviewed the two possible locations for deformations: model and texture spaces. During the static deformation the object is deformed only once and the actual deformation happens before the object is rendered. However, in most cases the object needs to be deformed in real time. Some of the examples include a smoke or dust clouds. Real time deformation can also be used during the collisions where the shape of the object needs to be changed. All deformation techniques can be split into two groups: physically based and geometrically based (Dean, 2004).

### Geometrically based deformations

Geometry based deformations allows user to deform the object by directly manipulating with it. One of the examples of such deformation is Free Form Deformation (Dean, 2004). The main idea behind such technique is the certain control points placed in the same space as the objects allowing the direct interaction with it. Every control point need to be linked with certain parts of the object it is interacting with. There are usually no forces or other physics elements applied to interaction so the object can be deformed in any way it is required. Such deformation is perfect for modelling applications where the object needs to be sculpted or its shape needs to change in real time without any collisions applied. However, it is not very suitable in case where deformation is a direct response of the collision between two or more objects.

### Physically based deformations

Physically based deformations allow the object being deformed by trying to mimic its movement and physical properties found in real life examples (Dean, 2004). The usual way to deform the object is to apply certain force to one or more points of the object’s surface and let the physics model to perform the deformation based on certain properties (Jacob, 2008). Some of the example of physically based deformation techniques include: Mass-Spring, Finite Element Model and 3D Chain Mail technique.

The main idea behind the Mass-Spring technique is that every mass point is connected by damped springs (Jacob, 2008). Once the force has been applied to one of the points, the force is being propagated trough the object by the springs (Drager, 2005). Such technique can be easily implemented, but suffers from few flaws. First of all, due to the force being transferred with the use of springs the deformation might not look right if the deformed object is of rigid type. Therefore, such transformation is ideal in use of cloth simulation (Drager, 2005).

In Finite Element Model the object is first divided into simple elements like triangles (Jacob, 2008). Each of the elements can hold a various information such elasticity and strain. During the deformation the Finite Element Model is a lot more accurate then Mass-Spring. Due to the large number of computations required for such method it is also a lot more expensive. Even if the results can be very accurate, the computational cost for running such simulation can be too high.

3D Chain Mail technique was introduced by Gibson in 1997 (Varley, 2005). It is very similar to Mass-Spring, however, instead of springs the points are being connected with links. Every point in space can have 4 or 6 neighbours connected depending on whether it is in 2D or 3D space. The main idea behind these links is that they do not act like springs, but instead have minimum and maximum values representing their movement in space. Also, it has proven to be a relatively cheap to compute in comparison with Finite Element Model and making it ideal way to deform even large data sets (Varley, 2005). The following picture illustrates the main idea behind 3D Chain Mail technique:

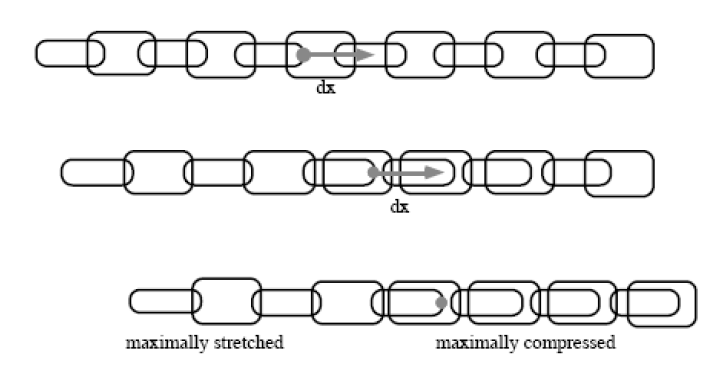


Figure 8 3D Chain Mail Algorithm

If there is no force applied the links remain in their original positions. Once one of the links is moved it is pulling the link that is on the side opposite to the force. The link position on the other side of the one being pulled is being moved in the same direction of the force. The whole movement is similar to the chain and if multiple rows are considered the final simulation would look like a chain mail armour, hence the name “3D Chain Mail”.

## Conclusions

In conclusion, the volumetric rendering remains the key technique in rendering the 3D scalar data. The games industry is starting to use such technique to improve the visual quality of effects such as smoke or dust. It allows to create a more realistic 3D worlds and further enhance the player’s experience. While in games industry the data for volume is usually generated, in medical use such data is being scanned using CT or MRI scanning machines. The use of such scans in combination with volumetric rendering can be used in order to better understand and visualize objects that otherwise are not visible to the naked eye.

Volumetric rendering is not exactly a new technique. For example, the marching cubes algorithm has been presented in 1987 by Lorensen and Cline (Sio et al., 2011). However, the algorithms such as slice rendering is a lot easier to implement and actually is taking advantages of the recent hardware and software improvements. One of the key improvements was the support of 3D textures in graphics API’s that allows to store the whole volume data in a single three dimensional texture. This improvement together with addition of programmable hardware support has led to one of the newest volumetric rendering techniques. Ray casting algorithm has an advantage over marching cubes and slice rendering by eliminating the need of polygonal meshes and also greatly improving the visual quality of the rendered volume.

The collision detection for volumetric rendering can be achieved using similar methods proposed by Iain Cantley (Cantlay, 2007). One of the advantages of checking for collisions in image space over the object space is the computational cost. Due to the volume occupying the space in real time application the object space approach would require additional resources from CPU therefore resulting in lower efficiency. The image space approach can take an advantage of programmable hardware to perform the depth comparisons and calculate the collisions. Similarly to collision detection methods, the two main methods for collision responses have been reviewed. The deformation in model space can result in an uneven visual quality and requires larger amount of operations in order to create a more realistic collision response. The deformation in texture space can result in higher quality deformation, but still suffers from the requirement of additional steps in order to increase the points of available deformation. One of the ways to simplify this method is to perform the textures coordinates offsetting operation in fragment shader. This way the efficiency could be greatly improved due to using the graphics processing unit instead of CPU.

To perform dynamic deformations in texture space a few techniques can be explored. In section 4.8.2 three possible techniques has been reviewed. The Finite Element Model can be computationally expensive and therefore not very suitable if efficiency is one of the aims. The Mass-Spring can be relatively easy to implement and is computationally efficient. However, due to the way the simulation is executed it can look odd if the rendered object is of solid type. The final deformation technique, 3D Chain Mail, remains efficient even if simulation is performed on large datasets. Also, due to the limited movement between the control points the simulation is also very fitting for deformations of solid objects.

# Research Methodology

## Introduction

In the following chapter the project’s methodology will be described. The section 5.2 describes the research strategy and the plan for implementing the application required for the experiment. It will also describe the rationale behind certain design choices that were made based on the literature review found in section 4.9. The section 5.3 will describe the data generation methods and the strategy of how the experiment will be executed.

## Research Strategy

The primary focus of this project will be implementation of the renderer capable of rendering 3D scalar data. The renderer also needs to be scalable and support methods for rendering triangular meshes for scene simulation. In order to successfully complete the application “Agile” methodology will be used. The agile methodology uses iterations to evaluate the completed work and allows any changes necessary to the project. Even if some of the changes might be required the outline of major steps will remain the same. The first step will be to implement a simple renderer capable of basic rendering techniques such as mesh rendering, support for diffuse lighting and model loading. It is an important step where all base code will be completed and additional features will not require structural changes to the base level. The section 5.2.2 will have a more detailed description of the structure.

Once the first step has been implemented and tested the volumetric rendering will be added. The chosen method for volumetric rendering will be using ray casting algorithm. However, before the ray casting will be implemented, the slice rendering will be used as the prototype. One of the reasons for such prototype is the similarity between two volumetric techniques if comparison is made at the base level of each approach. The slice rendering technique is using 3D textures to store data and then a fragment shader to perform a texture sampling. The successful rendering of the volume using such technique will allow progression to the next level.

Ray casting algorithm reviewed in section 4.5.3 has a few major components: 3D texture to hold data, multiple rendering passes and fragment shader to perform ray casting. At this point the part responsible for creation of 3D texture should have been completed if the implementation of slice rendering has been successful. The second component is responsible for multiple rendering passes required to acquire additional data for volumetric rendering. In order to correctly render the volume the start and end points needs to be calculated for the rays that will shot in the fragment shader. One of the ways to acquire such data is to render a geometry used as bounding box for the volume in two rasterization modes: back and front culling. To store this data two additional textures will need to be used to store start and end points for rays. The following example illustrates the result of two rendering passes:

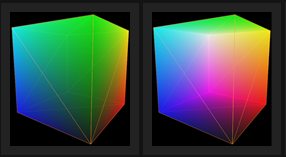


Figure 9 Back and front case culling

The last rendering pass will use both textures acquired and the 3D texture that holds objects scalar data. The following algorithm will be used to perform the ray casting in fragment shader:

Lookup ray start position

Lookup ray end position

Compute ray of sight direction

For each step during length of ray

Get current position at the step

Look up value in texture

Accumulate opacity

Figure 10 Ray Casting algorithm

### Collision Responses

Volume deformation will be the collision response chosen for this project. Such response has been reviewed in section 4.7.2. During this process the texture coordinates for the object are adjusted based on the collision. However, to simplify and potentially improve the efficiency a major update will be made to it to include a fragment shader as part of this process. One of the drawbacks of the deformation in texture space is the requirement of the additional geometry added to the model in order to create a more accurate deformation. This process can be simplified by using another “look up” texture during the execution of fragment shader. The second texture will be created containing the offset positions for volume’s data and the final pixel colour will be adjusted based on the values sampled from it. The following algorithm will be used to create deformation:

Lookup offset position from offset texture

Add position to texture coordinates

Lookup up value in texture using new coordinates

Figure 11 Texture coordinates offset algorithm

### Engine Structure

The engine will use Direct3D functionality to rendered both static meshes and for volumetric data. The main requirement for engine is the scalability factor. One of the ways to accomplish it is to create self-contained layers that are responsible for a specific area. The following picture illustrates such separation:

Figure 12 Engine structure

The system layer will be responsible for windows initialization. It will also initialize the graphics and input layers. During the execution of the application the system layer will be processing the message loop and enabling the graphics layer to do the rendering until the end of application’s execution.

The input layers is responsible for capturing the users input. User input can be used for such controls like changing the rendered model during the execution. Also, to support both slice rendering and ray casting techniques various rasterization states could be changed during the execution to better visualize the differences between two methods.

The graphics layer will be responsible for initializing both the Direct3D and shader management layers. It will also hold the instances of all the models created that can be used during the rendering process. During the main rendering look the following process will be executed:

Shader management layer will have all required shaders for application. The reason for such separation is the possible addition of new shaders if additional functionality is required. The minimal requirement for shader manager will be the implementation of colour, lighting and texture shaders.

Figure 13 Rendering process

For each loaded model

Set up vertex and index buffers

Setup required rasterization state

Set rendering target

Use shader manager to render model using the required shader

## Data Generation Methods

The main data generation method will include the generation of collision responses and storing them in 3D texture as described in section 5.2.1. This 3D texture will be used in the fragment shader to deform the volume during the rendering.

### 3D Chain Mail algorithm

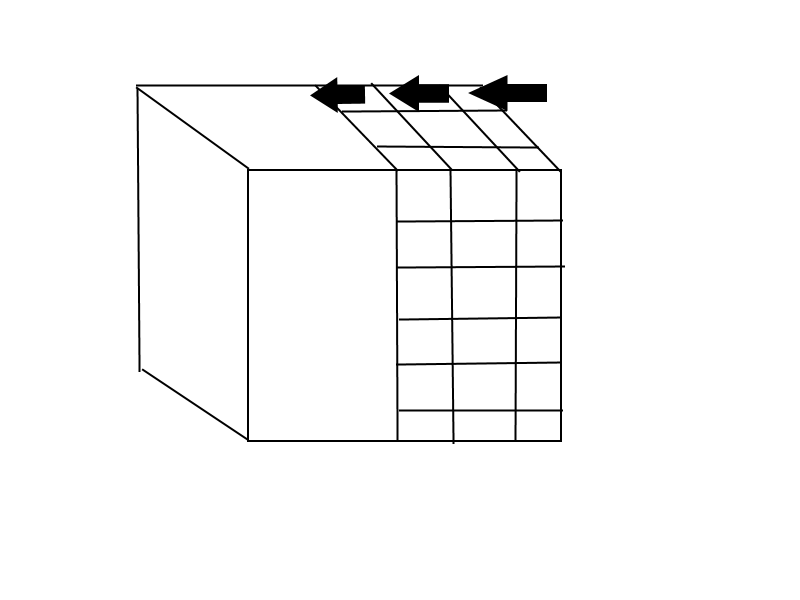
3D Chain Mail algorithm has been described in section 4.8.2. The modified version of this algorithm will be used to generate the collision vectors for 3D texture. The following picture illustrates how offset vectors are applied to each voxel, so the fragment shader could read the location and apply the offset to texture look up:

Figure 14 Deformation vectors

The original 3D Chain Mail algorithm has been used for volumetric data to simulate collision response and then build triangular meshes from translated points in volume data (Gibson, 1996). However, the method has been improved by Florian Schulze, Katja Buhler and Markus Hadwiger by applying the Enhanced Chain Mail algorithm to generate deformation in real time for the volumetric data (Schulze, Buhler, and Hadwiger, 2007). Their method included the ray casting technique for volume rendering and calculating the deformation using the multithreaded approach. During the experiment they made the comparison of four models that had different width, height and depth values. Frames per second have been captured for both rendering with and without deformations.

### Data

The data will be collected by running the renderer described in section 5.2.2. The following list will describe the objects that will be used during the simulation. Each entry will describe the name of the data file, resolution and short description.

1. “bighead.den”. Resolution: 256x256x225. The CT scan of human skull.
2. “fuel.raw”. Resolution: 64x64x64. Simulation of fuel injection into a combustion chamber.
3. “BostonTeapot.raw”. Resolution: 256x256x178. The CT scan of Teapot with a lobster inside it.

Each of the objects will be tested by applying a force vector in the multiple locations and then running the 3D Chain Mail algorithm to measure the frames per second and CPU load values.

### Modified 3D Chain Mail algorithm

The original algorithm involved created a data structure that would hold the position values of each point in 3D volume (Gibson, 1996). Once the force has been applied to a single point the algorithm is used to simulate the deformation by applying the same vector to other points connected to it. Such method requires the deformations to be applied directly to the volume because the data will be later used for rendering pass. The modified chain mail algorithm will use a simple three dimensional array to hold just position for each point and references to all 6 connected points to it. This array will then be used as data for 3D Texture creation that is used in fragment shader to offset texture values of the volume’s data. The key difference between the original and modified 3D Chain Mail algorithm is the size of data array required for simulation. When the collision texture and volumes texture is being compared in fragment shaders the size of both textures do not need to match because of the interpolation of the texture coordinates lookup. This allows to create a much smaller array that can be used as a data for collision texture.

### Limitations

The main limitation of the project will the way the deformations are simulated. To isolate such operation and get more accurate results the deformation will be generated in the exactly the same way for all three objects. This means that the collisions will not be checked during the execution. Instead, a manually picked points will be used in 3D Chain Mail algorithm. One of the reasons for doing this is to eliminate extra computational cost required for collision detection.

## Conclusions

The purpose of this chapter was to describe the design choices made for the application that will be used during the data collection. The sections 5.2.1 and 5.2.2 has described the implementation strategies for the renderer and included a brief overview of the most important application components. The section 5.3 has described the data generation methods. The modified 3D Chain Mail algorithm will be used to generate offset vectors in 3D texture that is used to deform the model in fragment shader. In the next chapter obtained results will be compared to the data generated by the Florian Schulze, Katja Buhler and Markus Hadwiger using the Extended Chain Mail algorithm (Schulze, Buhler, and Hadwiger, 2007).

# Findings and Analysis

## Introduction

In this chapter the analysis will be performed between the results gathered from the Extended Chain Mail algorithm described in section 5.3.1 and the modified 3D Chain Mail algorithm described in section 5.3.3. The results will be analysed by comparing the frames per second required to render volumes.

## Analysis

### Enhanced Chain Mail algorithm results

The following table displays the results gathered by executing Enhanced Chain Mail algorithm on four different models (Schulze, Buhler, and Hadwiger, 2007):

|  |  |  |  |
| --- | --- | --- | --- |
| Data Set | Dimension | Rendering only | Rendering + Chain Mail |
| hydrogen | 64x64x64 | 31.1 fps | 30 fps |
| endoscopy | 512x512x333 | 11.3 fps | 9.5 fps |
| head | 512x512x512 | 15.3 fps | 9.1 fps |
| beetle | 832x832x494 | 4.23 fps | 3.65 fps |

Figure 15 Enhanced Chain Mail tests

Figure 16 Fps difference

In Figure 16 the values are calculated by measuring difference in percentages between just rendering the models and rendering with dynamic deformations. It is clear that the main factor that increases the fps difference is the size of the volume. The hydrogen model has the smallest dimensions therefore the when Enhanced Chain Mail algorithm is executed the number of points moving due to simulation is quite low and results in a minor performance hit.

### Modified Chain Mail algorithm results

The following table will display the result gathered by executing the modified chain mail algorithm:

|  |  |  |  |
| --- | --- | --- | --- |
| Data Set | Dimension | Rendering only | Rendering + Modified Chain Mail |
| bighead | 256x256x225 | 315 fps | 228 fps |
| fuel | 64x64x64 | 320 fps | 229 fps |
| BostonTeapot | 256x256x178 | 313 fps | 229 fps |

Figure 17 Modified Chain Mail test

In Figure 18 the fps differences are displayed gathered from using the modified Chain Mail algorithm. Same as in Figure 16 in section 6.2.1 the values are calculated by measuring the difference in percentages between the rendering with and without deformations. All three models shows a similar Fps readings and the performance hit when deformations are enabled does not result in noticeable difference between objects. One of the reasons for such behaviour is the size of deformations texture. All three models can use the same size texture to sample the offset vectors. Because the dimensions of the texture does not change, the difference between models in fps drop is just a few percentages.

Figure 18 Fps difference

### Accuracy

One of the biggest advantages of performing deformations using the collisions texture is the lack of requirement for it to be the same size as the targeted volume. The collisions texture used during the experiment has a dimension of 16x16x16. It means that only 4096 points can be adjusted by applying the offset vectors. The following pictures illustrates the results by using the different dimensions for collisions texture:

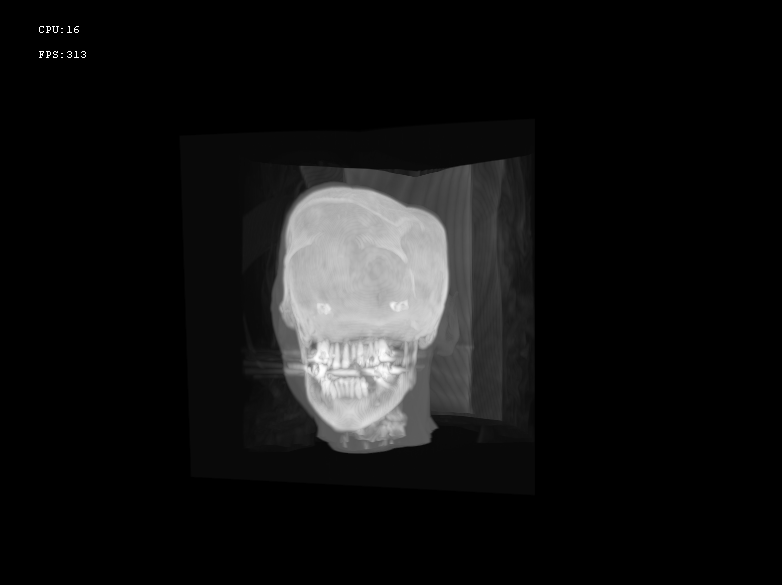
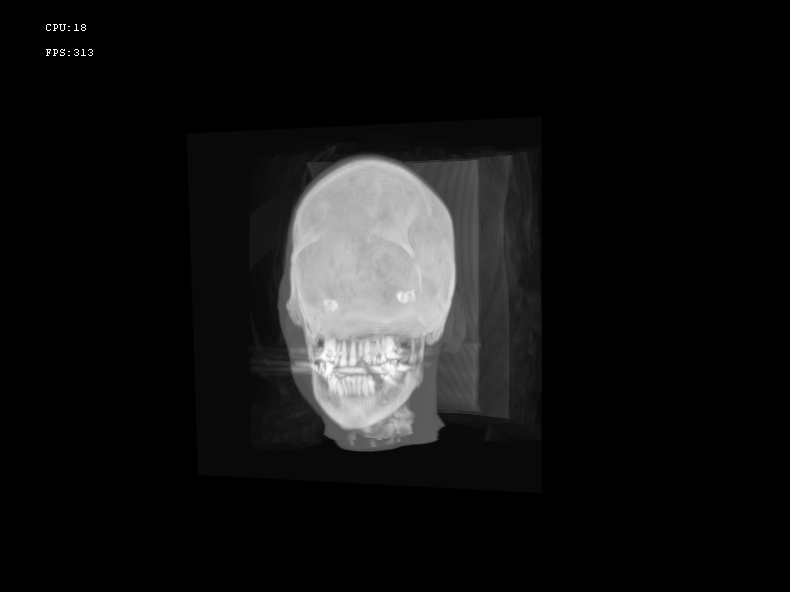


Figure 19 No deformations

Figure 20 Using 16x16x16 texture

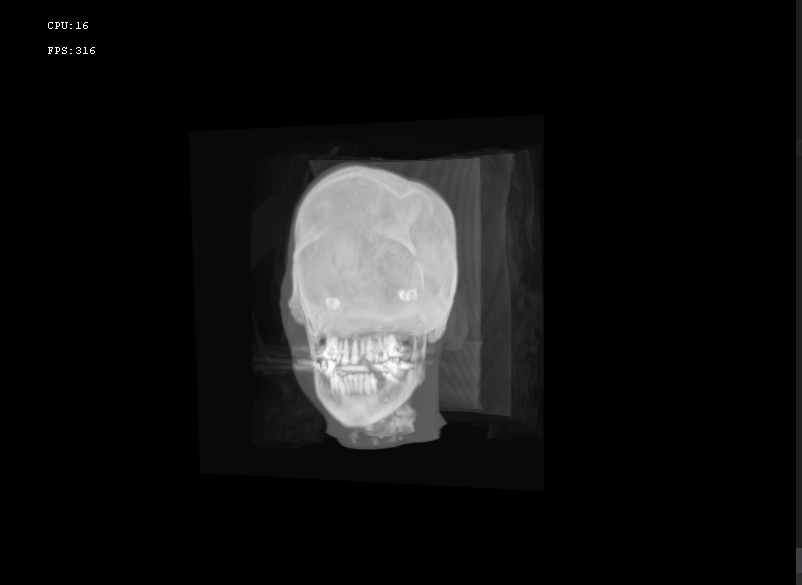
The different dimensions for collision texture resolution impacts the accuracy of the deformation. In figure 17 the image shows the model being rendered using no deformations. In figure 18 the same model has a single vector applied to the point located at the top of the volume. Once the force has been applied, the modified chain mail then traverses through the neighbouring points from the original location and each points receives less and less force due to the falloff variable. In figure 19 the dimensions are twice as big as the ones used in figure 18. Therefore, the amount of points available for deformation has doubled. This means that using the same falloff value the same deformation vector will produce different results. As a possible solution to the problem the offset value can be changed depending of the dimensions of the collisions texture.

Figure 21 Using 32x32x32 texture

# Conclusion and Evaluation

Referring to section 4.3 the primary focus of this project was to investigate and improve real time dynamic deformations with volumetric data. The key things during the deformations are efficiency and accuracy. Performing deformations on large volumetric objects can greatly impact the application performance as outlined in section 7.2.1. In section 7.2.2 the tests has been created to measure the efficiency of modified Chain Mail algorithm that uses 3D texture for deformations. The results show that performing deformations in texture space can eliminate the connection between the performance and size of the volume data.

## Limitations

### Accuracy

One of the limitation of proposed method is the accuracy of the deformation. In section 6.3.3 the modified 3D Chain Mail algorithm has been described and the use of small textures for deformations has been stated as its biggest advantage over the Extended Chain Mail algorithm. However, using the same size texture means that deformations will be calculated ignoring the dimensions of the rendered volume. In section 7.2.3 Figure 19 and Figure 20 shows how same deformation vector has been applied to textures using different dimensions.

### Collision detection

During the test deformations were performed by applying the force vector to one of the points in deformation texture. However, the vector has been hard coded and no collision tests were performed.

## Future work

### Collision Textures

As outlined in section 7.2.3, the same size of collision texture can produce same deformation results that does not depend on the size of the volume that is being deformed. For larger volumes the different size of texture could be used to create a more accurate deformations. Once the dimensions of the textures increases it allows a lot more points to be moved while performing the deformation.

### Interaction with other objects

In section 8.1.2 the lack of collision detection has been stated as one of the limitations of the proposed method. The project can be extended by adding the support of checking for collisions during the simulation. In section 5.6 the image space collision detection has been reviewed. During the volume rendering the bounding volume is rendered for ray casting technique. This bounding volume could be used for collision detection by rendering its geometry to depth buffer and then comparing the depth values with other objects in the scene.

## Closing Statement

The Chain Mail algorithm has been successful improved by adding the collision textures to perform the deformations. In section 7.2.2 the test results shows that modified algorithm has increased the performance of deformations applied to large objects. However, one of the limitations of proposed method is the lack of accuracy during the simulation, as detailed in section 8.1.1.

The increasing demand for better visual quality in computer games means that more realistic looking effects such as smoke or dust clouds will be used. Such effects can only be achieved by using the volumetric rendering technique because it can accurately represent the properties such as smoke density or its particles movement.

However, the dynamic interactions with the rendered volume can be computationally expensive and needs to be highly optimized. The improvements to existing methods of deformations such as Chain Mail algorithm allows the volumetric rendering to be used in more applications that require a better and more realistic visual quality.

# Bibliography

Bartz, D. and Meißner, M. (1999) Voxels versus Polygons: A Comparative Approach for volume graphics.

Cantlay, I. (2007) High-Speed, Off-Screen Particles.

Engel, K., Hadwiger, M., Kniss, J., Rezk-Salama, C. and Weiskopf, D. (2006) Real-time Volume Graphics. United States: A. K. Peters.

Gorkin, T. (2009) Volume Rendering using Graphics hardware.

Jang, H.-Y., Jeong, T. and Han, J. (2007) Image-Space Collision Detection Through Alternate Surface Peeling.

Kindlmann, G. (2002) Transfer functions for Direct volume rendering.

Larsson, R. (2010) Interactive real time smoke rendering.

Lorach, T. (2007) Soft Particles.

Meissner, M. (no date) Available Datasets. Available at: http://www.gris.uni-tuebingen.de/edu/areas/scivis/volren/datasets/datasets.html (Accessed: 10 April 2015).

Meißner, M., Pfister, H., Westermann, R. and Wittenbrink, C. M. (2000) Volume Visualization and Volume Rendering Techniques.

Rezk-Salama, C. (2001) Volume rendering Techniques for general purpose graphics hardware.

Sio, C. C., Ngan, M., Yi, J. and Chen, X. (2011) Volume Rendering with Marching Cube Algorithm.

Stopford, D. (2006) Representing smoke in computer graphics.

Wrenninge, M. and Bin Zafar, N. (2011) Production volume rendering.