Implementation and initial assessment of VR for scientific visualisation

Extending Unreal Engine 4 to visualise scientific data on the HTC Vive

Magnus Elden

Thesis submitted for the degree of Master in Informatics: Programming and Networks 60 credits

Department of Informatics
Faculty of mathematics and natural sciences

UNIVERSITY OF OSLO

Spring 2017
Implementation and initial assessment of VR for scientific visualisation

Extending Unreal Engine 4 to visualise scientific data on the HTC Vive

Magnus Elden
Abstract

Virtual Reality (VR) for scientific visualization has been researched from the 90s, but there has been little research into the fundamental aspects of VR for scientific visualisation. Questions like "Is VR ready for adoption?", "How does VR design differ from design for monocular systems?" are two examples of fundamental questions yet to be addressed. In this paper a scientific visualiser based on the game engine Unreal Engine 4 (UE4) was developed and tested by educators and researchers. A full ray marcher was successfully implemented and a near zero-cost cutting tool was developed. VR is found to have a lot of potential for improving visualisation of data sets with structural "interleaved complexity". VR has also been deemed ready for limited mass adoption. Through field testing visualisations of volumetric and geometric models, three major issues are identified: Current VR hardware lacks adequate input options. Menu and interaction design must be reinvented. Furthermore, 90 FPS is required for comfortable and extended VR use, which makes most current algorithms and data sets incompatible with VR. The conclusion reached through analysis of and feedback regarding the computational cost and design challenges of VR is that VR is best utilised as a tool in already existing monocular visualisation tool kits. By using a monocular system to perform most of the encoding and filtering and then use VR for inspecting the pre-processed model, it is possible to obtain the best of both worlds.
# Contents

1 Introduction

1.1 VR as a scientific tool ........................................... 2
1.2 Simplification of complexity by utilizing the natural power of the brain ............................................. 3
1.3 Currently available input systems are inadequate ............ 3
1.4 Goals ...................................................................... 4
1.5 Achievements ......................................................... 5
1.6 Scope, assumptions and preconditions .......................... 5
1.7 Thesis structure ....................................................... 6

2 Prior research

2.1 Scientific Visualisation ............................................. 9
2.2 Ray tracing ............................................................ 9
2.3 Early interest for VR in the 1990s ................................. 10
2.4 Intermittent phase from 1999 to 2011 .......................... 11
2.5 The second wave of interest in VR from 2012 to 2017 ....... 12

3 Glossary and an explanation of core concepts used in this thesis.

3.1 Glossary .............................................................. 15
3.2 What are geometric rendering and ray traced volumetric rendering? ................................................. 16

4 Demonstrations and tester evaluations

4.1 Demos .................................................................... 21
4.1.1 The Genome demo .............................................. 21
4.1.2 The Rat Brain demo ............................................ 23
4.1.3 The Artery demo .................................................. 25
4.2 Feedback from tests performed in the field .................. 27
4.2.1 Demo specific feedback ....................................... 27
4.2.2 Common feedback ............................................. 28
4.3 Alternative use cases presented by testers and correspondences ......................................................... 30

5 Importing scientific data files into UE4

5.1 Exporting genome data as geometry for the Genome models ....................................................... 32
5.2 Exporting segments of a rat brain atlas as geometry for the Rat Brain model .................................. 33
5.3 Exporting volumetric data of an artery with an aneurysm for the Artery model ................................................. 33
5.4 Exporting flow field as geometric arrow glyphs for the Artery model ................................................................. 35

6 Implementation of a Virtual Reality Scientific Visualiser in UE4 37
6.1 Implementing a ray marcher in UE4 using encoded textures and the fragment shader ........................................ 38
6.2 The cutting effect .................................................................................................................................................. 48
6.3 Model interaction .................................................................................................................................................. 51
6.4 Implementing efficient and comfortable user movement options ........................................................................ 54
6.5 Implementing a post process material to draw multi coloured outlines ............................................................. 57
6.6 Bottlenecks found during this thesis ................................................................................................................. 58

7 Analysis of VR technologies for scientific data 63
7.1 VR can be useful for research, education and industry if used right ....................................................................... 63
7.2 VR helps learning and understanding across familiarity levels ............................................................................. 65
7.3 Observations made during the implementation and testing of this thesis ............................................................. 67
7.4 Input systems .......................................................................................................................................................... 70
7.4.1 Currently available input systems severely limit the design of a VR tool ....................................................... 71
7.4.2 UX design must be reinvented to avoid clutter and obstruction ...................................................................... 73
7.5 Game Engines in scientific visualisation ........................................................................................................... 73
7.5.1 UE4 .................................................................................................................................................................. 74
7.6 Non software related challenges ......................................................................................................................... 76
7.7 Research, education and product development have different HMD requirements ........................................... 77

8 Conclusion 79
8.1 Several challenges solved for monocular systems remain to be solved for VR ..................................................... 79
8.1.1 VR needs a powerful standard input device .................................................................................................. 79
8.1.2 The limit of light speed changed all of physics, VR’s required 90FPS changes algorithm design .................... 80
8.2 Ray tracing volumes in VR is possible if simplified .............................................................................................. 80
8.3 Current VR is best utilised as an extension of monocular systems ........................................................................ 81
8.4 Unlike monocular visualisers, VR’s usefulness grows as model complexity grows .................................... 81
8.5 Concluding statements .......................................................................................................................................... 81
8.6 Future work ........................................................................................................................................................... 82
List of Figures

1.1 Left: The artery model. Middle: The Genome models. Right: The Rat Brain model ................................................. 4

3.1 1. The different parts that makes up the view space. 2. The view frustum cutting away the sphere as it is outside the frustum. 3. The world space with a shadow representing the occlusion of the second box. 4. The scene after it has been transformed to view space using a projection transform. ................................................................. 19

4.1 The genome data visualised inside the VR visualiser. ............ 22
4.2 The complete geometric representation of some of the larger rat brain segments visualised inside the VR visualiser. .......... 24
4.3 The geometric and volumetric artery model visualised inside the VR visualiser. The cut separates the volumetric and the geometric model. ................................................................. 26

5.1 These are the manual steps taken to convert the scientific data to FBX mesh files and texture files. Green represents geometric data and light pink represents volumetric data. The nodes on the left represent the raw data and the rest represent programs used during the conversion. .................. 31
5.2 Two slices photographed with equal distance between the slices and the cameras ......................................................... 34
5.3 A 16x16 slice set of a single time step for a single volumetric artery model. ................................................................. 35

6.1 The design of the ray marcher implemented in this thesis. It is implemented in the fragment shader and consists of two main steps; setting up the environment for the generic ray marcher and the ray marcher itself. .............................. 38
6.2 1. The ray hitting the $1 \times 1 \times 1$ volume. 2. Shows the samples taken within the volume. ......................................................... 39
6.3 1. The ray vector scalar for the entry point we call T0 and the scalar for the exit point we call T1. 2. The ray unit vector and the vectors $\vec{A}$ and $\vec{B}$ broken down into its components. .......... 40
6.4 An example grid of size $8 \times 8$. Each slot corresponds to a slice in the volume. ................................................................. 42
6.5 A sample along the ray defined by a blend between a sample from both encapsulating slices. ........................................ 44
6.6 A geometric box where each side’s normal is facing inwards, making the faces transparent if viewed from outside.

6.7 1. With depth testing. 2. Without depth testing. 3. Object behind volume without early ray termination. 4. Geometry and volume intersecting with early ray termination.

6.8 The left image shows a geometric model being cut while the right image shows a volumetric and geometric hybrid being cut.

6.9 Left: World Space. Right: Volume space.

6.10 Each object has its own local coordinate system while they share a world coordinate system. The blue area is the view space which determines what the camera can see.

6.11 The structure of the classes handling interaction. Gray nodes represent custom code while white nodes represent functionality already present in UE4. The orange arrows indicate the direction of the information flow.

6.12 This is a screenshot of the rat demo with the thalamus mesh disabled by using the menu. The model is also cut by the cutting plane.

6.13 The teleportation arc at a steep (left) and shallow (right) angle. The arc can be tweaked to balance between distance and ease of control.

6.14 The plane determined by the orientation of the camera. The blue arrow was the rotational axis of the thumb pad, represented as the blue circle.

6.15 Left: The outline as viewed in the visualiser. Centre: The custom depth buffer. Right: The stencil buffer with stencil values.

7.1 The top two images show the same car model one second apart. The bottom three images show three different configurations of the same interior. Images taken from the Epic Games stream (McLaren and Epic Games, 2017)

7.2 A breakdown of a rendered frame of the Artery Demo in UE4. Total render time: 8.39ms Time budget remainder: 1.61ms
I would like to thank my supervisors Simon Wolfgang Funke and Carsten Griwodz for their guidance and support during this thesis. I underestimated the challenge of writing a good thesis and it is thanks to them that this thesis was written. I want to thank the researchers at the Institute of Basic Medical Sciences, University of Oslo, who acted as testers and provided the data sets used to evaluate the VR visualiser. I want to thank the Unreal Engine community for helping me with the small issues that came up and especially Ryan Brucks, the Principal Technical Artist at Epic Games, for pointing me in the right direction when implementing the ray marcher. Finally, I would like to thank my girlfriend Tomori Kana for her constant encouragement, love and support. Without their help this thesis would not have been possible.
Chapter 1

Introduction

Research and education are central to modern society, and visualising data is a ubiquitous tool that is used in almost any part of life in some form. Visualising data is not a modern concept. Graphic portrayal of quantitative information has a long history stretching from at least the Babylonian map of the world; usually dated to the 5th century BC. Scientific drawings as we know it came about during the Age of enlightenment and researchers have employed all manners of techniques and tools to create ever more detailed and powerful visualisations. Even though scientific visualisations existed before the age of enlightenment, they were mostly hand drawn. Leonardo da Vinci’s Vitruvian Man is a famous example of this. The human element was removed when machines were used to capture data. The use of mechanical machines\textsuperscript{1} became commonplace.

With every new invention new possibilities are born and require testing, research and new ways of thinking to be utilised properly. Just like the electrical computer has revolutionised the way humans process and visualise data, Virtual Reality (VR) also opens new possibilities to improve visualisations. If the new technology is to be used to its fullest potential the developers using this new technology must first understand the advantages and restrictions of the new technology. VR is such a radical departure from standard 2D screens that many of the designs and algorithms that work on 2D screens are not applicable to a VR system. Further, VR has not yet found what it excels at. As with any tool, VR is unmatched in some areas while not viable in others.

Ever larger and more complex data sets are visualised and new visualisation techniques are required to meet the demand for efficient analysis of such data sets. VR provides an option to visualise spatially complex data that simplifies interactions and understanding. The largest data sets require constant RAM and storage access. Due to the high frame rate requirements of VR it might not be possible to transfer data to the graphics memory while rendering a frame. This would restrict the sizes able to be visualised to the limits of the graphics memory. Given that

\textsuperscript{1}Some used electrical signals; for example the seismograph built by Luigi Palmieri in 1856 measured the vertical vibration intensity using electricity. Oldroyd, 1996, p. 228 Lee et al., 2002, p. 228
the data can be accessed sufficiently fast to render at the high frame rates required. VR provides unparalleled visualisations for spatially complex data that make learning and understanding more efficient.

1.1 VR as a scientific tool

Current visualisation methods are all mostly displayed on 2D screens and is used to visualise many types of data over many dimensions. For many types of research the models are relatively simple, but the underlying data can be complex. A lot of visualisation techniques focus on extracting the data that is interesting and presenting it in a compact and simplified way. For many use cases the resulting models are mostly convex with few overlapping or interlocking parts. Some fields of study, for example fluid dynamics, often result in complex structures interleaved into each other and various techniques are used to encode depth. Techniques like shading and shadow casting is also used to enhance the sense of depth. For many models this can be enough, but just like having two eyes helps with judging relative distances and understanding complex 3D shapes, VR provides alternative approaches to understand depth that makes the above process faster.

Another benefit VR offers that can improve understanding is the removal of algorithms designed to encode depth. Expensive algorithms that handles encoding distance and removes obfuscating data can be replaced by giving the user a natural way to simply move into the volume. By moving his head the user can go past the obfuscating part of the model and see the data that is of interest to the user.

Parallax movement and movement aligned with the line of sight will be much easier to discern. Since you now have binocular vision you can also use summation to avoid confusion when faced with ambiguous images. One eye might be fooled to think one thing, but the other eye has a sufficient angle to filter out some of the erroneous alternatives the first eye sees. Unlike a monocular system like classic 2D screens where the user must slowly piece together a mental model by moving back and forth, a binocular system enables use of the the acute sense of depth and motion humans have developed thanks to their binocular sight. The human brain is very good at stitching together an accurate mental model from the binocular vision and is so important that people who lose sight from one eye struggle with many basic tasks like grabbing and pressing for a while.

Having said all that, the potentially single most important benefit VR can provide users is the reversal of the control scheme. Humans are used to moving their heads to look around and change perspective. In most 3D rendering software, scientific or not, the camera is stationary and you edit the location, scale and orientation of the model. This outside in perspective requires learning and is not as intuitive and natural as the inside-out controls that VR offers. Viewing and moving through the model is as natural as looking at an object in real life.
1.2 Simplification of complexity by utilizing the natural power of the brain

As mentioned above, VR utilizes the mental capacity of humans to process 3D structures to enhance the user’s spatial understanding. Traditional 2D screens can already be used to render 3D models and it is often still easy to get a sense of the structure of convex models. For simpler models VR has very little to offer, but as the complexity grows VR grows more and more beneficial. Part of the reason for why this is so much more effective is the precision of the human neck. You can go up close from whatever angle you want at any distance you want without having to perform several separate actions. Your brain is naturally guiding your neck because it knows exactly what to do to get the view it desires.

The other major ability that the brain has is analysing the binocular stream of images to create a mental representation of the 3D structures that are placed in front of them. This ability pairs naturally with the ability to move the head into the appropriate position to get the data one needs. The continuous movement provided by the neck will give the brain a continuous stream of information without diverting attention and focus to moving the camera.

1.3 Currently available input systems are inadequate

In the current state of the VR technology available we are limited to quite crude forms of interacting with the virtual world around us. We do have some buttons and the movement of the controllers themselves to work with, but the available options are very limited compared to a mouse and a keyboard. This forces a more streamlined design for input. If the user has to learn complex button combinations to perform basic actions we are essentially back to square one. Until systems that allow the user to track their own hands and use these hands as input devices we will be limited in the granularity of the input options offered by controllers. After we create a system that can offer reliable tracking of the hands of the user we still have the issue of tactile feedback. All of these input related challenges combined makes it currently impossible to create systems of high enough fidelity to replace 2D screens. A good temporary middle ground could be using voice to select input modes that allow the limited input options we have available to perform a wider set of interactions.

Interactivity is very important when analysing data. Several methods of visualization and several encodings might show the researcher a unique part of the whole. Having the ability to edit properties in realtime with visual feedback is indispensable for expedient research as quick iteration cycles means less time overall spent. Standard screens can be used with a keyboard and mouse since you are sitting down and you can always look at your desk to find where to place your hands and fingers. The mouse pointer has a width of a single pixel so very compact menus are still usable because the mouse is so accurate. Further, small input boxes work well
because the keyboard can manually input any value it wants. In VR typing is a major issue because it is slow and error prone. It can still be done, but the benefit of VR will be reduced.

Another issue with classical approaches to interaction through menus is the fact that the render window is fixed to a certain location and a certain size on the screen. You move the model, not the window so you will always have the menu available in front of you without it being obfuscated by or obfuscating the model you are trying to view. Menus in VR have to be placed in the world which quickly leads to a cluttered scene. VR menu based interaction should be restricted to only the bare minimum without some method of showing a menu while it is being used and then hide it again.

1.4 Goals

This thesis aims to identify and provide a set of universal guidelines for developing good VR tools for any kind of scientific visualisation. To achieve this, three test cases, called demos in this thesis, were developed in collaboration with medical researchers using the HTC Vive and Unreal Engine 4(UE4). The three demos were made to try to highlight some of the key aspects of what a VR tool is or is not good at in regards to scientific visualisation. What VR can offer education, research and product development, and what identifies a good task for VR, are analysed using the three demos; The artery demo, The rat brain demo and The genome demo. The artery demo displays a volumetric representation of an artery with an aneurysm and was made to evaluate VRs applicability with volumetric models and structurally simpler models. The rat brain demo visualises the neural sectors of a rat brain and was developed to analyse VRs applicability in an educational setting and to identify what VR can offer structurally semi-complex models. Finally, the genome demo visualises two geometric representations of a genome and its chromosomes and was developed to identify the applicability of VR in cutting edge research and with very complex and dense three dimensional structures. Figure 1.1 shows the three data sets visualised. More detailed description of the demos will be given in chapter 4.

![Figure 1.1: Left: The artery model. Middle: The Genome models. Right: The Rat Brain model](image)

With the new interest in VR it is important that a new analysis of VR is performed. Is VR ready for scientific visualisation? How does VR design
differ from standard monocular design? How is VR best utilized and what makes a task suited for VR? It is important that basic questions like the ones above are answered. This thesis aims to answer such questions to pave the way for more in-depth and specialized research to be performed.

1.5 Achievements

An interactive VR visualiser with a volumetric ray tracer and a near zero-cost cutting tool running at 90 FPS was developed. It supports complex renders of geometric and volumetric scenes with a two-step outline render added in post process. A complete set of interaction options were developed using the HTC Vive controllers and in-tool menus. This thesis also provides a set of guidelines for designing and implementing effective and user friendly VR tools for any kind of scientific visualisation. Based on field testing and feedback from researcher who used the VR visualiser, fundamental requirements were identified and a clear correlation between model complexity and VR superiority when compared to monocular alternatives was discovered. New and unique avenues for visualisation, for example effectively visualising shapes while hidden by opaque geometry or dense volumes, was discovered and found to offer completely new methods of inspecting multiple parts of the same model. Finally, we have implemented a volumetric ray marcher that can be placed inside a larger scene. Together with earlier work it successfully shows that current VR and current desktop computers have reached a maturity level sufficient for widespread scientific use.

1.6 Scope, assumptions and preconditions

This thesis will only consider three dimensional visualisations. It is assumed that two dimensional or one dimensional visualisations will not benefit from VR. For higher dimensional data it can either be time represented as a changing three dimensional model or higher dimensions projected down to only three dimensions and as such will be included in the consideration.

When talking about Virtual Reality or VR in this thesis it only refers to completely immersive 3D display techniques based on head mounted displays (HMD). The Oculus Rift and the HTC Vive are examples of such HMDs. 3D glasses and 3D screens are not considered. Further, augmented reality solutions where the virtual scene is overlayed the real world are also not considered.

To explore the full extent of what can be done with modern VR, the VR system needed to support all six degrees of freedom, spatially tracked controllers, high resolution, low refresh times and reliable tracking. At the start of the implementation, the HTC Vive and the Oculus Rift were the only commercially available Head Mounted Displays (HMD) that filled the
requirements. Due to delayed sale of the Oculus Rift’s controllers and due to the reported higher reliability of the HTC Vive’s tracking it was decided to use the HTC Vive. UE4 completely supports the HTC Vive and SteamVR, which is the software platform between the hardware of the HMD and the application running on it. Further, according to steam survey (Steam Hardware Survey 2016) the HTC Vive had 66% market share when this thesis was written.

It was also assumed that writing a complete visualiser toolkit using using only for example OpenGL and C++ would not result in any meaningful results due to time constraints. As such a quick assessment of alternatives were made and the two viable alternatives decided upon were (1) using a game engine with support for the HTC Vive or (2) using an already existing scientific visualiser tool kit like Paraview. As the game engine candidate, Unreal Engine 4 (UE4) seemed to be a good choice. The thesis required a rendering engine and a physics engine, support and incorporation of the HTC Vive and a series of functions to manipulate the data and implement interaction options. Paraview, one of our best contenders had a vast array of functions and rendering techniques that enables visualisation of many types of scientific data, but it did not have VR support or a simple environment for implementing interactions. Since the goals of this thesis was to explore VR as a tool for scientific visualisation it was decided that being able to quickly implement and test a wide variety of algorithms and interactions were more important than pushing the limits of what VR could offer. UE4 is open source and allows for implementing and prototyping mechanics and interactions as well as rendering techniques quickly. UE4 also made creating flexible UI much simpler. That is why UE4 was selected as the framework within which the project was to be developed.

1.7 Thesis structure

To provide context to this thesis, chapter 2 presents a concise and chronological look at scientific visualisation in general, ray tracing as a visualisation technique and the attempts at using VR for scientific visualisation.

To enable readers unfamiliar with rendering related terms and concepts to also follow the implementation details and discussion chapters, chapter 3 provides a glossary of important terms for understanding this thesis. Some terms are also defined to distinguish between similar, but distinct concepts used for more nuance. These terms are highly specific and can differ from the dictionary definitions. A short introduction to core concepts like how volumetric and geometric rendering works and what a geometric mesh actually is is also provided.

To give the test cases context and to make the discussions easier to follow, chapter 4 presents the data and research the demonstrative test cases were based on. These demonstrative test cases were used to
evaluate the VR visualiser when compared to already existing solutions. The chapter also summarises who the testers were, how the tests were conducted and the feedback provided by the testers.

Since the VR visualiser presented in this thesis did not have a reader for the file types used, the data sets were manually processed to create static demonstrative models. Chapter 5 describes the manual process of converting the scientific data files provided by the testers to a format compatible with UE4.

The body of the thesis starts with chapter 6. This chapter describes the implementation details for each module UE4 was extended with. Challenges overcome and alternative approaches that were considered, but was discarded, are also discussed. Bottlenecks found during the implementation and VR specific optimisation is also presented.

Chapter 7 discusses VR for scientific visualisation based on previous work, lessons learned through the implementation of the VR visualiser and feedback from tests in the field. An analysis of the pros and cons of VR as a visualisation tool for research, education and industry is provided. Problems with current VR technology is presented and discussed. Game engines are assessed as frameworks for creating scientific visualisation tools.

Chapter 8 summarises the main conclusions of this thesis and provides a roadmap for future research.
Chapter 2

Prior research

To be able to analyse and understand VR as a tool for scientific visualisation it is important to understand standard visualisation. Only then can key differences between the standard 2D screens and VR be identified.

2.1 Scientific Visualisation

According to Friendly (2007) data visualisation has deep roots stretching from the earliest maps. Modern computer aided visualisation software started out as highly specialised, command driven software. The closer we get to the present day the software became real-time and interactive and more general. This is also reflected in the research of Upson et al. (1989). Volumetric data can be rendered by using several methods. Runge-Kutta integration is commonly used for 2D and 3D visualisation of vector fields by seeding path lines, streak lines or stream lines (Max, Crawfis and Grant, 1994). Segmentation is also important for efficient volumetric rendering (Sherbondy, Houston and Napel, 2003). A generic description of what Scientific Visualisation would be and how to create such software was given by Haber and McNabb (1990). Scientific visualisation has been adopted by and applied to several different fields and disciplines.

2.2 Ray tracing

A simple method of rendering a volume using ray tracing is Maximum Intensity Sampling. Like an X-Ray scan only the highest density sample is stored (Parker et al., 2005; Schreiner, Paschal and Galloway, 1996). A major disadvantage of this type of rendering is that depth and rotation information is lost.

Scattering, also called emission, and absorption models are used to mimic real life rays, for example light rays. It is best described by Blasi, Le Saec and Schlick (1993, p. C-203): “Absorption corresponds to the transformation of energy from the visible spectrum into warmth. Scattering corresponds to a distribution of energy in the whole space when a ray of light intercepts the surface of a particle”. For 2D screens such rendering
techniques is quite common for volumetric data (Blasi, Le Saec and Schlick, 1993; Sabella, 1988; Govaerts and Verstraete, 1998; Jensen and Christensen, 1998; Drebin, Carpenter and Hanrahan, 1988). Ray tracing can also be used to render geometric scenes (Govaerts and Verstraete, 1998; Jensen and Christensen, 1998).

Most optimisation of ray tracers come from structuring data sent to the GPU for fast access and from removing unwanted or empty sections of space. Further optimisation can be achieved by skipping samples after the cumulative density reaches a threshold or by skipping uninteresting sections of space (Wrenninge, 2016; Penney, 2016).

2.3 Early interest for VR in the 1990s

Early interest in using VR for scientific visualisation arose in the 1990s with Mercurio and Erickson (1990) and Cruz-Neira, Sandin and DeFanti (1993) and their research into VR’s viability as a tool for scientific visualisation. The first two attempts at VR visualisation defines the two types of VR we have seen this far; The stereoscopic display, called an HMD, and the encompassing box upon which the scene is projected, called a CAVE system. Mercurio and Erickson (1990) developed an HMD that had six degrees of freedom and a resolution of 360 × 240. They used a glove and gestures to translate the user and to ”grab” the model they were testing on. They evaluated VR to have immense potential due to its natural movement, interactivity and sense of depth, but it was not ready for mass adoption since the technology was not powerful and compact enough yet. Cruz-Neira, Sandin and DeFanti (1993) developed a system where the user was tracked and the scene would then be projected onto the walls and floor of a blank cube within which the user was located. This design effectively removes issues with weight, collision with the unseen environment and cable tangling complicating the design and use of a HMD system like the one developed by Mercurio and Erickson (1990).

Wexelblat (1993) collected and edited a collection of essays speculating over the possibilities of VR. Information can be simplified more than ever, and travel and exploration of even other planets are possible. Education can be revolutionised and new methods of visualisation will enable yet to be imagined possibilities. Even though this book is mostly a collection of speculations for what VR can become it does provide a single extremely important point that has since been mostly ignored; Cooperation. He claims that ”human activity is inherently cooperative”(Wexelblat, 1993, p. 24) and that VR is suited for providing a cooperate experience.

A few years later Bryson (1996) takes a more general look at VR and lists a few core requirements for VR applications; fast response time, a fast frame rate(10FPS or more) and a sufficient fidelity of visualisation to be useful. He also presents some very accurate observations; a natural interface is required, some consistency in the world is required, the visualisation must be in real time and that VR provides a cheap and fast way to inspect data with a superior depth sense. He also discusses what VR can be used for and
how to best design for VR. He predicts that haptic feedback and sound can be used to enhance the experience and specifies a series of guidelines for every facet of a VR visualiser. He has a tendency to become too verbose and making too many rules. Many of his guidelines are either almost self-evident or not quite accurate. Still, he presents many useful observations and conclusions which this thesis builds upon.

The early period of VR scientific visualisation highlighted the many challenges still hindering VR from being viable at the time. The possibilities VR offered was concluded to be immense, but the technology required to make VR useful was still not available.

2.4 Intermittent phase from 1999 to 2011

Interest for VR died down with the turn of the millennia, and HMDs were left for a while. Fröhlich et al. (1999) tried a new approach by having a hologram display the geological data while the user controlled the model using their hands through a physical cube as proxy. By providing the user with a proxy cube Fröhlich et al. (1999) were able to make interaction with the virtual cube natural and instantly familiar in spite of the technological limitations of the time. They conclude that interacting with the model using your hands is the most natural option and can offer great benefits if used.

Dam et al. (2000), Nguyen et al. (2001), Boyles and Fang (2003) and Ohno and Kageyama (2007) were some of the few projects that explored using VR for widely different fields. Dam et al. (2000) analysed the challenges of designing input devices that are intuitive and effective in a VR environment and the limitations of the available technology that must be resolved before VR can become widely and comfortably used. Nguyen et al. (2001) developed a system for remotely controlling complex robots after a stereoscopic camera had garnered success during the Mars Pathfinder mission. They concluded that VR offered an improved situational awareness for the operator and that the increased spatial awareness allowed for much better analysis of the rover’s surroundings. Ohno and Kageyama (2007) was able to implement a real time high fidelity volumetric renderer for the CAVE system and concluded that volumetric data can benefit from the increased depth perception of VR, something that Boyles and Fang (2003) had predicted earlier.

During this period VR was starting to mature and the consensus was that the only restrictions that were left before VR could be widely adopted and used outside of research were hardware related. Solutions that was powerful enough and approachable enough to be useful were developed and tested. CAVE systems were favoured as such solutions did not have the unsolved issues related to not being able to see your environment and the cumbersome nature of HMDs. Interface design was also losing traction during this period as well and the problem of creating natural and intuitive input devices for VR was left an open challenge.
2.5 The second wave of interest in VR from 2012 to 2017

With the successful kickstarter by Oculus for their VR headset in 2012\footnote{https://www.kickstarter.com/projects/1523379957/oculus-rift-step-into-the-game/} the interest in VR was rekindled. Even though the technology was marketed for entertainment, several companies and research teams started exploring the capabilities of the new system.

Hybrid Reality (HR) refers to a system where different kinds of media and the real world is blended. For example, Reda et al. (2013) developed a system that allowed several people to view stereoscopic 3D images and text and each other at the same time by utilising stereoscopic LCD panels to create a 320 degree panoramic view. The authors do not describe how the stereoscopy was achieved so it can only be assumed that shutter glasses were used. These hybrid systems are the continuation of the CAVE system and has several of the same downsides. Price, space and user interactivity becomes major hindrances for mass adoption. Multi-user systems like the one presented by Reda et al. (2013) also does not allow for user tracking. Song et al. (2014) used both CAVE like systems and HMDs for rehabilitation research. Together they indicate that research into and using VR is becoming more and more saturated and widespread. Reda et al. (2013) showed that high quality volumetric and geometric renders are possible using current technology in a CAVE like system.

HMD based research has also increased in the last few years. According to the survey performed by Ott, Freina et al. (2015) there were 93 papers on "Immersive Virtual Reality Education" and 18 papers on "Head Mounted Display Education" written and indexed for the years 2013 and 2014. This clearly shows a resurgence of interest in HMD based VR even in scholarly spheres. It has also been used for pain management testing on patients (Dascal et al., 2017) or treatment of Schizophrenic people (Moritz et al., 2014).

VR has also been getting more interest from education as well, as it is the natural next step of older research into education (Roussou, 2004; Trindade, Fiolhais and Almeida, 2002; Blasco-Arcas et al., 2013). Huang, Liaw and Lai (2016) concluded that VR helps with student understanding and enthusiasm for learning. VR can also be effectively combined with gamification to create interesting and motivational classes even for younger students (Xu and Ke, 2016). VR has also been successfully used to let university students experience Parkinson’s disease psychosis (Goldman et al., 2016).

Corporate research into and development of commercial VR HMDs also sparked non-scholarly entities to invest into researching optimisation and design of VR software and interfaces (Vlachos, 2017; Bunn, 2017; McLaren and Epic Games, 2017). Game creators and other interactive media creators have been exploring how to move the user without causing discomfort and how to create powerful and natural interactions.
Penney (2016) implemented a restricted volumetric renderer that added volumetric clouds to their animated VR movie. They achieved this by using simple geometric meshes to define the starting location of the ray and only collect a few samples. The users would be relatively stuck inside a certain area limited by their own physical room. This enabled them to use simple blending techniques for their sampling and to use baked shadows. Many of the assumptions they could safely make does not translate to scientific visualisation where the user would be able to go anywhere and inspect the model from any angle. Still, it demonstrated that making a hyper efficient VR volumetric ray tracer implemented in UE4 was possible.

Pohl et al. (2015) and Weier et al. (2016) used eye tracking and foveated imaging to increase the FPS of their geometric ray tracers significantly with minimal perceived visual quality loss. This technique is similar to the technique described by Vlachos (2017) where unseen parts of the render is removed and where the external ring around the focal point is is rendered with a reduced number of pixels.

It finally seems like high quality volumetric and geometric visualisations are possible with currently available computers while also ensuring low latency and 90 FPS. Every volumetric ray tracer implementation were severely restricted, leaving open the question of whether it is possible to have a complete volumetric ray tracer run at 90 FPS or more in VR. A pilot project developed alongside this thesis indicates that a high quality interactive volumetric render is possible. More information can be found in appendix 8.7.

A major issue prior research collectively have is that none have analysed the fundamental aspects of VR as a tool for scientific data. VR is a new paradigm and offer unique challenges. Prior research rediscover the same benefits and limitations of VR during their research. VR requires radically different designs to be effective and universal "best practices" would expedite future research. Fundamental questions about how to create VR tools, how they differ from standard tools and what the restrictions and possibilities of VR are are left mostly unanswered.
Chapter 3

Glossary and an explanation of core concepts used in this thesis.

This thesis assumes an understanding of programming in general. Code provided are either pseudo-code or High Level Shading Language (HLSL) or Python. This chapter defines the terms used in this thesis and explains more specialised concepts needed to understand the thesis. In section 3.1 terms as they will be used in this thesis are defined. In section 3.2 core concepts will be explained in some detail.

3.1 Glossary

To avoid confusion between terms that have similar meanings in common parlance and words that has a special meaning in the technologies used a list of definitions is prepared. These definitions might differ from the definitions found in a dictionary since they are domain specific terms used in graphics programming, rendering tools and this thesis.

**World Space** - The world space is the mathematical space where each object share a global origin. A detailed explanation is provided in 3.2.1.

**Scene** - A scene is the world the end user sees. It is almost the same as world space, but the key difference is that the scene is a higher level term thought of as the stage for your content. Implementation details and the mathematical underpinnings are not considered.

**Fragment** - A fragment is a pixel candidate in geometric rendering. If two or more geometric polygons are overlapping at a pixel, a fragment is generated for each overlapping mesh.

**Shader** - A combination of two programs executed on the GPU. The vertex shader is called once per vertex, that is a point in space, and handle most basic calculations and mesh morphing. The fragment shader is run once for every fragment and is usually responsible for shading, shadows and other visual effects. The fragment shader tends to be the more expensive program of the two.
**Shading** - Shading is the darkening of pixels to give the appearance of depth. As the angle between the normal of the surface and the vector to the light source increases the less light that fragment receives and is darkened. Shading does not consider obstruction of light.

**Shadow casting** - Shadow casting, also referred to as shadows in this thesis, is darkening of meshes due to light being blocked by other geometry. It can be blocked by the meshes own geometry or by other geometry in the scene. Volumes can also cast shadows.

**User** - A generic user. Used when making general statements or when discussing hypotheses.

**Tester** - A tester is a user that actually tried the VR visualiser and provided feedback. Unlike a "user", a "tester" refers to a finite and defined group of people.

**Mesh** - A mesh, also called a static mesh in this thesis, is a geometric data type. Unlike volumetric data, geometric data is represented by a set of two dimensional triangles.

**Model** - A model is a representation of data. It can be any kind of data and it can have any kind of encoding or processing performed on it. A model can contain several data types or other models. For example, the rat brain model has several distinct meshes and the artery demo has a volumetric data space and a geometric mesh.

**Demo** - A demo is a scene with one or more models and the interactions available to the user. Each demo presented in this thesis is self contained.

**Geometric rendering** - Geometric rendering is rendering of triangles, lines or points. In this thesis only triangle based data, known as meshes, are used.

**Volumetric rendering** - Volumetric rendering is rendering of volumetric data where a bound and finite space of data is rendered. Unlike geometric data, volumetric data covers a volume of space.

**Ray tracing/marching** - Ray tracing and ray marching are techniques used to simulate a ray of light travelling through the scene before reaching the camera. Often the ray is emitted from the camera instead of from the light sources in the scene. This thesis uses camera emitted rays.

**Interleaved Complexity** - A term coined for this thesis. Interleaved complexity refers to structural complexity where the structures overlap and entwine.

### 3.2 What are geometric rendering and ray traced volumetric rendering?

This section provides a quick explanation of the core concepts of and differences between rendering techniques used in this thesis. Some of the details are UE4 specific and explained as they are used in this thesis. It is highly recommended that they are understood before attempting to understand the implementation and analysis in chapter 5, 6 and 7. Akenine-Möller, Haines and Hoffman (2008) presents a detailed and
complete description of the graphics pipeline and common algorithms used to render both geometric data and volumetric data.

Ray traced Volumetric rendering

Ray traced Volumetric rendering is a rendering technique that aims to simulate light moving through the volume and hitting the camera. It starts out by defining one or more rays per pixel where each ray originates in the camera. The algorithms samples a discrete series of points along each ray and blends these samples according to some set of filters, encoding and blending rules. The rays can be absorbed and scattered, refracted or reflected or stopped by opaque objects. This is important to remember to understand the difference between volumetric and geometric rendering. The different methods will be discussed in more detail in chapter 6 when describing the implementation of the VR visualiser.

Volumetric models have three stages to their render. The first is the preparation of the scene. All rotation, scaling, translation and variable editing happen during this stage. All interaction, physics calculations and data swapping also happens at this stage because it relies on the CPU. Uniquely to volumetric data any segmentation or removal of unwanted data is performed at this stage. The next stage is the sending of the data from the RAM to the graphics card’s memory. After the data has been prepared it is packed together into a predefined structure and is sent to the GPU in batches. The final stage is the shader stage where the actual ray tracing is performed. The GPU simulates a series of rays and samples the data. When the ray completes the pixel it represents is coloured based on the sampled data.

Geometric rendering

Geometric rendering is a method of rendering that defines a series of bounded 2D planes in 3D space. Programs might represent the geometry using any kind of 2D shape, but before being processed by the GPU the more complex bounded planes are translated and broken down into 2D triangles. These triangles are drawn by providing the GPU with sets of three points in 3D space. These triangles usually share edges and vertices with other triangles in order to give the appearance of a continuous 3D shape.

In geometric rendering there are also three stages to rendering a scene. The first two stages are very similar to volumetric rendering. Only the kind of data being processed, how it is being processed and how it is sent to the GPUs differs. The final stage is the shader stage where the GPU processes the data based on its shader code. The fragments are sorted and filtered out; colour, shading and shadows are calculated and the final image is constructed.

To understand the solution to the problem of cutting complex data models at 90 FPS presented in section 6.2, it is important to understand that overlapping geometric meshes will all be rendered in full and they
will all create their own fragments. This leads to every pixel potentially having an array of fragments that can become the final pixel. Normally only the closest fragment to the camera is selected to become the pixel, but to achieve the accurate and fast cutting effect developed in this thesis a custom selection algorithm is used.

What is a mesh?

A mesh is represented as a series of data structures containing vectors and information on how they are connected in the model space. These data structures are called vertices and often contain information about the direction of the surface, the normal of the surface at that point, and the texture coordinates. Additional information can also be stored in the vertices that affect the behaviour and rendering of the mesh. These traits can dictate how other objects interact with the mesh and how the mesh is rendered. A texture is often used by a material to apply detail, colour and patterns. In this thesis the term material is used to mean the collection of any data or trait required to achieve the desired render; for example opacity, textures, roughness and specularity, and the shader code that uses it. The term mesh is used implying that a material is added.

Model, World and View space

There are three main spaces one talks about; Model space, World space and View space. The model space is also referred to as local space, the world space is also referred to as global space and the view space is also referred to as camera space. This thesis uses the terms local space, world space and view space to avoid confusion between the terms model and mesh as defined above. Affine transformation matrices are used to move between these spaces and they consist of a translation, a rotation and a scaling. When the terms transform or transformed or transformation matrix are used it is meant to represent any translation, rotation or scaling required to achieve the desired result.

The local space is the space within which the geometric mesh exists by itself. Each vertex of the geometry is defined as a vector from a common origin. This origin does not have to be inside the geometry of the mesh and each unique mesh has its own local space with its own origin.

The world space is the common space where all instances of every model share a common origin. Each object has its local space transformed by a matrix to place it in the world space. For example, if we have a simple box that has a dimension of $1 \times 1 \times 1$ and a centred local origin, we can make two instances of that box where one is translated to $<6, 5, 0>$ with a rotation around the world space X axis of 45 degrees. The other is translated to the other side of the YZ plane $<-6, 5, 0>$ with no rotation, but with a scale of $<2, 1, 1>$. The base mesh is the same, but each instance has its own transform that places the model in the common world space at different locations with different orientations and scales.
Figure 3.1: 1. The different parts that makes up the view space. 2. The view frustum cutting away the sphere as it is outside the frustum. 3. The world space with a shadow representing the occlusion of the second box. 4. The scene after it has been transformed to view space using a projection transform.

Finally, the view space is the space of the theoretical camera that we render to. To understand this space we first have to look into how the camera is defined in the rendering pipeline. In the world space the camera is simply put a frustum that determines what one can see. Some animals, including humans, have spherical pupils so we often talk about a cone of vision in regards to eye sight\(^1\). The camera, however, has a rectangle called the view plane. Figure 3.1 shows you the parts from which the frustum is mathematically defined in world space. The view plane has the same function as the back wall of a classical pinhole camera. The only difference is that since the camera is a mathematical construct we can place the plane in front of the hole at a distance of \(\frac{1}{2}\) and thus avoid the image being mirrored through its centre. Each pixel on the physical screen corresponds to a point on the view plane. The view frustum is a pyramid with the top corresponding to the pinhole and the sides of the frustum intersects the view plane edges at a distance of 1. Since computers can only handle discrete data we also define a near plane and a far plane along the Z axis that cuts the pyramid at two distances \(z_{\text{Near}}\) and \(z_{\text{Far}}\). Putting it all together we have the view frustum that defines the extent of the view space. The view space is the only space of the three that is restricted. The reason is that the view transform matrix takes the world space and transform it

\(^1\)For example: Figure 12 from this site http://www.dsource.in/course/product-drawing/perspective

\(^2\)The reason we place it at a distance of 1 is to simplify the mathematics. Please refer to the pinhole camera model
to the camera’s cubed space where the frustum becomes a $2 \times 2 \times 2$ cube around the origin. The cube goes from $<-1, -1, -1>$ to $<1, 1, 1>$. The frustum acts like a three dimensional cookie cutter that separates the world space into the inner space and the outer space by defining everything that is not within this $2 \times 2 \times 2$ cube to be not visible. The inner space is kept and rendered, the outer space is discarded. In figure 3.1(3-4) the ball is cut away since it is outside the frustum.
Chapter 4

Demonstrations and tester evaluations

There are several use cases that can benefit from a proper VR visualiser. This chapter will present the three demos that were developed for this thesis and the feedback received by testers. Working with researchers from the medical faculty of the University of Oslo two demos were created; The rat brain demo and the genome demo. The same researchers acted as primary testers of the visualiser. The artery demo was created based on artery data provided by a researcher at Simula Research Laboratory. Each model presented in this thesis has a distinct level of complexity and structure in order to assess how VR can be used with various data sets.

The feedback will be presented in a summarised form in this chapter, but an in-depth discussion of the implications of this feedback will be presented in chapter 7. Towards the end of this chapter some suggested alternative data types that can greatly benefit from VR will also be briefly mentioned.

4.1 Demos

Each data set has been processed and reduced to some degree before being rendered in the VR visualiser. To reduce a geometric mesh means to remove vertices, edges and faces to a lower total triangle count. A more complete description of the preprocessing steps will be presented in chapter 5.

Each demo can be scaled, rotated or translated using the controllers.

4.1.1 The Genome demo

The genome visualisation is based on the work of Paulsen et al. (2017). Knowledge about and research into DNA and the genome was limited to the DNA letters and the histones on one end and the chromosomes and the genome on the other. Gene clusters called Topologically Associating Domains (TAD) have become a major target for research. They form the middle layer in the ordered hierarchy of the genome. Mutations can cause TADs to break up or group up and thus lead to diseases. For
example, consider a TAD A and a TAD B. If they are not properly separated the proximity of the TADs causes genes that should not be activated or expressed in TAD B to also be affected by enzymes that were meant for activating and expressing genes in TAD A. Further, since TAD B is using some of the enzymes the degree of expression genes in TAD A experiences is altered. It is believed that a breakdown of the separation of these TADs leads to many of the malformations and diseases related to genetics. The borders between TADs are usually stable, but when they break or is located at an erroneous location the effect of the genes are altered. Understanding the 3D structure of these TADs has become very important for understanding cancer, mutations and gene interactions.

Model data

Figure 4.1: The genome data visualised inside the VR visualiser.

Paulsen et al. (2017) have developed a method to build a 3D model based on a probabilistic volumetric model that consist of many layers of contact matrices. These contact matrices uses a method called the Hi-C technique and it maps the chromatin in the nucleus. This new method was developed because even the most powerful microscopes were unable to get a clear picture of the chromosomes. The software they created, Chrom3D, allows users to build a geometric representation of the genome, chromosomes and the TADs. In figure 4.1 each sphere is a TAD. The geometry is coloured and shaded automatically in the VR visualiser.

The data was provided as CMM files (3.36 MB in total). CMM files contain an numerical ID, a position vector, a radius, a colour vector and two alphanumerical ID strings for the TAD and the chromosome. The geometry generated by Paulsen et al. (2017) totalled a size of 44.4 MB when exported
and consisted of 27,709 separate meshes totalling 3.7 million triangles and 2.3 million vertices. These were reduced and combined to 46 geometric meshes totalling 2.7 million triangles and 1.3 million vertices. Each TAD of a chromosome was combined to a single mesh.

Available interactions

Each mesh can be hidden or revealed, made translucent and opaque, and outlined. It can also be cut using the cutting tool developed for this thesis.

Importance of research

Understanding the structures of these TADs can become paramount to answering many of the unanswered questions related to mutations, hereditary traits and diseases and genetic disorders. Two major issues with the new method are the relative difficulty of navigating and understanding the topology of the geometric models and the difficulty many researchers are still faced with while using their software. Often researchers request a developer to test their hypothesis since the researchers lack the required skills to write the testing scripts themselves. A VR visualiser can significantly expedite the iterative process by enabling researchers to test many hypotheses themselves and by enabling researchers to achieve a much higher degree of spatial awareness within the genome. For example, a hidden chromosome’s outline can be rendered on top of the occluding geometry, giving the user a sense of relative scales and locations. Since the nucleoli of a genome is more active in terms of gene expression than other parts, being aware of a chromosome inside the core of the genome can be vital to understanding the whole.

4.1.2 The Rat Brain demo

Three-dimensional visualization of complex structures is an important tool in biomedical research and teaching, and a wide range of technologies are used to acquire, visualize, and analyse 3D images. Volumetric images are typically inspected in three-plane image viewers, or as geometrically rendered 3D objects that can be interacted with. In neuroscientific research, stereoscopic 3D visualization of selected objects has occasionally been used for publication purposes. At the Institute of Basic Medical Sciences several research groups study complex spatial structures in 3D. The Neural Systems laboratory at the Division of Anatomy develops database applications for image data. Their research revolves around computerized data acquisition, 3-D reconstruction, visualization and quantitative analyses of brain regions in the mammalian brain. These methods are used to investigate the architecture of large projection systems in the mammalian brain, as well as to study whole brain distribution patterns for gene, molecular and circuit level data.
Model data

The Neural Systems laboratory at the Division of Anatomy have manually segmented and labeled the raw MRI data of the brain (Papp et al., 2014; Kjonigsen et al., 2015). The volumetric MRI data set, the segmentation data set and the label data set provided for this thesis were used to develop a VR demo. The demo contained a complete, but sparse geometric model of a rat brain. Only a small selection of large structures were used in order to reduce the number of geometric meshes being rendered in VR and to minimise artefacts from reducing the high triangle count of the models. Large meshes can be reduced much more without losing too much structural detail. The resulting reduced geometry is coloured and shaded automatically in the VR visualiser.

The entire volumetric MRI data, segmentation and label files totalled 2 GB. 80 segments were exported to high-detail STL files. The 80 geometric structures were reduced from 16 million triangles at 1.1 gigabyte to 3.7 million triangles and 1.8 million vertices at 149 megabytes.

Available interactions

Each mesh can be hidden or revealed, made translucent and opaque, and outlined. It can also be cut using the cutting tool developed for this thesis.

Importance of research

The brain is a highly complex organ and its structure and function is still poorly understood. Brain research is needed to better understand and treat many different brain diseases. Due to the complexity of the brain, it is challenging to visualise the 3D structures that are easy to understand. For
students of medicine, dentistry, and nutrition it is an important to achieve a solid 3D understanding of structures at the level of molecules, tissues and whole organs. In this context computational rendering of structures can serve as a valuable aid for practical courses in microscopic and macroscopic anatomy.

4.1.3 The Artery demo

Intracranial aneurysms (IA) is a balloon-like bulge of an artery wall in the brain. An aneurysm can put pressure on nearby brain structures and may rupture. A ruptured aneurysm releases blood into the spaces around the brain and is a life-threatening type of stroke. There are four types of aneurysms: Saccular, fusiform, giant and traumatic. IAs are associated with disturbed hemodynamic flow, but much about the mechanics of IAs is still unknown. Many techniques for measuring blood flow exist, but they can be divided into two categories: Blood flow measured from imaging and blood flow obtained through subject-specific computational fluid dynamics. Simulations of blood flow through aneurysms can offer insight into factors relevant to the development and progression of IAs and their short and long term responses to treatments.

Model data

After artificially causing an aneurysm to develop the test animals were anaesthetised and then subjected to "an accelerated 4D PC MRA flow study with a radially under-sampled acquisition approach, PC-VIPR" (Jiang et al., 2011, p. 6301). The volumetric temporal data produced by Jiang et al. (2011) was sampled in two ways. First, a geometric glyph based mesh was generated for every time step. Each geometric glyph was scaled, oriented and coloured by the flow data of the artery. Colour and length represent the velocity of the flow at the base of the glyph. The orientation of the glyph reveals the direction of the flow at the base of the glyph. Secondly, the volumetric data was encoded into a 2D lookup grid to be used in the volumetric renderer implemented in the VR visualiser. This volume was colour coded similar to the geometric glyphs.

The volumetric data used to generate the UE4 compatible models was 3.97 MB in total. Since the artery model was the only model that contained an temporal dimension it is the only animated data set. It contains 17 distinct time steps. From the volumetric data set we generated 17 geometric meshes totalling 4.84 MB. In addition 17 volumetric lookup grids were generated by encoding 255 slices into a $16 \times 16$ flip book texture. Each time step texture had a size of 3.938 MB, which totals to a total of 66.946 MB.

Available interactions

Through a menu the volumetric and geometric representation can be affected. The model has 17 time steps and the temporal dimension is
Figure 4.3: The geometric and volumetric artery model visualised inside the VR visualiser. The cut separates the volumetric and the geometric model.

controlled through a menu that has a play/pause toggle, a forward and backward stepper and check boxes to toggle the geometric or volumetric representation. Both the volume and the geometry can be cut with the cutting tool developed for this thesis.

The render of the volumetric model can also be changed through four variables:

- Sample weight: The weight of the new sample. Higher weight means less of the accumulated colour is kept.
- Opacity: The final opacity after the pixel is rendered.
- Sample threshold: The threshold intensity that filters out low intensity samples.
- Sample Density: This value determines the number of samples taken.

**Importance of research**

A better understanding of the development and progression of IAs is important for preventive and early treatment. How aneurysms respond to treatments can help the development of safer and better treatment
alternatives. Flow fields are difficult to visualise in a manner that makes it easy for humans to understand. More accurate and intuitive data collection methods and more accurate and intuitive inspection tools are needed to achieve an increased understanding of IAs.

4.2 Feedback from tests performed in the field

Seven testers were asked to use the visualiser; three experts actively researching TADs and epigenetics, two experts in neuroscience, one developer with a medical background and one student studying neuroscience. Only the developer had any prior experience with VR. The testers had a varying level of familiarity with the data sets used in the rat brain demo and the genome demo. None of the testers had any experience with the artery model’s data.

The tests were performed on-site with their own VR setup. The demos were also available to be tested by researchers outside of the seven testers mentioned above. Feedback was regularly collected through meetings, notes and emails. The tests were not controlled as each tester was left free to use the VR visualiser as each tester was asked to consider six questions when providing feedback. These questions were:

- How can VR help visualise data used in your field?
- What visualisation challenges does VR provide a superior solution to?
- What aspects of visualising your data would VR not be applicable to?
- What features would be required from a VR tool before you would be willing to adopt VR as an alternative to monocular systems?
- What are the main issues hindering adoption of VR tools given current HMD technology?

The feedback is summarised and separated into demo specific feedback and common feedback.

4.2.1 Demo specific feedback

Genome demo

The VR visualiser was compared with the Chimera package from the Computer Graphics Laboratory, University of California, San Francisco (supported by NIH P41 RR-01081)\(^1\) for this demo.

During the testing the overall feedback was positive. The testers were excited with the possibilities and benefits VR could provide their research. The main benefit of VR in their specific research was linked to the ability of VR to intuitively show the user where specific genes are related to (1) other

\(^1\)Available at https://www.cgl.ucsf.edu/chimera/
genes, (2) regulatory DNA-areas and (3) the membrane of the cell nucleus. Negative feedback was either related to physical sickness and fatigue, or to the limitations of the implementation presented in this thesis.

**Rat brain demo**

The VR visualiser was compared with the ITK-SNAP (Yushkevich et al., 2006) for this demo.

VR was described as "a highly exciting and promising technology with considerable potential value for biomedical teaching and research." The only demo specific feedback provided was negative. The quality of the geometric models were inadequate, there were many missing features that the testers would consider necessary and the design must be reworked to be more user friendly, powerful and pedagogic.

**Artery demo**

The VR visualiser was compared with Paraview (Ahrens et al., 2005) for this demo.

It is important to note that none of the testers were experienced with the type of data used in the artery demo. The feedback provided was given by medical professionals of other fields.

Little was said about the artery demo by testers. Testers found it hard to comment because the simple structures of the artery demo offered little to no advantage over the monocular visualisation. More complex visualisations of the flow field can benefit greatly from VR. Another point brought up was the lack of a need to enter the model. They found they could get a good understanding of the structure of the artery and the aneurysm by viewing it from an outside perspective. If the interesting data covered a larger volume it might have been very beneficial to view it in VR since obfuscation and depth becomes very real issues for the user. This demo was by far the least interesting of the three, but the testers voiced an interest in inspecting larger and more complex volumetric data instead.

It appears that the negative feedback and the lack of positive feedback regarding the artery demo was due to a poor choice of data to visualise. The model was too structurally simplistic and it is worth re-evaluating flow field visualisation with structurally more complex data and with more complex visualisation algorithms like for example streak, stream and path lines.

**4.2.2 Common feedback**

The effect of having a bright screen shining into your eyes is well known for anyone who have experienced it. It tires the eyes quickly and can be hurtful. Having a generally dark scene with few distracting elements is important for extended and focused use. Some users have indicated that having something more welcoming like a sky would be good to have as the
virtual world appears real enough to cause feelings of worry when faced
with a pitch black void all around them.

All testers agreed that using the cutting tool and handling the model’s
transform were much more intuitive and easy to use in VR. The accuracy
was inferior to what could be achieved with monocular systems, but none
of the testers found it to be an issue. The demos presented in this thesis
had too few features to properly test the effect of crowding the controller
with features. However, feedback indicate that the basic interactions,
that is scaling, rotating and translating, benefited from the using the VR
controllers as implemented in this thesis.

All testers agreed that VR could offer much just through its ability to
make the spatial understanding of complex models much easier. Not only
does VR provide a superior understanding of the structure as a whole,
but the relative scales, orientations and locations were also much easier
to understand. The epigenetics researchers who tested the VR visualiser
predicted that a VR visualiser would expedite the iterative process of
testing hypotheses by allowing for early validation of such hypotheses. The
intuitive nature of VR and the superior spatial awareness the user obtains
makes it much easier to identify which hypotheses are flawed and which
hypotheses are plausible.

Controlling the camera was found to be very natural and much faster
than controlling the camera in any monocular software the testers had
tried. Testers mentioned two possible reasons for why controlling the VR
camera was superior; it was as natural as inspecting something in real life,
and it was possible to translate and rotate the camera continuously and
simultaneously. It was also mentioned that similar to how a human has a
sense of where his body parts are located and oriented without direct visual
confirmation, VR provided a sense of where the testers head was in space
and by extension also where the camera was.

Another reported benefit of VR is a combination of the previous two.
Following structures in VR was much easier than using standard 2D
interfaces. Testers reported that accurately following structures could take
up to minutes in standard monocular solutions, but only required seconds
in VR. Less confusion was also generated by complex or large movements.

Operating menus in VR was challenging for several testers because of
unsteady controllers. The controller tended to move when the testers
pressed the selection button. Since the attached laser pointer was pointing
on the menu element the testers wanted to interact with before the press,
the small rotation of the controller resulted in a sufficiently large translation
of the laser point to miss the intended target. Similarly, models had a
tendency to be too unsteady for proper inspection when held.

Each of the demos were reported to have too few features and they were
too limited for their use. The testers agreed with each other that this was an
issue related to the solution presented in this thesis, not to VR as a whole.
Mentioned missing features included the ability to generate new segments
and surfaces from a cut, the ability to extract information about and select
single components of the chromosome and the ability to load objects in
runtime.
Concerns related to data set sizes and data resolution was mentioned. The genome had 46 meshes; one for each chromosome, but it did not support extracting data about a single TAD. The artery model also had a low resolution volume that raised concerns about being able to render more complex volumes. Many large data sets will not fit in the GPU memory. This might become a problem if streaming the data in chunks would prove to be too slow.

Finally, the lack of multiuser support limited the usefulness of the VR visualiser, especially for educational purposes. It was very hard to try to teach across the barrier between immersed and non-immersed testers.

4.3 Alternative use cases presented by testers and correspondences

One of the use cases the author was given during a visit to the medical faculty at the University of Oslo was visualisation of neurons. One model showed a single neuron by itself in high detail while the other showed several interconnected neurons in a neural network. These were all volumetric in nature. A neural network is a complex system of interconnected cells in a three dimensional network. This complexity utilises and benefits from VR to a similar degree as the genome demo does.

The other use case presented was an educational museum experience that had several models on display. The examples given were sub systems of the brain. Examples given were the auditory system and the neocortex segmented into its sub-components. Testers also said that it would also be of interest to be able to show a semitransparent shell of the brain placed such that the user can see where the sub systems were located. By placing all models in the same scene the users could interact with the model of their choosing.

Path, streak and stream lines would also benefit from VR as shown by Ohno and Kageyama (2007). The genome backbone model has many similarities with fields and show how fluid dynamics can benefit from VR. Encoding depth and dealing with line clutters should become less of an issue with VR.

Finally, geologic fault models could benefit from VR. According to a personal correspondence with an American student in the field, many professors at the student’s university expressed an interest in using VR for geological visualisations. For example, understanding how the fault lines are placed relative to one another is important information to better understand where earthquakes originate. A better understanding of the fault lines is important for predicting such quakes.
Chapter 5

Importing scientific data files into UE4

This chapter outlines the steps taken and alterations made to ensure stable framerates while in VR in UE4. It addresses the challenges of rendering the high detail of scientific data as well as the challenge of converting a 3D volume to a 2D flip-book texture.

Figure 5.1: These are the manual steps taken to convert the scientific data to FBX mesh files and texture files. Green represents geometric data and light pink represents volumetric data. The nodes on the left represent the raw data and the rest represent programs used during the conversion.

Most data and file formats that are used in the scientific world are not supported by UE4. The only 3D geometric model file types it supports are FBX and OBJ, and there are no readers for non-geometric models. Most common image formats are supported; .bmp, .png, .psd, .tga, .jpg. All models must be converted to one of the file types it supports or loaded during runtime by writing a custom reader for UE4 in C++. For this thesis a manual conversion was performed, but future versions can include a
export/import system. The Assimp library\(^1\) is open source and can be linked to UE4.

Figure 5.1 shows the data import steps from left to right on the example of the three data sets used in this thesis. Each connection represents a conversion of file type. Each colour change represents a conversion of data type; geometric to volumetric or vice versa. The arrow symbolises a change of software without a file type change.

Common modifications that had to be done to the meshes were ensuring centred or coinciding origins, also known as pivots, for each mesh used in the same model. Centred means that the pivot is in the centre of the bounding box of the mesh or model, and coinciding means that each mesh has the same origin. Another common issue was the size of the meshes. Often the models were too large or small, causing floating calculations to slow down UE4. Finally, the poly count had to be reduced to around four million triangles per demonstration to ensure the required 90 FPS could be reliably kept.

### 5.1 Exporting genome data as geometry for the Genome models

The genome model was based on a series of CMM files (Chimera markers) opened in the Chimera package from the Computer Graphics Laboratory, University of California, San Francisco (supported by NIH P41 RR-01081). Chimera could export structures as DAE files that Autodesk 3DS Max\(^2\), hereafter 3DS Max, supports. 3DS Max was responsible for the alterations made to the geometry and structure and for exporting it to the FBX file format understood by the UE4 editor.

Each CMM file contained the data required to generate a geometric model for a single chromosome, which was locally translated and rotated such that all chromosomes shared a common origin. This simplified the following steps as no manual placement of the chromosomes were required. The DAE files Chimera exported contained many meshes per chromosome. In total the entire genome consisted of 46 chromosomes, constructed from 27,709 spheres. This proved to be too expensive due to the cost of that many draw calls in UE4, causing the frame rate of the visualisation to drop as low as 12 frames per second. 3DS Max reduced the number of meshes to 46, one for each chromosome, and the number of triangles to around four million triangles. The commands used were called Collapse and Optimize respectively. Welding the triangles did not produce a noticeable effect. The modified meshes were exported as FBX files and sent to UE4.

\(^1\)http://www.assimp.org/
\(^2\)Available at http://www.autodesk.com/products/3ds-max/overview
5.2 Exporting segments of a rat brain atlas as geometry for the Rat Brain model

The geometric meshes used for the rat brain model were extracted from a volumetric and segmented atlas. ITK-SNAP was used to export each segment of the brain as an STL file. These geometric meshes were then opened in Blender\(^3\) to be converted to OBJ or FBX files. 3DS Max imported the models and performed all modifications made to the geometry and structure.

ITK-SNAP requires three files to segment and label the models; the atlas and a segmentation file, both stored as NII files, and a LABEL file for the labels. When all files are loaded the program lets the user select a segment to export and generate a geometric hull along the boundary of the segment. Due to instability experienced with 3DS Max using all 16 GB of RAM available on the computer used for development, Blender was used to first convert the files to FBX files. These FBX files caused no issues when imported into 3DS Max. Since these meshes were too detailed, they needed to be reduced and transformed in 3DS Max. The meshes were reduced such that the entire model consisted of 4.5 million triangles. Most of the meshes had to be transformed such that each segment was where it should be inside the model. For some reason the exported meshes were of several relative scales and heights and a lot of manual work went into moving and scaling the meshes to match the volumetric model seen in ITK-SNAP. Properly processed, the model was exported as an FBX file and sent to UE4.

5.3 Exporting volumetric data of an artery with an aneurysm for the Artery model

The artery was provided as an XDMF file and was opened using a Paraview python script. After applying the filters and transfer functions the script stored several images to disk where each image was a slice of the volume. These images were stitched together into a grid as seen in figure 5.3 using a standalone standard python script. These super images were then imported into UE4.

After setting up the visualisation in the Paraview editor one can export the filters and encodings as a Paraview python script. Please refer to code 8.1 in the appendix for a complete example. To extract the N slices needed from the volume the camera and the slice filter were moved through the entire volume at a fixed distance, as seen in figure 5.2, and a screenshot was saved to disk for each slice. The entire pseudo code can be seen in algorithm 5.1.

\(^{3}\)Available at https://www.blender.org/
Algorithm 5.1: Added to the basic exported python script are two loops; one for each time step and one for each slice of that timestep. Full example code: Algorithm 8.2 in the appendix.

All images for each timestep were stored in the same folder. The images were then tiled using standard python into a super image as seen in figure 5.3. The grids were sized to the smallest square number, $2^n \times 2^n$, larger or equal to the number of images to simplify the ray marcher, but any number can be made to work and would be preferable with non-square slices.

As seen in figure 5.3, data is already processed and encoded into the image using RGBA colours. This simplifies the ray marcher discussed later in section 6.1. The compiled images can be imported as textures into UE4 and will be referred to as such. Each texture acted as a flipbook texture that the ray marcher sampled from.
5.4 Exporting flow field as geometric arrow glyphs for the Artery model

It was also required to visualize the glyph representation of the artery. To achieve this, a glyph filter was applied to the artery volume. For each timestep, the geometric glyphs were exported as a PLY file and imported into Blender. Blender converted it to FBX to enable 3DS Max to read the meshes. 3DS Max collapsed the meshes and centered the origin of the models and then exported the meshes to UE4.

Most software that support volumetric data sets support some kind of geometric model generation. Paraview had a filter that generates an encompassing surface mesh or a series of glyphs that represent the vector data. Both filters were applied to the flow field and the resulting meshes were geometric arrows where length, color, and direction expressed the direction and velocity of the flow. These meshes were exported as PLY files and converted via the free software Blender since 3DS Max had no support for the format. It was important to translate the model pivots, the origin of the mesh, such that it was located in the center of the mesh. This simplified working with the models once inside UE4. It was also important that all meshes share the same origin to ensure a smooth animation when changing timesteps.
Chapter 6

Implementation of a Virtual Reality Scientific Visualiser in UE4

To create a test scientific visualiser it needed to be able to render both volumetric and geometric data and to enable basic interactions like scaling, translation and rotation. Some features of already existent 2D alternative visualisers were implemented; A cutting tool that cuts the models, menus that extend the input options past the limits of the controllers, a tool for measuring distance and a tool for extracting the name of a component of a model. Additionally two VR specific features were implemented; a teleport mechanic that allows for large user translation and a new and innovative method of visualising internal structures through

In order to implement a virtual reality scientific visualiser with UE4 it was extended with seven main components:

- A ray marcher - 6.1
- A cutting tool - 6.2, 6.3.4
- GrabberComponents/Grabbable classes - 6.3.1
- Control Schema for the controllers - 6.3.2
- Interactive menus - 6.3.3
- Name Extraction and Distance Measurements - 6.3.5, 6.3.6
- Teleportation - 6.4
- Outline postprocess material - 6.5

During the thesis, it became evident that implementing a scientific visualiser in VR posed as many design challenges as implementation challenges. The unique restrictions of VR demanded new approaches to software design while the unique power of VR made new avenues for data inspection possible.
The major restrictions on the design of the VR visualiser were the cost of rendering a VR scene and the lack of input options. Further, the available input options were perceived less precise. Complex input, for example button combinations and text and number input, which are common in 2D toolkits, was slow or awkward in an VR environment. This lack of user input options and input fidelity limited the amount of features possible to activate at once and all menu based interaction was limited to only sliders and buttons. The other major hindrance to reusing algorithms designed for standard monocular systems is the 90 FPS requirement of VR and the increased cost of rendering VR makes most algorithms too slow. 10 MS is insufficient to complete many commonly used algorithms with commonly used data sets.

Implementation and design of solutions to the challenges of creating a VR visualiser will be presented in this chapter. Brief justifications of key design and implementation decisions will be provided with a discussion of abandoned alternatives if applicable. An analysis of bottlenecks encountered and how they were solved is given in 6.6.

A more in-depth analysis of the VR Visualiser and VR tool design in general will be presented in chapter 7.

6.1 Implementing a ray marcher in UE4 using encoded textures and the fragment shader

![Diagram of the ray marcher](image.png)

Figure 6.1: The design of the ray marcher implemented in this thesis. It is implemented in the fragment shader and consists of two main steps; setting up the environment for the generic ray marcher and the ray marcher itself.

Ray marching is a simpler version of ray tracing since scattering or reflection of the ray is not considered. It is a simplified system where all rays only travel in a straight line. Since the environment does not influence
the render, much of a ray tracer can be skipped. This can safely be ignored since the main benefit of most advanced ray tracers is the realistic scene renders. The scientific data rendered in this thesis was stylised and isolated.

![Figure 6.2: 1. The ray hitting the $1 \times 1 \times 1$ volume. 2. Shows the samples taken within the volume.](image)

To circumvent the limitation of UE4, a way to create a volumetric effect in an engine with no volumetric support had to be implemented. In a ray marcher the ray is emitted from the camera and passes through the view plane at a location that coincides with the pixel position in screen space. This ray is represented as a location vector for the origin (camera) and a unit vector that dictates the travel direction. If the ray intersects the volume we perform four steps to visualize the volumetric model:

1. Calculate the entry and exit point
2. Calculate the number of steps required to traverse the entire volume
3. Sample by converting the 3D volume location to a 2D texture location
4. Blend the extracted sample with the previously cumulated colour

As seen in figure 6.2 the ray is travelling through the volume(1) and samples several location inside the volume(2).

**Calculating the entry point and exit point of a volume**

As seen in figure 6.3, the entry point is called $T_0$ and the exit point is called $T_1$. The problem of calculating the intersection can be simplified to a series of one dimensional problems where each axis is solved by itself. The problem was further simplified by choosing a coordinate system that the volume is aligned with. To find the number of ray unit vectors required to reach the entry point $T_0$ and the exit point $T_1$ we used a two step calculation. For each dimension the distance from the camera to the volume bounds, defined by the points $\vec{A}$ and $\vec{B}$ in figure 6.3, is calculated. First two vectors with the candidates are generated. One vector holds the smaller scalar for each dimension.
Let $\vec{Ray}$ be the ray unit vector and $\vec{Ray}_x$, $\vec{Ray}_y$, and $\vec{Ray}_z$ are the X, Y and Z components of the unit vector $\vec{Ray}$.

Finally, let $\vec{C}$ be the location of the camera.

$$
\begin{align*}
T0\text{Candidate} &= \begin{pmatrix}
\min(\vec{A}_x - \vec{C}_x, \vec{B}_x - \vec{C}_x) \\
\min(\vec{A}_y - \vec{C}_y, \vec{B}_y - \vec{C}_y) \\
\min(\vec{A}_z - \vec{C}_z, \vec{B}_z - \vec{C}_z)
\end{pmatrix} \\
T1\text{Candidate} &= \begin{pmatrix}
\max(\vec{A}_x - \vec{C}_x, \vec{B}_x - \vec{C}_x) \\
\max(\vec{A}_y - \vec{C}_y, \vec{B}_y - \vec{C}_y) \\
\max(\vec{A}_z - \vec{C}_z, \vec{B}_z - \vec{C}_z)
\end{pmatrix}
\end{align*}
$$

By dividing $\vec{A} - \vec{C}$ and $\vec{B} - \vec{C}$ with the corresponding component of the directional vector of $\vec{Ray}$, the smallest and largest $\vec{Ray}$ scalar can be calculated. The smaller scalar represents the number of $\vec{Ray}$ needed to enter the bounds while the larger scalar represents the number of $\vec{Ray}$ needed to exit it. The largest of the smallest scalars and the smallest of the largest scalars are selected as $T0\text{Scalar}$ and $T1\text{Scalar}$ respectively.

$$
\begin{align*}
T0\text{Scalar} &= \max(T0\text{Candidate}_x, T0\text{Candidate}_y, T0\text{Candidate}_z) \\
T1\text{Scalar} &= \min(T1\text{Candidate}_x, T1\text{Candidate}_y, T1\text{Candidate}_z)
\end{align*}
$$

The actual vector values, as seen from the camera, are then given by the following calculation

$$
\begin{align*}
\text{EntryPoint} &= T0\text{Scalar} \times \vec{Ray} \\
\text{ExitPoint} &= T1\text{Scalar} \times \vec{Ray}
\end{align*}
$$

This works because the smallest values all guarantee that the ray has intersected with the box at least once; its own dimension. Thus, by selecting
the largest of them it can be guaranteed that the ray had intersected with every dimension and thus that the point was not on the outside of the box. 

T0y is the scalar required to intersect the box for the first time along the Y axis in figure 6.3(2). The ray is inside the box bounds along the Y axis, but it has yet to intersect the box along the X axis. Only after the ray has intersected the bounds along all three dimensions will the ray be inside the volume. That is why the largest of the smallest values is selected. This principle can be reversed to determine T1Scalar. T1y is the larger value and represents where the ray leaves the box along the Y axis. However, as it has left the bounds already along the X axis it is not a valid value for the exit point. The first time it leaves the bounds along any axis the ray is also leaving the volume. As such the smallest of the largest values is selected as the scalar for the exit point.

The code output of this calculation is a four dimensional vector where the first three floats represents the location in world space of the entry point. The arrow in figure 6.1 represents this data. Let us call this position Entry Point. The last float was a scalar value that told us how many direction vectors there are between the entry and exit point of the volume. This value is later used to calculate how many steps are required to traverse the distance the current ray covers inside the volume. Let us call this scalar value Travelled Distance.

Calculating the number of steps required to traverse the entire volume

The Travelled Distance multiplied with the Max Steps, a variable set in the software by the user, provided the number of steps required to traverse the volume. Since the distance was calculated to a floating point the result of the above multiplication would usually result in a number that was not a natural number, necessitating a flooring of the value to obtain the actual number of steps required. The remainder was used for the last step which smoothed the edges in some cases.

\[
\begin{align*}
\text{Temp} &= \text{TravelledDistance} \times \text{SampleDensity} \\
\text{NumSteps} &= \lfloor \text{Temp} \rfloor \\
\text{FinalStep} &= \text{Temp} - \text{NumSteps} \\
\text{StepSize} &= \frac{1}{\text{MaxSteps}}
\end{align*}
\]

Sampling the ray by mapping a 3D volume location to a 2D texture location

One major hindrance encountered when using UE4 was that it did not allow for any three dimensional data sets to be sent to the GPU. There are no custom datasets that can be sent without reworking major parts of the rendering engine as a lot of the power of the GPU is not exposed to the engine user. To achieve a volumetric effect and to be able to still read
three dimensional data we had to encode the volume into sheets of two dimensional planes stacked in a grid.

For each step through the volume the data associated with that point was processed and blended into the accumulated values. This thesis used a flipbook texture as seen in figure 5.3 where each tile was a slice of the volume.

![Figure 6.4: An example grid of size $8 \times 8$. Each slot corresponds to a slice in the volume.](image)

On the GPU the texture would always have a size of $1 \times 1$ where it spanned the range of $0 \Rightarrow 1$ in both X and Y dimensions. Each plane in the Z dimension was placed with the first plane in the top left corner followed by each subsequent plane added going right and down as seen in figure 6.4.

It is possible to generate a grid that has a different number of tiles per side, for example $8 \times 4$, but it was opted to use encodings where the number of total tiles can be written as $n^2$ where n is a natural number. This limitation simplified the volume renderer code as only a single variable was required.

A texture in UE4 has a maximum size of $8192 \times 8192$ demanding a balance between the resolution per plane and the number of planes we want. Another way to view the problem is that the resolution between the axes must be balanced as increasing the resolution along one axis demands that another axis suffers a decreased resolution. This was due to the fact that the maximum allowed data was constant: $X \cdot Y \cdot Z = 8192 \cdot 8192$.

The mapping itself was straightforward. The Z value of the sample point in volume space was converted using a 1D to 2D conversion, seen at the top of algorithm 6.1. The XY coordinates can be used as is as UV coordinates, otherwise known as texture coordinates. Since most samples did not land on a single plane two planes had to be sampled, the one above
and the one below, and linearly interpolated between.

```cpp
float2 Convert1dto2d(float XSize, float idx)
{
    float2 xyidx = 0;
    xyidx.x = fmod( idx, XSize );
    xyidx.y = floor( idx / XSize );
    return xyidx;
}
```

```cpp
//** Tex **// Input Texture Object storing Volume Data
//** inPos **// Input float3 for Position, 0-1
//** xsize **// Input float for num frames in x,y directions
//** numFrames **// Input float for num total frames
float4 PseudoVolumeTextureColour(Texture2D Tex, SamplerState TexSampler,
float3 inPos, float xsize, float numframes)
{
    float zframe = floor( inPos.z * numframes ); //Changed to floor from ceil.
    float zphase = frac( inPos.z * numframes );

    float2 uv = frac(inPos.xy) / xsize;

    float2 curframe = Convert1dto2d(xsize, zframe) / xsize;
    float2 nextframe = Convert1dto2d(xsize, zframe + 1) / xsize;

    float4 sampleA = Tex.SampleLevel(TexSampler, uv + curframe, 0);
    float4 sampleB = Tex.SampleLevel(TexSampler, uv + nextframe, 0);
    float sampleLen = length(sampleA.xyz)*length(sampleB);
    float4 sample = lerp( sampleA, sampleB, zphase );
    if(sampleLen < 0.01){
        sample.a = 0;
    }
    return sample;
}
```

Algorithm 6.1: The code above comes with courtesy from Ryan Brucks, the Principal Technical Artist at Epic Games, who created this custom sampler for his own projects. The only change we performed was to change the ceil function call to a floor function call.

The index of the tile was given by multiplying the 0 \( \Rightarrow \) 1 Z-component with the total number of tiles: 4 \( \Rightarrow \) N. To calculate the column and the row of a Z position the renderer uses a standard 1D to 2D translation by using the width of the grid with fmod and with integer division. Since each tile occupied a space of size \( \frac{1}{T} \times \frac{1}{T} \) where T is the number of tiles, the start of the subtile was found by multiplying the result of the 1D to 2D conversion with the width of a single tile. The final step was to add the XY components.
to find the desired sample texel. Since the volume space goes from $0 \Rightarrow 1$ while the texture has $T$ tiles in the same range we have to divide the UV coordinates by $T$ to get the actual texel location.

\[
\text{AllTiles} = \text{TilesPerSide}^2
\]

\[
\text{OffsetStep} = \frac{1}{\text{TilesPerSide}}
\]

\[
X_{\text{Offset}} = (\lfloor Z \ast \text{AllTiles} \rfloor \mod \text{TilesPerSide}) \ast \text{OffsetStep}
\]

\[
Y_{\text{Offset}} = \frac{\lfloor Z \ast \text{AllTiles} \rfloor}{\text{TilesPerSide}} \ast \text{OffsetStep}
\]

\[
\text{TexelPosX} = X_{\text{Offset}} + \frac{U}{\text{TilesPerSide}}
\]

\[
\text{TexelPosY} = X_{\text{Offset}} + \frac{V}{\text{TilesPerSide}}
\]

With the texel coordinates the card will return the desired texel’s data. However, since the sample would usually be located between two tiles, as seen in figure 6.5, two samples must be extracted. Once both sub-samples are obtained they can be blended by using the fractional component of the floating index, inPos.z * numframes in the code above, as the alpha in the linear interpolation.

![Figure 6.5: A sample along the ray defined by a blend between a sample from both encapsulating slices.](image)

Since UE4 allows for 32 textures at the resolution of $8192 \times 8192$ each where each texel holds 4 bytes of data, namely RGBA, it was possible to use a total of 8.192 GB of data. However, in order to use all that data the solution would have to add another two layers to the mapping from the three dimensional volume space to the two dimensional texture space; calculate which texture to access and which byte in the four byte array to
use. Further, all this data had to be present on the GPU or streamed in. Today there are only few cards that can hold 10 GB of data or more at once and streaming all that data was too slow to allow for the required 90 frames per second. Based on initial feedback from researchers this thesis held no interest directly. Incorporating the solutions developed for this thesis into already existing solutions was preferable. As such it was decided that a simple system using only a single texture was adequate for the demonstrative purposes of this thesis.

**Accumulating samples through alpha-beta blending**

In a standard volumetric rendering pipeline a series of functions and tables are applied to the extracted sample to encode it for rendering. Commonly used blending and encoding methods include transmittance and density, scattering and absorbency, and maximum intensity projection (Sabella, 1988; Govaerts and Verstraete, 1998; Blasi, Le Saec and Schlick, 1993; Jensen and Christensen, 1998; Parker et al., 2005; Schreiner, Paschal and Galloway, 1996). For this thesis the data was already encoded from the steps described in chapter 5.2 so only the blending of the coloured samples remained. The cheapest and easiest way to achieve this was through alpha-beta blending.

\[ \text{Colour} = (\alpha)\text{Colour} + (1 - \alpha)\text{Sample} \]

Since the transfer functions and all the complexity of encoding data in the ray trace was encoded already in the RGBA values of the texture many of the instructions that would normally slow down a render could be skipped. The trade off was a reduced ability to affect the encoding of the data, and thus the visualisation.

**Using an inverted box to allow the user to enter the volume**

Due to the way back-face culling works a standard box with the surface normals pointing outwards would completely vanish if the user were to enter the box. By flipping the surface normals the 2D shape of the box would be preserved while viewed from outside and the user would be able to see the volume from inside as well. By allowing the user to enter the volume the Entry Point of the ray was placed behind the camera when the camera was inside the volume. The scalar T0 would become negative and the part of the volume that was behind the camera would also be blended in with the part of the volume in front of the camera. By using the result of \( \min(T0, 0) \) as the effective \( T0 \) the resulting render would only include the part in front.

**Adding support for intersecting geometric and volumetric models while optimising the ray tracer**

To stop the geometric models from blocking the volume as seen in figure 6.7(1) the depth test was disabled. Without the depth test the volume was
always drawn on top of the rest of the scene, figure 6.7(2), which hid the environment from from the user. Allowing geometric models to stop the ray early was achieved in two steps: 1. Queuing the volumetric model for rendering after the geometry had been rendered. 2. During the calculation of the entry point and travelled distance in section 6.1, $T_1$, the scalar for the exit point, was clamped between zero and the depth value at that pixel to terminate the ray early, figure 6.7(3, 4).

By terminating the ray early a slight decrease in render times was recorded with intersecting geometry, but due to the FPS lock it did not increase the FPS. The optimisation comes for free, but it only effected render times in when geometry intersects the volume. Further, situational improvements proved to be without interest as the worst case render times
had to fit within the 10ms time budget.

**Space optimisations turned out to not be applicable to the UE4 Ray Marcher**

Additional algorithms and data structures for optimising the visualisation exist, but most of these techniques focused on skipping as much of the volume as possible (Sherbondy, Houston and Napel, 2003; Levoy, 1990; Parker et al., 2005). Techniques such as space subdivision and volume segmentation allowed other ray tracers to use a more detailed space for the closest part of the volume as well as removing large areas that contain nothing of interest. With the technique used in this thesis a potential benefit from skipping the empty space by using a geometric model to dictate the starting depth of the ray could be implemented. The same depth buffer could be used to terminate rays early when blocked by opaque geometry. The ray could also be terminated when further sampling will not effect the final colour to any significant degree because of the density of the volume and the distance it has travelled through it.

Due to the lack of custom data structure support only three options were available in UE4. Early termination due to intersecting geometry, early ray termination due to loss of sample impact and empty space skipping. Both the skipping of empty space and early ray termination due to opaque geometry require the depth buffer. Depth based ray termination was selected since it enabled geometric models and volumetric models to be used together. UE4 seems to only have a single custom depth buffer forcing the choice above.

The other optimisation techniques, for example BSP space subdivision and oct trees, relied on a re-computation of custom data structures and could not be applied to this thesis.
6.2 The cutting effect

Figure 6.8: The left image shows a geometric model being cut while the right image shows a volumetric and geometric hybrid being cut.

To enable slicing a tool was placed in the scene. This tool consisted of a large semitransparent plane attached to a white handle. How the user could interact with and control the tool is described in section 6.3.4. Being provided with the location of a point on the cutting plane and the normal to that plane the GPU mathematically sorted all points into three groups; under the plane, on the plane and over the plane. Once sorted the unwanted points were discarded. For geometric meshes this translated to discarding fragments that are being considered for rendering. For the volumetric case it translated to discarding samples while ray marching.

Figure 6.9: Left: World Space. Right: Volume space.

The geometric fragments’ location were given in world space which added the benefit that no transform had to be applied and the cutting plane normal and position could be sent in as it was. Since all volumetric calculations happened in the volume space, figure 6.9 shows how the normal had to be corrected by applying the volume’s inverse rotation
matrix and how the location was corrected by applying the entire inverse transform.

Every frame the location and the normal vector were updated and sent to the GPU. By keeping it visual only, the performance impact was near zero even with 90 updates per second.

**Cutting Geometric meshes in World Space**

![Diagram](image)

Figure 6.10: Each object has its own local coordinate system while they share a world coordinate system. The blue area is the view space which determines what the camera can see.

UE4 provided a version of all standard vectors in world space by applying the inverse of the World to View transform matrix to each vector. Figure 6.10 shows how this simplifies the problem by removing the need for moving between multiple spaces. Using the fragment’s location(green) and the cutting plane’s location and normal the fragment shader used the dot product to sort the fragments.

Let $\vec{CPL}$ be the location of a point on the cutting plane and let $\vec{CPN}$ be the normal to that cutting plane. Let $\vec{FL}$ be the location of the fragment. Another vector can then be defined as $\vec{CPtoFrag} = \vec{FL} - \vec{CPL}$.

$$\text{DotProd} = \vec{CPtoFrag} \cdot \vec{CPN} \quad (6.1)$$

The DotProd’s value sign was used to branch the execution to mask out one side of the plane. In the case of figure 6.10 only Cube1 would be visible.

Since any angle lower than 90° will have a positive cosine value and all angles over 90° will have a negative cosine value the vectors did not require normalisation since we only care about the sign bit. This saved a few instructions per iteration in the ray marcher loop.

To mask something is the act of filtering out a sub-set of fragments from the render pipeline. This is often done in UE4 by using textures where $R$,
G, B and A can sort each fragment into two groups each. Unlike boolean values a mask in UE4 is invisible for all values under 1. To stop the mask from removing the fragments that occupied the space where the DotProd was between 0 and 1, the branching statement rounded up the value:

$$\text{mask}(\text{prod}) = \begin{cases} 1, & \text{if } \text{prod} > 0 \\ 0, & \text{otherwise} \end{cases} \quad (6.2)$$

This roofing resulted in the effect seen in figure 6.8. The above masking causes the fragment to be prematurely discarded and allowed the closest remaining fragment to reach the screen. In other words, since the fragments would not update the depth value if they were masked out, the now visible geometry would not fail the depth test. Finally, the material disabled the face culling such that the inside of the models were visible when cut.

**Transforming the plane to and cutting the volume in Volume Space**

The volumetric version of the cutting tool used the same principal for sorting samples, but a few additional steps complicated the solution.

To first simplify the problem as much as possible the basic shape of the volume was set to a $1 \times 1 \times 1$ geometric inverted cube going from $<-0.5, -0.5, -0.5>$ to $<0.5, 0.5, 0.5>$. This allowed the shader code to utilize the world bounds provided for us by UE4 to calculate the $\vec{CPLocation}$ in volume space.

$$\vec{CPLocation} = \frac{\vec{CPLocation}}{\text{VolumeBounds}} \ast 2 + 0.5 \quad (6.3)$$

The VolumeBounds variable would increase as the world scale of the volume actor increased, normalising the distance to the cutting plane in world space. By not normalising it the cutting effect would always cut as if the volume was at a scale of 1. The addition of 0.5 at the end offset the local space to match the volume space. Once in volume space each sample along the ray checked if that position lies on the ignored side of the plane or not.

Let $\vec{CPL}$ be the location of a point on the plane and let $\vec{CPN}$ be the normal to that cutting plane. Let $\vec{SL}$ be the location of the sample. A new vector can then be defined as $\vec{CPtoFrag} = \vec{SL} – \vec{CPL}$\(^1\). The product of interest is given by formula 6.1. The alpha of the sample multiplied with the result of formula 6.2 effectively remove it from the blending queue if on the hidden side of the plane.

Figure 6.2(1) shows a single ray from a camera being sent through the volume. As mentioned before, width, length and height were all of length 1. The entry and exit points are visualized in the figure 6.2(2) as the blue(entry) and red(exit) lines. Once the entry and the exit point are

\(^1\)Note that the vector is still called $\vec{CPtoFrag}$ even though it goes from the plane to the sample. This is done to make it work with the previously defined formula.
calculated the number of fixed distance steps needed to cover the inter-
volume part of the ray was calculated as seen in section 6.1.

For each step, after advancing the ray position and sampling the
volume texture, the sample was discarded if it failed one of the following
conditions:

1. The sample intensity must exceed the threshold
2. The sample must be on the correct side of the cutting plane

The threshold was a variable set by the user in the VR visualiser.

6.3 Model interaction

The user could interact with the models in two ways; directly via the
controllers and indirectly through a menu placed in the scene.

Figure 6.11: The structure of the classes handling interaction. Gray nodes
represent custom code while white nodes represent functionality already
present in UE4. The orange arrows indicate the direction of the information
flow.

The Grabbable class

The Grabbable class was the super class of all actors that can be interacted
with via the controllers. The class handled itself while the Grabber
Component sent in action requests. By making the controllers send data
and make action requests to the model several separate controllers can
interact with the same model simultaneously. This enabled the scaling
action that uses both controllers. The Grabbable class also held a priority
value used to rank the instance’s position in the grab queue. The actor with
the highest value would be "grabbed" by the Grabbing Component.

Three Grabbable child classes were developed for this thesis: Geometric,
Volumetric and Hybrid. The geometric sub-class was used for geometric
models; The Rat Brain demo and the Genome demo. The volumetric
class handled volumetric data and was only used as a part of the hybrid.
The Hybrid used an instance of both the geometric and volumetric sub-
class in the Artery demo.
Each sub-class had separate custom code to prepare the models for rendering and separate collision boxes.

**The GrabberComponent class**

The Grabber Component was a custom class that could only interact with objects derived from the Grabbable class. The VRPawn’s controllers handled input from the user and when the trigger button was pressed it called the Grabber Component’s `GrabOverlappingActor()` function. This function asked the engine for all overlapping actors and filtered out all actors that were not children of the Grabbable class. The Grabbable instance with the highest priority value was selected and attached to the controller until the trigger button is released.

The Grabber Component made making actors able to interact with children of the Grabbable class easy. Any class that added the component would be able to interact with any instance of any child class of the Grabbable class.

**Scaling, rotating and translating models using the HTC Vive controllers**

Interacting with the model directly provided the testers with an intuitive method for scaling, rotating and translating the model. A dedicated button sent the command to the grabber component to "grab" the model. This grabbing action is set in quotation marks because the Grabber Component only found the Grabber actor, if any, with the highest priority and requested that it attached itself to the controller. Attaching an object makes the object dependant on the parent object; in this case the controller actor. If grabbed with both controllers the model would grow or shrink with the centre of the scaling located at the centre point between the two controllers. The new scale of the model was defined as $\text{NewScale} = \text{OldScale} \times \frac{\text{NewDist}}{\text{OldDist}}$. To centre the scaling around the controllers, the new model centre was set to $\vec{\text{NewCentrePos}} = \vec{\text{OldCentrePos}} \times \frac{\text{NewDist}}{\text{OldDist}}$. The vectors $\vec{\text{NewCentrePos}}$ and $\vec{\text{OldCentrePos}}$ were the vector offsets from the grabbing controller to the centre of the model being scaled.

The main benefit of this new approach to model interaction was the inversion of the controls. The models would transform instead of moving the camera. This new method of interaction was instantly picked up by most testers during testing and allowed them to inspect the models much more easily than before. Operations that would take several minutes and much confusion was reduced to seconds.

The challenge of grabbing the cutting plane when inside another model was solved by using the priority value of the Grabbable class. The Cutting Plane actor has the highest priority, which allowed for guaranteed interaction with the tool.
Variable adjustment through menus in VR

Another form of interaction presented in this thesis is the menu based interactions. A menu in VR is a 2D menu placed in the world space as a bounded 2D plane as seen in figure 6.12.

Interacting with a VR menu in UE4 required two components; a 3D widget and a widget interactor. Both are provided by UE4 and are made to interact with each other, but the design and the implementation of the functionality is left to the developer.

Using 3D widgets allowed for more complex interactions through menus similar to the kind used in standard solutions for 2D screens, but due to the inaccuracy of human wrist movement the size of each element had to be significantly larger than the 2D screen counterpart.

![Rat Brain](image)

Figure 6.12: This is a screenshot of the rat demo with the thalamus mesh disabled by using the menu. The model is also cut by the cutting plane.

Interaction with and control of the cutting plane tool

The cutting plane is a unique subclass of the grabber class described in section 6.3.1. The user interacted with the cutting plane by grabbing it with the trigger. The orientation and location of the plane was updated every frame and sent to the GPU as described in section 6.2. The right hand controller had a button that toggles the cutting plane and allowed testers to remove the tool from cluttering the scene.

Extracting mesh names as model part identities

The researchers requested the ability to extract the name of the chromosome or the brain section they wanted. By using the mesh name of the mesh last overlapped with the controller it was possible to display the name on a small text renderer attached to the left controller. To allow for accurate name extraction a small sphere was attached to the controllers to act as "mouse pointers" in VR.
Due to the complexity of the models the geometry itself had to be used for the intersection calculations. Complex collision calculations are much more expensive than simple collisions based on cubic, spherical or cylindrical collision volumes and might not be viable for more advanced visualisations.

**Measuring distance using both controllers**

The other controller request given was the ability to quickly measure distances in VR. The distance was calculated by the following formula:

\[
\text{Distance} = \frac{\text{InterControllerDistance}}{\text{ModelScale} \times \text{DataMagnification}}
\]  

(6.4)

By manually entering the magnification scale between the scientific data and the UE4 representation it was possible to accurately provide an accurate measurement much faster than previously available solutions.

### 6.4 Implementing efficient and comfortable user movement options

VR posed some unique challenges when designing a translation system for the camera. Since the HMD covers the entire field of view, the user lost the stable environment the eyes uses as reference. Normally one could see the physical screen, the table and the room so the brain has enough visual cues to not be confused by the moving image displayed on the screen. The lack of these reference points in VR caused nausea and dizziness in several testers.

During testing it was also found to be uncomfortable to use a continuous motion when translating the user. The inner ear signals the brain that the player is standing still, while the eyes tell it that the user is moving. This disconnect caused nausea and dizziness. It became clear quite early that two viable options were available:

1. Translating the model instead of the user
2. Teleportation of the user

**Eliminating discomfort by providing the user with stable reference points**

The proposed solution solves the former issue elegantly by having a static floor that matches the physical floor of the real world and various static elements in the scene. It was sufficient to have a semi transparent floor or an opaque pattern applied to the floor. These stable reference points replaced the lost reference points and proved sufficient to not disorient the user. No reports of nausea and dizziness were received unless the FPS dropped to 45 FPS. With 45 FPS nausea was reported after about 30 minutes of use.
Implementing a modular multi-user translation mechanic with minimal discomfort using teleportation and the concept of an "area of interest"

Using the thumb pad to translate the user continuously caused major discomfort for several testers and was replaced by instant teleportation. The instant translation avoided the discomforts effectively and allowed for much higher modularity when interacting with the scene.

Teleportation offered a series of advantages over the alternative. It allowed for each user to move independently of each other and they could each work with different models. By frugal use of input options, a single button could allow for all desired movements as it used the orientation of the controller to dictate the target location. To enable precise placement of the area of interest a preview was shown when the button was pressed. Upon release the user was moved to that location.

Figure 6.13: The teleportation arc at a steep (left) and shallow (right) angle. The arc can be tweaked to balance between distance and ease of control.

The use of an arc for teleportation had two unique benefits over using a straight line; the user could move to higher floors if there were multiple and the accuracy was fairly constant even at longer ranges. Using a straight line caused the users to often miss their target when the angles between the laser and the floor became small. The final benefit was one of speed. If the user wanted to move quickly the user only had to click the teleport button rapidly. This made inspecting several separate models or several separate areas of a single model trivial and effective.

The only downside to the teleportation solution was the lack of accurate vertical movement. You were locked to the floor so if your area of interest was not at that height the user had to drag the model into place using the controllers. This deficiency took some time for the testers to effectively work around.
Moving the area of interest by teleportation did not limit the accuracy of the camera

Since interaction with the model was done standing and the user moved his head and body to translate and rotate the camera relative to the model, a continuous movement mechanic was not required. The user selected one section of the model he had an interest in and placed the area of interest at that location. Any minor adjustment could be done by moving the model using the controllers. The precision of movement of the human head allowed for extremely precise inspections of the models. It was found to be adequate for the test cases presented in this thesis.

Using a NavMesh and PredictProjectilePath to construct the teleporter preview

The implementation was based on the system used by the VR template within UE4. The same meshes were used. As figure 6.13 shows, the teleportation destination was indicated by an arc, a target circle and a rectangle representing the area of interest. Calculating this arc was made trivial by the UE4 blueprint node PredictProjectilePath and a NavMesh.

The PredictProjectilePath node takes a direction, a start location and a velocity and calculates the arc of a projectile if fired. Once a collision is detected the the path stops and a series of locations are returned. By using these locations to combine static meshes the appearance of an arc could be created.

To provide more control over where the user could move a NavMesh was used to limit the teleportation. A NavMesh is a volume used by the engine to calculate what areas are navigable by AI. The NavMesh enabled the use of a function called ProjectPointtoNavigation(FVector position), which tries to find the closest point on the NavMesh from the provided vector. If the projection fails a zero vector is returned and was used to determine if the location was valid.

If determined to be valid the arc meshes were made visible.

Continuous global translation using a thumb pad or thumb stick

Translating the models by using the thumb pad or thumb stick worked well. It allowed the user to translate in all six directions by using the HMD orientation or the controller orientation to determine the thumb pad plane as seen illustrated in figure 6.14.

By reversing the vectors and applying it to the models in the scene the user appeared to translate along with the floor and other static objects in the scene. This proved to be effective at avoiding nausea and dizziness as the floor provided a stable visual reference frame that matched the signals from the inner ear.

There were two weaknesses with this approach. Such a system could only support a single user since each user would only translate the models. This made it impossible to design a solution that allowed the users to move
Figure 6.14: The plane determined by the orientation of the camera. The blue arrow was the rotational axis of the thumb pad, represented as the blue circle.

relative to each other. For single user software it might still be undesirable since the translation requires the entire touch pad. The limited number of available buttons on concurrent controllers already limited the interactions directly available to the user. By reserving the four thumb pad directions for movement the power of the control schema was further diminished. This became a problem as features were added.

6.5 Implementing a post process material to draw multi coloured outlines

UE4 provides access to two additional frame buffers: a custom depth buffer and a stencil buffer. The custom depth buffer works identically to the standard depth buffer. It is possible to draw the depth of selected geometric meshes to this buffer. Each object could also decide to draw to the stencil buffer with a custom 8 bit value, seen in figure 6.15.

Figure 6.15: Left: The outline as viewed in the visualiser. Centre: The custom depth buffer. Right: The stencil buffer with stencil values.

To ensure a match between the mesh colour and the outline colour, both were determined by using the mesh index to calculate the hue in the HSV
colour model. The stencil value was set to

\[ \text{StencilValue} = \left( \frac{\text{MeshIndex}}{\text{NumMeshInModel}} \times 2^8 \times 5 \right) \mod 2^8 \]  \hspace{1cm} (6.5) \]

Equation 6.5 ensured that each mesh had a unique stencil value for up to \(2^8\) meshes per model. This translates to every mesh having a unique colour. The magic scalar, the 5, caused the colours to have satisfactory contrast. Without that term the colours of neighbouring meshes were too similar to be distinguishable.

Drawing the outline was achieved by using a modified Sobel-Feldman operator to extract the edges between stencils. For each pixel of the edge the largest stencil value was selected.

\[
\begin{align*}
H & = \frac{\max(\text{kernel})}{2^8} \\
S & = 0.9 \\
V & = 0.3
\end{align*}
\]  \hspace{1cm} (6.6)

Equation 6.6 shows how a colour was determined by dividing the extracted 8 bit stencil value and dividing it by \(2^8\) to get the HUE value, which had to be between 0 and 1 to be compatible with the custom converter, algorithm 6.2. In the HSV colour model the HUE determines the colour. After converting the HSV colour to RGB it was drawn on top of the completed render.

```c
float3 HSVIn;

//Required magic numbers
float3 a = float3(1, 1, 1);
float3 b = float3(1, 0.67, 0.33);
float3 c = float3(3, 3, 3);
float3 H = saturate(HSVIn.x);
float3 p = abs(frac(b+H)*6 - c);
return lerp(a, clamp(p - a, 0, 1), HSVIn.y) * HSVIn.z;
```

Algorithm 6.2: The HLSL version of the custom UE4 HSV to RGB converter. Based on the algorithm provided by Sam (2017).

For this thesis a \(3 \times 3\) kernel was used and the highest stencil value was given precedence to be drawn on the screen. To create a thicker outline a larger kernel is required.

### 6.6 Bottlenecks found during this thesis

**GPU bottlenecks**

Most meshes generated from volumes have a very high poly count and as such must be reduced. The poly count is the first of two aspects of a
mesh that can greatly reduce framerate. For a NVidia GTX 1080 and a Intel i7-4770 it was found that four million triangles was a good benchmark to aim for when reducing models. Anything above four million slowly started to reduce the framerate under the 90 FPS requirement. The shader complexity can also be a limiting factor that might force the acceptable poly count to be reduced further, but complex shaders were not required for any of the demonstrative models used in this thesis with the exception of the volume renderer. Shader complexity is the complexity of the shader code and the pixel count and complexity of the effects applied to a mesh act like multipliers for the cost of the render. The higher the complexity, the fewer triangles you can render. When generating the meshes it can be important to cut away any expensive effects or to reduce the accuracy of the effect to make the shader cheaper. The only geometric shader bottleneck found during this thesis was shadow casting. Due to the complexity of the models shadow casting costed over half the time budget(6ms) and were not required so it got removed. Shading was kept to convey the three dimensional shape of the models. The other, often forgotten, bottleneck is the draw call count. A draw call is a function call that the host, the CPU, sends to the device, the GPU. A draw call is quite slow because transferring the data from the RAM to the graphical memory is very slow. Geometric models are much more compatible with UE4’s rendering pipeline, allowing for high quality even within VR. The amount of detail was not found lacking as four million triangles can give you very complex models. To achieve this high quality the scene is rendered in batches. Associated with every batch is a draw call and the mesh data for every object to be rendered during that batch is sent to the GPU. The higher the number of draw calls, the more data is sent and the more time is spent on the buss between the RAM, the CPU and the GPU. Each separate mesh and each separate layer in the deferred renderer can result in a new draw call. The fewer meshes you have the fewer draw calls you will also have. There are techniques for bundling meshes together for a single draw call, but the UE4’s methods can not handle extreme numbers of meshes. The original chromosome model had 27 709 meshes, effectively slowing the FPS down to under 45 FPS. After combining the parts of the chromosome, the TADs, to a single mesh for that chromosome the FPS jumped to a stable 90 FPS.

CPU bottlenecks

The GPU is not the only place where a bottle neck can occur. As described in 3.2, there is also the preparation of the models that are rendered. Occlusion culling is a group of techniques that are used to deduce ahead of time what is visible on the screen. What is not visible is removed from the list of objects that is about to be rendered. The complexity and cost increases exponentially with the number of meshes to sort so if possible the separate meshes should be combined into as few meshes as possible. You lose the ability to interact with the smaller meshes, but you reduce the cost of rendering your models by spending less time sorting and filtering the models. This happens on the CPU and must be done before either of the
GPU steps can be started. An alternative to combining the meshes would be to remove the expensive calculation by disabling occlusion culling, but UE4 did not support disabling it directly.

To avoid rendering what is not visible a series of culling types are used to speed up the renders:

- View frustum culling This was explained in detail when describing the render pipeline in 3.2.1.

- Contribution culling Once an object occupies a sufficiently small area of the screen the impact on the final image can be negligible. As such it is often removed from the render queue.

- Occlusion culling Determines what is hidden by other geometry.

- Backface culling In geometric models the notion of a front side and backside can be thought of as the outside and inside of the model. Since nobody is expected to pass inside a geometric model the inside can be safely discarded and only render the outside of the meshes. This is the culling that was used to achieve the cutting effect (section 6.2) and the ray tracer (figure 6.6).

Normally culling decreases the render time, but occlusion culling had an negative impact on the rendering speed since the CPU could not prepare the data fast enough. Unlike games which has fewer and larger meshes, a cloud of small independent meshes as can be found in some datasets will cause a lot of computation time to be spent on occlusion culling. The effect can be felt both on the CPU and on the GPU. During initial concept testing a ordered structure of $16 \times 16 \times 16$ tetrahedrons caused an entire 6 ms per frame to be used on occlusion culling alone. That was 60% of the time budget for a frame. When all 4096 tetrahedrons were combined into a single mesh to avoid the occlusion culling the render ran at 90 FPS again. Doing so freed up time that could be spent on more complex rendering effects or more complex models. The only downside to combining meshes is that the meshes can not change after being combined with a few exceptions.

**Optimisations for VR**

To achieve the 90 FPS target a series of optimisations have been developed by VR software and hardware developers. Unless otherwise specified the following optimisations only apply to geometric data.

Weier et al. (2016), Pohl et al. (2015), Vlachos (2017), Bunn (2017) and Weier et al. (2016) reduced the total number of pixels rendered by rendering the periphery at a reduced resolution or by removing unseen sections of the screen from reaching the fragment shader. This can be done because the barrel distortion of the lens, parts of the screens will never reach the users eyes and the curve of the lens stretch the pixels as a function of the distance to the centre of the screen. This stretching leads to a blurring effect and by reducing the number of pixels to be rendered in the outer area a significant increase of FPS can be achieved while not losing much detail.
At the Game Developer Conference (GDC) 2015 Vlachos (2017) summarised the results of their research into geometric optimisation. To remove the unseen pixels from the rendering queue they use a “Hidden area mesh” that is put in front of the camera to cover the unseen areas. Since the lighting is constant between the two frames it can be calculated only once for single GPU computer systems. For multi GPU systems the transfer cost of the shadow data and the idle time on the extra GPUs made having each GPU render the shadows for their own use the easier alternative. For volumetric models shadows can not be shared unless baked into the volume data. Two GPUs provide an potential resolution increase of $2 \times$ while four GPUs provided an potential resolution increase of $3 - 4 \times$. To avoid reprojection SteamVR includes an adaptive quality algorithm that automatically changes the quality of the render based on previous frame times and a linear heuristic. Anti-aliasing and resolutions are safe to change, but shadows, shading and specularity are not since they will result in obvious flickering during renders.

During the Steam Developer Days 2016 Bunn (2017), a representative for Epic Games, summarised a series of optimisations benefiting VR and effects to avoid. All these optimisations do not apply to volumetric rendering algorithms. By using VR Instanced Stereo Rendering the CPU avoids duplicate draw calls and buffer compilation by reusing the same geometry for both eyes. Additionally, since the same geometry is used for both eyes the total overhead is reduced, thus reducing GPU usage. The post process effects Lens Flare, Screen Space Reflection, Temporal AA, Screen Space Ambient Occlusion and Bloom were recommended against. If shadows are required dynamic lights should be avoided. Use static lights and baked lights where possible since dynamic lights has an multiplicative cost in VR. Finally, use meshes for visual effects (VFX) and animations. Since billboard effects are 2D sprites aligned with the camera they are not affected by most optimisations currently used.
Chapter 7

Analysis of VR technologies for scientific data

The following observations and analyses are based on the personal experience of the author, feedback from testers, and trends found within the VR world as it was when this thesis was written. Many of the topics discussed and presented in this chapter could each benefit from a more rigorous study. This chapter is meant to analyse the feedback and initial reactions to the VR visualiser and to re-analyse the technical challenges and possibilities unique to VR as a visualiser tool through a modern lens.

7.1 VR can be useful for research, education and industry if used right

Education

Given the looser demand and more static nature of educational software, education is primed to benefit greatly from this technology. Simplifying concepts and illustrations in order to streamline a student's road to comprehension is a commonly used tactic in education and lend themselves naturally to custom solutions that are optimised for VR. We have many times seen that the productivity of students and interest in, and thus understanding of, a topic increases when presented with the knowledge in an interactive and interesting way as shown by Roussou (2004), Trindade, Fiolhais and Almeida (2002) and Blasco-Arcas et al. (2013). Holding an animal cell where all the components are marked and have descriptions would be one example of VR providing interactive and almost tangible models for self-driven and natural learning. They can dismantle it as a toy and inspect each part before attempting to reassemble it again. They can be asked to provide a named component. A puzzle game where the student has to find clues and specific objects from the curriculum and construct a complete cell matching a description is an example of how VR can be used. This technology engages the entire body and two of the users strongest senses.
It turns out that interest is far more significant than readability. When students have strong interest in what they read, they can frequently transcend their reading level.

Student Attitudes Toward Reading: A Case Study, (Seitz, 2010, p. 31)

The principal issue with poor performance amongst students is not the complexity or difficulty of the subject, but rather the lack of interest caused by a lack of stimuli. Older methods like clicking on a screen and flipping through a book simply are unable to interest and engage to the same extent as the alternatives and we see this effect every day in modern students. More and more find it hard to stay engaged with the classes and often resort to daydreaming or talking or playing with their phone or tablet. In this ever more stimulating daily life, education is falling behind. VR can offer a way for the curriculum and class work to catch up to the alternatives that might have distracted them.

Once engaged, students can learn much more in a shorter time period with less effort if presented with complex ideas and models. According to the professors from the University of Oslo guiding the development of the demos presented in this thesis, VR should be able to make complex and difficult to understand models of advanced brains and sub-cell structures easier to understand. The only downsides they saw was the price tag and the space requirements. They were positive to a solution where the student could utilise their own phones while a select few used the HTC Vive or the Oculus Rift.

Industry

Figure 7.1: The top two images show the same car model one second apart. The bottom three images show three different configurations of the same interior. Images taken from the Epic Games stream (McLaren and Epic Games, 2017)
For the corporate market where engagement and entertainment are not the main foci many similar benefits can be imagined. Many aspects of handling the data become more natural, as described above. Designers can gather around virtual models and discuss, instead of making expensive physical models for the same purpose. The virtual models can be animated in realtime, thus providing the ability to iterate slight design variations quickly. Further, as it is a computer system simulations can affect the data as the user are doing everything above. An example can be seen above in figure 7.1 where the McLaren car is rendered in UE4 and controlled via a separate device. It allows you to turn around the car, change the paint, enter and exit the car to modify the interior. It allows for explosion of the car to inspect different layers of the car design. The McLaren demo was created for marketing purposes more than scientific visualisation, but it illustrates the potential of using VR for inspecting a simulated product. Wind tunnel simulations and road bump simulations are common simulations a concept car is tested in before creation of prototypes.

Research

It seems likely that research will eventually benefit the most from visualising data in VR due to the complex and challenging data used in the forefront of science. For example, in traditional flow field visualisations a lot of the challenge lies in properly removing unwanted data and encoding the remaining interesting data. Encoding depth or removing obfuscating parts of the visualisation that hinders the user from seeing the region of interest are two common challenges of flow field visualisation algorithm design (McLoughlin et al., 2010; Max, Crawfis and Grant, 1994). Encoding depth often effects the appearance of the visualisation and uses one method of conveying information. If the depth can be seen through stereoscopic means the visualisation is open to be tweaked for increased cleanliness or increased information density. The other problem, that of obfuscation, can also be diminished by the users ability to move into the volume and to perceive depth. Visualisations of fields tend to show a global picture where the perspective comes from the outside. By using a VR solution the camera can travel inside the models giving the user an increased awareness that is normally lost when entering a complex structure. The final main benefit is the speed users will be able to move the camera. With standard solutions the camera is translated and rotated iteratively until a satisfactory transform has been achieved. With a VR system the operation is reduced to a single continuous movement of the neck. The model that best visualised this specific advantage was the Genome demo presented in this thesis.

7.2 VR helps learning and understanding across familiarity levels.

The researchers who provided the genome and rat brain models used in this thesis described VR as a new unparalleled method of viewing said
models. It also aided their immediate understanding of what they saw and the patterns became more obvious and easier to discern.

Every tester claimed that they would benefit from using VR in some respect. The testers of the genome demo reported that the internal structure was much easier to understand and that the relative scales, orientations and locations of each component were more apparent in VR. Tasks researchers had to rely on slow calculations for could be replaced by simple inspection. This would help with prototyping and iterating hypotheses during research. Researchers that are not familiar enough with the tool they are using to write the required scripts themselves can, through VR, get an initial estimate of the viability of their ideas. This would save time for both the researchers and the developers they have to request the scripts from. Here the same benefits found with using proper visual aids to lecture students seem to also be benefiting researchers at the forefront of their fields. Given that this prediction holds true a massive boost to the speed at which research advances could be witnessed after a widespread adoption of VR.

The rat brain demo feedback highlighted only two benefits: 1. Their understanding of the relative scales and location of each brain section was heightened, and 2. The brain was much easier to inspect in VR due to the VR camera’s ease of use. Benefit 1 is the same benefit testers of the genome demo reported. Benefit 2 was more prevalent in the feedback given by the testers of the rat brain demo. In IKT-SNAP the users controlled the camera on the three planes XY, YZ, ZX while a 3D render could be viewed in the fourth sub-window. The users could also rotate the camera around the centre point selected through the three planes of control and move the camera linearly to and away from said centre point. These control options made operations like inspecting areas inside the brain and between sections very difficult. Often the tester would get disoriented and struggle to move the centre point properly and when successful the process would be arduous and slow. VR provided an alternative method to transform the camera as they wanted. Since the camera is controlled by the head it utilises the skill people have practised their entire lives to master and as such is extremely fast and intuitive. It can arguably be less precise, but none of the testers raised any issues regarding it.

In both the genome demo and the rat brain demo VR provided the testers with tangible and immediate benefits that could expedite any future research. Most negative feedback fell into four groups: 1. Weaknesses and missing features specific to the implementation and demos presented in this thesis, 2. Issues with the limited interaction options, 3. Visual quality and HMD weight and 4. The cost of a usable system hinders most immediate adaptations.

Issue 1 will not be discussed as future extensions can add higher quality datasets and features like an importer and exporter. Issue 2 will be discussed in 7.4. Issue 3 will naturally disappear as HMD screens become lighter and have higher resolutions. Issue 4 will briefly be addressed in 7.7. This issue should also be diminishing as commercial VR systems become more affordable with time.
7.3 Observations made during the implementation and testing of this thesis

During implementation and testing a series of differences between classic tool kit design and VR tool kit design came to light. VR requires special design adaptations and offer completely new approaches to data visualisation.

Designing a VR tool require several aspects of the real world to be represented in VR

The design needs to be familiar to the user. For example, in a normal visualiser or 3D software one rotates the camera around objects that hang in an otherwise empty void. This will not work in VR as we replace the real world with the virtual one. As such some rules that must be obeyed in the real world must be present in the virtual one. Examples are, loosely ordered by importance:

1. A solid floor or other stable entity from which the user can form a frame of reference
2. A stable environment where the camera is not moved unless the user initiates it
3. Ability to directly interact with objects via the users "hands"
4. A sense of gravity to give the user a sense of direction that matches the real world

For all testers only #1 was indispensable as without it people reported fear of heights, nausea, disorientation and dizziness. One tester almost fell when attempting to walk. Failure to provide the necessary visual cues or providing misleading visual cues can lead to amplifying the dizziness and nausea. #2 relates to the design of the translation mechanic used in VR. The two options were continuous movement or instant teleportation. Continuous movement has several inherent problems that prevent it from being the preferred translation method. Movement was discussed in more detail in 6.4. #3 was the feature of VR that the testers reported as "the most surprising improvement VR provides". Handling objects using hands is something humans are very adept at and familiar with. The skills related to the use of hands naturally translates into VR when allowed to handle the models using the controllers. Being able to grab and scale a model using hands is not a requirement, but it made such common interactions fast, intuitive and fun. The amount of "fun" the testers reported might be influenced by the novelty of VR however. A closer discussion of performing the basic transforms using the controllers will be given in 7.4.1

The last point, #4, was added because an early bug in the code caused some objects to "fall" upwards inside the VR environment. One tester found this disconcerting for some time.
The power of VR is the most apparent with complex systems.

The main advantage of using VR is the superior sense of depth it provides. The power of VR as a tool for visualisation grows as the complexity of the model and the importance of depth grows. The genome demo received the most positive feedback because understanding of the functions and interactions of the TADs relied on an understanding of the structures of the overlapping and interleaving chromosomes. Comparatively, the artery demo with its simple model without any overlapping layers represent a very low utilisation of the advantage of VR and this was reflected in the feedback.

It is important to note that only having depth is not what makes a model suitable for VR. There are several structures where the depth can vary and be complex. A model of New York would have towers of many sizes and shapes and heights, but viewing it in VR should add little insight. Comparatively the caudate nucleus is a small and relatively simple structure in the brain, but it would be much easier to understand the structure, size and location of the caudate nucleus in relation to the brain as a whole and by itself. We named this specific complexity interleaved complexity as the more overlapping and interleaving a model is, the more additional insight can be provided by VR.

Linking the camera to the head of the user provides unparalleled control and speed

Having a complex and dense structure like the Rat Brain used in this thesis makes viewing the small nooks and crannies of the model very hard. Following long structures requires many separate translations and rotations using common techniques. By linking the camera to the head, similar to how eyes are attached to the head, enables the user to easily and quickly look exactly where the point of interest is. Further, following a complex structure as it twists and turns in VR is much easier than in standard software solutions. If you imagine how a human is capable of following a cable in a mess of cables, the same ability can be used to track a complex structure inside other complex structures in VR. Finally, since the user naturally feels the scale and direction of any motion it is possible to maintain an estimate of the camera’s location even when blindfolded. This makes getting lost much harder and any subsequent attempts only take seconds to perform.

By using the natural ability of the human brain to understand 3D structures, spatially complex models can be made much easier to understand.

In addition to the natural ability of the user to very accurately and controllably move their head as discussed above, the other natural ability the human brain has evolved is the ability to understand 3D structures. The amount of information the users reported receiving from inspecting...
the models per time interval far outclassed the alternative tools the testers had experience with. The only exception was the artery demo where no perceived increase was reported.

The discovery that best exemplifies new design avenues opened by using the above mentioned natural ability was discovering the efficiency of conveying the 3D shape of hidden meshes to the user through an outline as seen in figure 6.8. By using a coloured outline the user could deduce the 3D shape thanks to the VR HMD’s stereoscopic vision. This allowed the user to get a detailed understanding of the structure, position, scale and orientation of parts completely obstructed by the rest of the model. Testers reported an increased understanding of the structure of the genome as a whole and insisted that “this can aid in discovering new relations and creating more accurate hypotheses.”

**Requiring 90 FPS to be kept without reprojection severely limits the viability of current algorithms.**

In addition to the restrictions the lack of input options apply to the design of a VR tool, the 90 FPS realtime render requirement also make many algorithms too slow to be viable. Several industrial parties leading the VR development have concluded that 90 FPS is a requirement for comfortable and extended use (Bunn, 2017; Vlachos, 2017). Testers have also reported sickness after about 30 minutes of use when the frame rate fell to 45 FPS. Reprojection, that is altering the previous frame to give a new semi-frame, can also not replace the 90 FPS requirement (Bunn, 2017; Vlachos, 2017). Many ray marchers and ray tracers take minutes for a single frame and even the algorithms that call themselves “real time algorithms” can often not produce more than 30 FPS for a monocular system. Compared to monocular systems commonly used in existing tools, VR is much heavier on the GPU than the CPU because it requires a multipass rendering to combat the natural barrel distortion and refraction of different wavelengths of light. Further, unlike a 2D screen you have two camera you need to render to. Finally, demands related to latency and FPS require that each frame is rendered and sent to the screen within 10 milliseconds in order to ensure 90 FPS while allowing the OS and other background tasks to run without dropping frames. Reprojection is not a viable alternative. For contrast, normal renderers using traditional 2D interactive interfaces only have to do a single pass once within 50 milliseconds to allow for acceptable interactivity. Adding it all together we can see that a VR system require:

- At least one pass for each colour; Red, Green and Blue
- one render for each eye
- 90 frames a second, or one frame every 10 ms

Compare that to a standard system:

- At least one pass for all colours
• One render for the camera
• 10 ∼ 20 frames a second, or one frame every 100 ∼ 50 ms

This leads to VR requiring up to 3 \times 2 \times 9 = 54 times the GPU power of the traditional monocular renders. This further increases the challenge of designing and implementing a scientific visualiser in VR.

The algorithm presented by Parker et al. (2005) only produced 15 FPS at 512 \times 512 image resolution while volume was 1734 slices of 512 \times 512, Sherbondy, Houston and Napel (2003) rendered a 128³ volume at 15-25 fps at “useful screen resolutions” (Sherbondy, Houston and Napel, 2003, p.174). For geometric renders Weier et al. (2016) managed to render geometric renders at 160 FPS at a resolution of 1182 \times 1464 per eye using ray tracing with a GTX Titan X. Foveated rendering with a small focal area was used to achieve the 160 reported FPS. Using similar optimisations while letting the accuracy drop slightly might be enough to have large volumes rendered efficiently at 90 FPS. Since the human brain and eyes are already inaccurate a small drop in accuracy for visualisation purposes might be undetectable. To get the best of both worlds the cheaper, but less accurate algorithm can be used when in VR while exact calculations can use the slower and more accurate algorithm. Further, if the volume can be processed to make it VR ready by hard encoding and optimising the model, then the accurate volume can be used with the standard monocular renderer and a cheaper and less flexible version of the volume can be used in VR. The workflow would then consist of getting a result that looks interesting and then “export” it to VR by clicking a dedicated button that processes the volume and launches a VR visualiser with the simplified model.

Initial attempts at implementing a ray marcher for this thesis was only able to render the volume just over twenty times per second. After pre-encoding the data into coloured textures and extracting the setup code from the ray marcher, section 6.1, the demo was stable at 90 FPS. Since a VR system in UE4 either runs at 90 FPS or at 45 FPS, exact frame rates can only be accessed by using the render times in milliseconds.

Even in the worst case scenario with un-packaged assets 1.61ms was left of the time budget. The WorldTick operations is related to the editor and can be ignored. The remainder could be used for data encoding and indicates that having the data encoded on the fly instead of using pre-encoded textures might be possible. Further time savings might be found when a dedicated and optimised software is created since the demos developed for this thesis were slightly slowed down by being run in the UE4 editor.

### 7.4 Input systems

Non-VR systems can utilise the entire keyboard and the mouse to generate unique input events. This provides the user with a high density of information and interaction options through compact menus and keyboard combinations. Using multiple screens further extends the amount of
concurrently available information and input options. Current VR options only have two controllers with a few input options each; usually 8 or less.

7.4.1 Currently available input systems severely limit the design of a VR tool

Most scientific tools are complex and powerful due to their customisability. Especially generic tools attempting to provide usable renders of a varied set of data types. To achieve this they require a series of options and menus to expose all their power to the user. With VR input options as they were during the writing of this thesis, few direct input commands can be made available to the user. To best utilize the input options the design of the tool must be streamlined and limited.

One way to partially avoid the issue could be to have controller schemas for different interaction types and use a menu to select between them. Since each menu must be placed in the world space and thus occupy potentially valuable space, designing to minimize the number of menus might be required to create a usable tool. Attaching the menus to the users arm could be used to add input options on top of the physical options the controllers offers1.

The best solution might come from Microsoft’s work on voice recognition. Microsoft has claimed that their AI has reached “human parity” (Xiong et al., 2016), but exactly how reliable and useful this is will require some further research. The Hololens uses voice control and has a built in voice analyser exposed to the application level which worked well for simple commands. A single word usually worked fine as long as the person had a proper pronunciation, but accents or phrases tended to cause the analyser

---
1An example can be found at: https://cdn.virtualrealityreporter.com/wp-content/uploads/2015/09/tilt-brush-nvidia-1.jpg
to fail. If a sufficiently robust and accurate voice recognition software were to be incorporated into the tool, the user could order the tool to spawn the menu or sub menu the user required. This would enable each interaction to be simple enough for the controllers to be accurate and complex enough to offer complete control.

However, moving, scaling and rotating the models are much quicker and intuitive.

Unlike traditional tools the VR controllers offered many benefits that made certain types of interaction much faster and intuitive. Several testers reported that the basic three interactions; Translation, Rotation and scaling, were much easier to use and much faster when performed in the VR visualiser.

Although the final transform was not as accurate when transformed by hand compared to using the traditional methods of number input and dial sliders, the speed of the interaction was far superior. Testers also reported that it was not required to be extremely precise when inspecting a model and that using their hands to interact with the model was accurate enough for their uses. The only negative feedback came from trying to inspect a model while holding it. Due to the small shakes and movements of the user hands the entire model moved and caused confusion. The user had to let go of the model and let it hang in the air to inspect it properly. This might cause an issue if a force like gravity is affecting the model.

Using the third dimension for gesture control.

Since VR works in a three dimensional space the controllers have another dimension to use for input. Gestures can be used to separate two different commands using a single button. For example, you can click and drag away from you to perform one action and click and drag left to right to perform another. Many of the design philosophies of mobile development can be used in VR as both have very few input options and too small a screen to allow a lot of menus. One example implemented in this thesis is the replacement of the scaling button with a grab and stretch gesture. Instead of clicking on a certain button and then dragging along the screen to scale the model, the user can just grab the model and separate the controllers to achieve the same result. Several interactions like translation and rotation can be performed simultaneously on up to two separate models with a single button click and a gesture. There are pros and cons with the currently available VR input systems, but the input options are noticeably more restricted and must be designed around.

Input systems in development

Some projects are currently being developed that aim to eliminate the lack of input options. The Magic Leap is trying to scan the users hands to remove the need for controllers. Having robust tracking is still some ways
off, but as technologies like these are developed it will become possible
to create virtual keyboards and use them for complex input while inside
the virtual world. Current text input options use the controllers to aim a
laser pointer at the letter and pressing the trigger or using the touchpads.
This is acceptable for occasionally entering numbers, but any extended
input or repeated input will soon prove to be many times slower than
performing the same actions using a mouse and keyboard. Users are also
prone to rotating the controllers slightly when they press the trigger and
thus moving the laser off the intended target. Further, since the trigger is
what controls the button’s "pressing" action it is very easy to not release it
quick enough or to press it too quickly when typing.

7.4.2 UX design must be reinvented to avoid clutter and obstruction

User Experience(UX) design has been quite standardised for PC. Menus
and hot-keys, mouse and keyboard events and command line input are
standard elements of UX design. Place a series of 2D menus on the screen,
use drop down boxes and key combinations to give the user access to a
wide variety of powerful tools. When performing a certain type of task
provide only related options in a custom fit UX system.

In VR everything that is not tied to the controllers must be placed in the
world with the models. This leads to two inherent problems with blindly
copying old UX design to VR. The first problem relates to cluttering the
scene. Menus can be turned away from you, obscure other menus or the
model you are interacting with and become scaled to a size that makes
them unusable just by allowing the user to move freely. Even if we do not
consider the low resolution of current VR HMDs, the menu will become
illegible at sufficiently long and short ranges. The other issue is accuracy.
When using a mouse the table provides the support needed for accurate
clicking. VR controllers does not have that so they tend to move slightly
while the selection button is pressed. This makes selecting the wrong
option or letter a common issue. This forces buttons and sliders to be scaled
up to allow for reliant use in VR.

To reduce these problems the menus were kept to as few as possible and
seldom used menus were hidden until requested by the user. This worked
well in the simple VR demos tested in this thesis.

7.5 Game Engines in scientific visualisation

Using game engines for educational tools could be viable as game
engines offer very efficient methods of creating the interactions and events
required to construct an interactive learning tool. Creating challenges
to be completed is a common part of game development and can easily
be adapted to create interactive tasks students can solve by themselves
without the teacher having to confirm the correctness of the result. The
issue that makes game engines less viable for research tools all relate to
two limitations inherent to such engines: 1. They are optimised for speed and not accuracy, and 2. They are designed for geometric visualisation only. The former problem was illustrated during initial exploration of UE4’s capabilities as a scientific visualisation tool. By creating a Vector Field in UE4 and a Particle Emitter to spawn particles that followed that Vector Field it became clear that accuracy was not what the algorithms used were designed for. If it was accurate enough it would have been great for visualising flow fields using streak, stream and path lines, but the resulting streak lines were pulsating and highly inaccurate. The difference between the extremes of the pulsating lines can be likened to a comparison between Euler’s method and Runge-Kutta integration when following a particle’s path through a vector field.

The second issue is more engrained into the entire design. There are many components of the engine that any game would use and as such can not be disabled. The best example might be Occlusion culling. It is expensive to perform, but saves more CPU and GPU time to result in a net cheaper render. For games accurate enough is adequate because most parts of a game only needs to appear real enough to be dependable for game mechanics. For example, the flow field example above is one feature that is only required to be accurate enough since it is a visual-only effect made to add short lived sparks or smoke puffs. One option to combat this deficiency would be to extend the engine with an accurate algorithm to create the flow field, but it seems advisable to instead implement VR functionality into a dedicated tool instead. If only a few algorithms are required to be accurate or the major workload is creating the interactive tasks the user must complete, then choosing a game engine might be the best option.

According to another developer working with the researchers who supplied me with the rat brain data, using Unity to visualise data should be much easier since Unity aims to be a more general engine than UE4. Unity had plug-ins for ray tracing available in the Unity marketplace. These plug-ins does not have the limitations described in this thesis and can be used with custom data. This design difference might make Unity a better option for future interactive game-like visualisers.

For now the different deficiencies and strengths of game engines make them unsuited for scientific visualisation. Instead, custom demonstrations and custom educational “games” using only simpler models would benefit more from being implemented in a game engine. Such experiences do not require the same precision and reliability as research, and each experience can remain as static examples and still be useful.

7.5.1 UE4

The Unreal Engine 4 tools and framework are designed to allow for rapid development of game like content. The available functions are there because they are useful for game development. The engine is built as a semi-generic rendering engine that allows for easy extension of certain limited areas. For games you would not need anything else, but the limitations and optimisations of the engine sacrifices certain key functions
that is vital for scientific use cases. It was originally assumed that extending the engine with more accurate algorithms using a third party library would be straightforward, but an OpenGL context could neither be created nor accessed on the same process without changing the render pipeline. It turned out that much of the render pipeline had to be adjusted and effectively pushed the workload past what would be possible in the time allotted for this thesis.

That being said, what UE4 was able to provide were:

- Generic and automatic handling of major VR HMDs; Oculus Rift, Gear VR, HTC Vive and Playstation VR
- Hit detection and overlap detection used for interactivity
- A simple scripting language to add functionality and features
- A customizable and powerful rendering engine for geometric data
- Miscellaneous classes and functions that made interactions easy to develop

Making the teleportation mechanic and the menus and the entire interaction system was very fast, easy and efficient.

**UE4 is not viable as a general scientific visualization tool**

In spite of its many strengths, UE4 is not viable for a general scientific visualiser engine. For research it is too restricted and implementing VR support into an already existing toolkit custom made for visualising their data is the sounder proposition. Due to UE4’s powerful scripting language and its focus on making the creation of functionality and interaction options, it would be perfect for creating smaller examples and interactive models for education. Since it has support for all major VR HMDs it can be used to make a myriad of highly custom and processed demonstrative models similar to those presented in this thesis. Further, UE4 provides a easy to use and reliable networking back-end that makes creating multi user experiences much simpler and easier. An entire class could join the same virtual world and VR homework could be done alone and in a group.

Custom animations, small demonstrations and VR based homework tend to be tailor made to fit the curriculum of the class in which it is used and UE4 offers a fast and effective tool kit for creating such experiences. For general research where accuracy and flexibility in visualisation is important a dedicated engine extended with VR support should be preferable.

**Using third party libraries with UE4**

UE4 supports the linking of as many third party libraries as is needed through a custom builder tool it uses between the C++ base and the engine itself. Without any engine changes the only limitation is that the libraries can not utilise the GPU. Sockets are available for reading networked data,
reading and writing to files and communication with other processes. For example, processing the data using VTK worked.

7.6 Non software related challenges

In addition to algorithm design and implementation challenges, current VR HMDs also have several inherent traits leading to problems that software design and implementation can not alleviate. As VR equipment improve we can expect many of these problems to become solved.

Cumbersome gear causes fatigue and discomfort.

However small, the controls and the headset, combined with the loss of some senses, will be a minor barrier that separates the model from the observer. The additional weight added to the front of the head causes users to be tired after extended use and due to its size it can get in the way of other tasks. Some users with large hair styles might find the HMD hard to use. Further, a few testers claimed their glasses were uncomfortably pressing against their skin due to the pressure of the HMD. Hygienic issues have also been reported and discussed. Strickland (2017) talks about perspiration causing irritation and other issues. More general hygienic concerns has also lead to the development of products aiming at simplifying the process of keeping the HMDs clean. Discussions on hygiene, especially on shared HMDs, are also being held (Wordpress, 2017).

VR requires a lot of space and can cause users to damage or destroy objects or living creatures in the vicinity

Most VR systems requires some practice to use since the boundaries of the real world can not be seen. Having a cable limits the distance you can move away from the computer and any physical object in the real world will become an invisible obstacle to the user.

Current solutions to avoiding the obstacles all revolve around clearing a space on the floor that is empty and marking that space by lines inside VR. The lines give the user a sense of where in the usable space is and the user can then keep themselves inside the safe area. This demands a large room, however, as a minimum of a \(2m \times 2m\) space is recommended for standing use with controllers. If one wishes to walk around as well, up to \(4m \times 4m\) is recommended. Most offices and schoolrooms do not offer sufficient space for safe and comfortable use.

Another solution the HTC Vive offers is to project a blue semi-transparent camera feed on top of the rendered scene\(^2\). This allows the user to see what is around them at all times and can allow more safe use. It does obstruct the view of the visualisation to some degree however.

Alternatives to VR like Augmented Reality(AR) illustrates what the ultimate solution might look like. AR adds models into the real world

instead of completely replacing it. The Microsoft Hololens is a good example of an AR system.

Users experienced difficulties when trying to interact with non-immersed people.

An issue that is very unique to this technology is the separation between those who are immersed and those who are not immersed. Depending on how many senses one immerses, anyone that is not immersed might have some trouble getting in contact with those who are immersed and this might lead to some stagnation in interpersonal communication. The testers attempted to show the people outside VR something by holding the object in front of the person or by pointing with the controllers. This lead to several problems with conveying observations and directions. "Look there!" is one such example. Testers had to learn that to show non-immersed people what they are seeing or holding they must make sure it is being looked at by the immersed person. The problem was also observed the other way around. When attempting to direct the gaze of new testers it was often challenging to convey direction. Often taking the testers hand and using the controllers as pointers was required and in some cases the HMD had to be passed over to let the experienced user show the new user by example.

Sometimes testers asked users not immersed in the virtual world to confirm that they were still present and where they were. This might indicate that VR can also have an exclusionary effect on the user, but the actual extend and form of this effect, if it exist, must be researched before reaching any conclusions.

7.7 Research, education and product development have different HMD requirements.

The target end user group will drive the choice of target HMD and thus also the restrictions the implementation and system design. During the thesis nine educators and researchers were asked to comment on what capabilities they would require and expect of a VR HMD. This section will summarise their responses.

In an educational setting there are often many simultaneous users restricting the accepted price to exclude high end options. The Gear VR would be a good example of a cheap solution that only has a single input option; head rotation. The teacher or professor can have a full system where all input options are available while the students, who are being guided by the teacher, can only sit still and watch the lecture.

For research a more complete option seems to be required. High mobility allowing for quick tear down and setup at a new location was more important than multi-user support if forced to choose. As future HMDs become more mobile and support for multiple concurrent HMDs
in the same space becomes more common this choice should vanish. A good compromise between price and quality was the preferred choice.

For meetings, presentations and quick demonstrations an AR system like the Hololens by Microsoft seems ideal since it can be voice controlled, lets the user see those around him and does not require a cable to tether it to a PC\textsuperscript{3}.

\textsuperscript{3}Based on a report by a correspondent at http://fracturereality.io
Chapter 8

Conclusion

Using VR for scientific visualisation is not a new idea. Several attempts have been made since the 1990s, but each attempt has either concluded that the required technology was not yet available or used the CAVE system. The massive cost and space requirement of a CAVE system makes such solutions not viable for most research where space and cost must be kept low. With the new wave of VR HMDs and the power of a high-end desktop computer it has become cheap enough for even modest research projects and the space requirements are much smaller. Further, the space used when in VR can be reused as normal working space by moving the very compact VR HMDs. It is currently at the cusp of being affordable and accessible enough to be mass adopted in education, research and corporate product development. The biggest challenges limiting VR right now is non-algorithm challenges like input hardware and UX design.

8.1 Several challenges solved for monocular systems remain to be solved for VR

Input technology for normal PCs have been standardised to a mouse and a keyboard with some custom input options for special tasks like drawing and scanning. UX design has also been mostly researched and standardised. The same menu elements are used to effectively convey information while ensuring that the user is comfortable with operating the menu. Both of these standardisations do not translate well into VR and highlights the need for research into and standardisation of their VR equivalents.

8.1.1 VR needs a powerful standard input device

Current input devices for VR only allow for a few input options to be made concurrently available to the user. Using voice recognition or controller schemas to compensate for the limited number of buttons will alleviate this deficiency temporarily, but demand for ever more complex interactions performed faster should eventually necessitate an evolution of input devices. Entirely new methods of input must be invented for VR
where the software design and hardware design uses the six degrees of freedom uniquely available to VR.

UX design in VR faces new challenges that must also be considered and designed around. New methods of creating menus and controlling menus must be developed to avoid having menus become unusable due to orientation, overlap and scale issues making the menus inoperable or illegible. Since placing a menu on the camera does not work in VR they must be placed in the scene. Placing each menu on a geometric structure that acts as a ship would let the menus follow the user around the world, but such a structure will obscure the data when scaled and it limits the size of the menus. VR menus require much larger elements to allow the user to accurately read and interact with the menus.

8.1.2 The limit of light speed changed all of physics, VR’s required 90FPS changes algorithm design

Similarly to how light speed is absolute, so is the 90 FPS requirement of comfortable VR. This is different from monocular systems where there is no absolute limit on time used. Some ray tracers take minutes to render a complex scene and the algorithms that are labelled as real-time algorithms only reported less than 30 FPS. Since a VR frame must be rendered in under 10ms most algorithms fail to perform adequately and must be simplified. Another potential issue arises when rendering large data sets. For larger sets data is streamed to the GPU in pieces on monocular systems, but this might be too slow to allow such techniques to be used in VR.

However, it has been shown that reduced ray tracers are capable of being run in VR and as shown in this thesis it is possible to run a full ray marcher with small data sets. So far every attempt at making a volumetric ray tracer in VR has kept all the data in the GPU memory necessitating no transfer of data between the host and the device mid-frame. Future research will explore the viability of transferring data mid-frame and to extend the ray marcher with filters and encoders to remove the pre-process steps used in this thesis.

8.2 Ray tracing volumes in VR is possible if simplified

The implementation of the simple ray tracer presented in this thesis managed to ray trace a volume at 90 FPS and had 1.61ms remaining on a NVIDIA’s GeForce GTX 1080. To achieve this the data was already encoded into the volume, thus removing the need for an expensive filter and an expensive encoder to be added to the ray marcher loop. This severely limits the interactivity of the models, but it should be possible to add some interactivity by using the 1.61ms to add more complex filtering and encoding algorithms to the ray marcher. If the models can be pre-encoded and reduced to fit in the GPU memory a volumetric renderer for scientific data is viable in VR.
8.3 Current VR is best utilised as an extension of monocular systems

The challenges and limitations of current VR make many interactions slow and awkward. Text and number input, precise menu selection and minor adjustments of variables is still best performed with standard 2D interfaces. Further, the extra cost of rendering VR makes it infeasible to render the same data sets in VR as one renders in monocular systems. Finally, since the data transfer time between the host and the device might be too slow, VR rendering algorithms might require the entire data set to be stored in the GPU's memory, thus capping the data sizes available. All of this led us to conclude that VR is best used as an alternative view mode where most of the work with encoding and filtering the data set is done while in monocular mode and then the data set can be pre-processed, reduced and opened in a separate VR mode for viewing. This also allows for a simpler control schema which is ideal for current VR input devices.

8.4 Unlike monocular visualisers, VR’s usefulness grows as model complexity grows

Due to the heightened awareness and control provided by using the human neck as the camera controller and thanks to the natural ability to understand stereoscopic vision humans have, VR's benefits grows exponentially with the interleaved complexity of the model. The more structural overlap and depth there is the more VR outperforms monocular systems.

8.5 Concluding statements

VR has been assessed to be a potentially valuable tool for scientific visualisation since the 1990s and the results of this thesis support that conclusion. VR as a tool for scientific visualisation is incredibly powerful for both geometric and volumetric data. However, given the 90 FPS requirement, most commonly used approaches to scientific visualisation fails to perform due to their high computational requirements. Since VR grows in usefulness as the model becomes more complex the models viewed in VR must be simple enough to render, but also complex enough to merit use of VR. The solution to this problem presented in this thesis was to pre-process the models to avoid expensive data transfers, filtering and encoding during rendering. The best way to ensure support for massive data sets with complex filtering and encoding algorithms is to make VR a separate mode to the standard monocular tools. By making VR a mode you launch into from the main program you can do most of the intensive work outside of VR and get the encoding and filtering correct and then start a process that generates a reduced version of the model that has all the filters and encoders integrated into the data itself. The VR mode then only has to
run a very simple volumetric and geometric renderer with only a minimal set of input options for changing the render that the computer and input devices can support.

8.6 Future work

VR is largely unexplored for scientific visualisation and there are many new areas that must be researched. We suggest three new topics for future research.

It would be interesting to see if alternative input methods, for example voice or glove based input, can alleviate some of the concerns raised in this thesis regarding the limited input options of current VR hardware.

It would also be interesting to explore indirect rendering methods for VR. Decoupling of the VR scene being rendered and model update processing, or mid-frame data transferral are possible methods for allowing more complex algorithms and larger data sets to be processed and rendered while keeping the comfortable frame rate of 90 FPS.

Finally, it would be interesting to see a proper analysis of the pedagogic, psychological and social effects of VR. Tester reports and tester behaviour while using the VR visualiser indicates that there can be adverse social and psychological effects, and that there are real pedagogic potential in VR. We were unable to research this properly since it is out of our field of study. Especially multi-user support is of interest as it should have a large impact on the experience.
Bibliography


Appendix

# A state file generated using paraview version 5.1.2

# setup views used in the visualization

#### import the simple module from the paraview
from paraview.simple import *
#### disable automatic camera reset on 'Show'
paraview.simple._DisableFirstRenderCameraReset()

# Create a new 'Render View'
renderView1 = CreateView('RenderView')
renderView1.AxesGrid = 'GridAxes3DActor'
renderView1.CenterOfRotation = [83.0539321899414, 68.3389301300049, 81.6978797912598]
renderView1.StereoType = 0
renderView1.CameraPosition = [7.617503226001742, 85.38200645650284, 75.03634295864647]
renderView1.CameraFocalPoint = [83.05393218994116, 68.33893013000495, 81.69787979125991]
renderView1.CameraViewUp = [-0.03040658946826605, -0.4777529380130999, -0.8779678636127631]
renderView1.CameraParallelScale = 29.4146320870407
renderView1.Background = [0.32, 0.34, 0.43]

# setup the data processing pipelines

# create a new 'XDMF Reader'
u150ref0xdmf = XDMFReader(FileNames=['C:\Users\NoobsDeSroobs\Downloads\data' + '\volumes\u150-ref0\u150-ref0.xdmf'])
u150ref0xdmf.PointArrayStatus = ['u150-ref0']
u150ref0xdmf.GridStatus = ['u150-ref0_0', 'u150-ref0_1', 'u150-ref0_2', 'u150-ref0_3', 'u150-ref0_4', 'u150-ref0_5', 'u150-ref0_6', 'u150-ref0_7', 'u150-ref0_8', 'u150-ref0_9', 'u150-ref0_10', 'u150-ref0_11', 'u150-ref0_12', 'u150-ref0_13', 'u150-ref0_14', 'u150-ref0_15', 'u150-ref0_16', 'u150-ref0_17']

# create a new 'Slice'
slice1 = Slice(Input=u150ref0xdmf)
slice1.SliceType = 'Plane'
slice1.SliceOffsetValues = [0.0]

# init the 'Plane' selected for 'SliceType'
slice1.SliceType.Origin = [83.0539321899414, 68.3389301300049, 81.6978797912598]
# setup color maps and opacity maps used in the visualization
# note: the Get..() functions create a new object, if needed

# get color transfer function/color map for 'u150ref0'
u150ref0LUT = GetColorTransferFunction('u150ref0')
u150ref0LUT.RGBPoints = [0.0, 0.231373, 0.298039, 0.752941, 448.682847083307, 0.865003, 0.865003, 0.865003, 1422.9655200093969, 0.705882, 0.0156863, 0.14902]
u150ref0LUT.ScalarRangeInitialized = 1.0

# get opacity transfer function/opacity map for 'u150ref0'
u150ref0PWF = GetOpacityTransferFunction('u150ref0')
u150ref0PWF.Points = [0.0, 0.9296875, 0.5, 0.0, 1422.9655200093969, 0.859375, 0.5, 0.0]
u150ref0PWF.ScalarRangeInitialized = 1

# setup the visualization in view 'renderView1'

# show data from slice1
slice1Display = Show(slice1, renderView1)
# trace defaults for the display properties.
slice1Display.ColorArrayName = ['POINTS', 'u150-ref0']
slice1Display.LookupTable = u150ref0LUT
slice1Display.OSPRayScaleArray = 'u150-ref0'
slice1Display.OSPRayScaleFunction = 'PiecewiseFunction'
slice1Display.GlyphType = 'Arrow'
slice1Display.SetScaleArray = [None, '']
slice1Display.ScaleTransferFunction = 'PiecewiseFunction'
slice1Display.OpacityArray = [None, '']
slice1Display.OpacityTransferFunction = 'PiecewiseFunction'

# show color legend
slice1Display.SetScalarBarVisibility(renderView1, True)

# setup the color legend parameters for each legend in this view

# get color legend/bar for u150ref0LUT in view renderView1
u150ref0LUTColorBar = GetScalarBar(u150ref0LUT, renderView1)
u150ref0LUTColorBar.Title = 'u150-ref0'
u150ref0LUTColorBar.ComponentTitle = 'Magnitude'

# finally, restore active source
SetActiveSource(slice1)

Algorithm 8.1: The code above is a standard python script file as exported from Paraview. It encodes the data set and filters it. Further, it sets the initial state of the camera.
# A state file generated using paraview version 5.1.2

# setup views used in the visualization

#### import the simple module from the paraview
from paraview.simple import *
from paraview.vtk import *
import os
import math
#### disable automatic camera reset on 'Show'
paraview.simple._DisableFirstRenderCameraReset()

# Create a new 'Render View'
renderView1 = CreateView('RenderView')
renderView1.ViewSize = [250, 250]
renderView1.AxesGrid = 'GridAxes3DActor'
renderView1.CenterOfRotation = [83.0539321899414, 68.33893013000488, 81.69787979125977]
renderView1.StereoType = 0
renderView1.CameraPosition = [0, 0, 200]
renderView1.CameraFocalPoint = [83.0539321899414, 68.33893013000488, 81.69787979125977]
renderView1.CameraViewUp = [0.3784900662153272, -0.18080221906603647, -0.9077752075029998]
renderView1.CameraParallelScale = 29.414632087040744
renderView1.Background = [0, 0, 0]
renderView1.OrientationAxesVisibility = 0

# create a new 'XDMF Reader'
u150ref0xdmf = XDMFReader(FileNames=[C:\Users\NoobsDeSroobs\Downloads\data\volumes\u150-ref0.udmf])
u150ref0xdmf.PointArrayStatus = ['u150-ref0_0', 'u150-ref0_1', 'u150-ref0_2', 'u150-ref0_3', 'u150-ref0_4', 'u150-ref0_5', 'u150-ref0_6', 'u150-ref0_7', 'u150-ref0_8', 'u150-ref0_9', 'u150-ref0_10', 'u150-ref0_11', 'u150-ref0_12', 'u150-ref0_13', 'u150-ref0_14', 'u150-ref0_15', 'u150-ref0_16', 'u150-ref0_17']
u150ref0xdmf.UpdatePipeline()
data = u150ref0xdmf.GetDataInformation()
bounds = data.DataInformation.GetBounds()

# create a new 'Slice'
slice1 = Slice(Input=u150ref0xdmf)
slice1.SliceType = 'Plane'
slice1.SliceOffsetValues = [0.0]

numSlices = 256.0
numCols = math.ceil(math.sqrt(numSlices))
numRows = math.ceil(numSlices/numCols)
volXSize = bounds[1] - bounds[0]
volXCellSize = volXSize / numCols
volYCellSize = volYSize / numRows

TimeSteps = u150ref0xdmf.TimestepValues

for t in TimeSteps:
    renderView1.ViewTime = t
    for i in range(0, int(numSlices)):
        # image = Image.new("RGBA", (volXSize, volYSize));
        # imageSize = image.size
        # pixels = image.load();
        # init the 'Plane' selected for 'SliceType'
        sliceZ = bounds[4] + (sliceDiff*i);
        slice1.SliceType.Origin = [83.0539321899414, 68.3389301300049, sliceZ]
        renderView1.CameraPosition = [83.0539321899414, 68.3389301300049, sliceZ + 53]
        renderView1.CameraFocalPoint = [83.0539321899414, 68.33893013000488, sliceZ]
        slice1.SliceType.Normal = [0.0, 0.0, 1.0]
        # set up color maps and opacity maps used in the visualization
        # note: the Get..() functions create a new object, if needed
        # get color transfer function/color map for 'u150ref0'
u150ref0LUT = GetColorTransferFunction('u150ref0')
u150ref0LUT.RGBPoints = [0.0, 0.231373, 0.298039, 0.752941, 271.4392904696593, 0.865003, 0.865003, 0.865003, 542.8785809393186, 0.705882, 0.0156863, 0.14902]
u150ref0LUT.ScalarRangeInitialized = 1.0
        # get opacity transfer function/opacity map for 'u150ref0'
u150ref0PWF = GetOpacityTransferFunction('u150ref0')
u150ref0PWF.Points = [11.584660925292043, 0.3984375, 0.5, 0.0, 522.3341517809932, 0.4140625, 0.5, 0.0]
u150ref0PWF.ScalarRangeInitialized = 1.0
        # set the visualization in view 'renderView1'
        # show data from slice1
        slice1Display = Show(slice1, renderView1)
        # trace defaults for the display properties.
slice1Display.ColorArrayName = ['POINTS', 'u150-ref0']
slice1Display.LookupTable = u150ref0LUT
slice1Display.OSPRayScaleArray = 'u150-ref0'
slice1Display.OSPRayScaleFunction = 'PiecewiseFunction'
slice1Display.GlyphType = 'Arrow'
slice1Display.SetScaleArray = [None, '']
slice1Display.ScaleTransferFunction = 'PiecewiseFunction'
slice1Display.OpacityArray = [None, '']
slice1Display.OpacityTransferFunction = 'PiecewiseFunction'

# show color legend
# setup the color legend parameters for each legend in this view
u150ref0LUTColorBar = GetScalarBar(u150ref0LUT, renderView1)
u150ref0LUTColorBar.Title = 'u150-ref0'
u150ref0LUTColorBar.ComponentTitle = 'Magnitude'

# finally, restore active source
SetActiveSource(slice1)

#Save the slice to a PNG image
s = "D:/Slices/Artery/
if not os.path.exists(s):
    os.makedirs(s)
s+= str(i)
s+= ".png"
WriteImage(s)

Algorithm 8.2: The code above is an example of the modified code that encodes, samples and stores each slice to disc in their respective folders.
Official reviews of the VR visualiser

The testers who regularly work with genomes provided the following feedback by email:

<table>
<thead>
<tr>
<th>Original Feedback</th>
<th>English translation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ser dere en fremtid i bruk av VR for visualisering av data som dere jobber på?</td>
<td>Do you see a future in using VR for visualising the kind of data you work with?</td>
</tr>
<tr>
<td>Ja, absolutt. Det å se genomet interaktivt i 3D kan gi bedre forståelse av genom-organisering.</td>
<td>Yes, absolutely. To see the genome interactively in 3D can provide a better understanding of the genome structure.</td>
</tr>
<tr>
<td>Hvilke aspekter av forskningen eller visualisering muliggjør forbedrer VR for dere?</td>
<td>What aspects of your research and of the visualisation of your data does VR make possible or improve?</td>
</tr>
<tr>
<td>VR visualisering tillater oss å forstå mer intuitivt hvor spesifikke gener er i forhold til (1) andre gener, (2) regulatoriske dna-områder og (3)cellekjøremembranen. Dette kan bidra til å oppdage nye sammenhenger og danne mer nøyaktige hypoteser. Det muliggjør også å møte andre samarbeidspartnere (som er fysisk langt unna) i et VR rom og diskutere genom-organisering.</td>
<td>VR visualisation enables us to better understand intuitively where specific genes are in relation to (1) other genes, (2) regulatory DNA-areas and (3) the nuclear membrane. This can aid the discovery of new connections and the development of more precise hypotheses. It also enables meeting other cooperating parties (who are physically far away) in a VR room and discuss genome structure.</td>
</tr>
<tr>
<td>Hvilke aspekter kan VR ikke brukes til? Det er fortsatt problematisk å bruke VR over lengre tid (&gt; 30 min) pga kvalme.</td>
<td>What aspects is VR not suited for? It is still problematic to use VR for an extended period of time (&gt; 30 min) due to nausea.</td>
</tr>
<tr>
<td>Før dere er villig til å adoptere VR som et verktøy, hva mener dere må være på plass? For at VR skal være nyttig i ny forskning, så må vi først og fremst utvikle algoritmer som finner en konsensus [rundt] 3D genom struktur ut fra en populasjon med tusenvis av tilgjengelige strukturer. I tillegg må vi ha muligheten til å hente inn data som er offentlig tilgjengelig for et genom.</td>
<td>Before adopting VR as a tool, what would you require from it? Before VR can be useful in research we must first and foremost develop algorithms that find a consensus regarding 3D genome structure from a population of thousands of available structures. Additionally, we must be able to gather publicly available data for a genome.</td>
</tr>
</tbody>
</table>

The following feedback was provided by the testers who work with brain mapping:
Three-dimensional (3D) visualization of complex structures is an important tool in biomedical research and teaching, and a wide range of continuously improving technologies are applied to acquire, visualize, and analyze 3D images. Volumetric images are typically inspected in three-plane image viewers, or as geometrically rendered 3D objects that can be rotated using interactive viewer tools. In research, stereoscopic 3D visualization of selected objects has occasionally been used for publication purposes. For students and researchers alike it is of key importance to develop a 3D understanding of the structures they study, and 3D visualization has been essential for discovering new patterns of spatial organization.

At the Institute of Basic Medical Sciences several research groups utilize computational tools to render and study complex spatial structures in 3D, ranging from studies of molecular constellations and organization of rodent brain regions. The Neural Systems laboratory at the Division of Anatomy develops database applications for image data, and employs methods for computerized data acquisition, 3-D reconstruction, visualization and quantitative analyses of large neuronal populations and brain regions in the mammalian brain (Leergaard and Bjaalie, 2002; Bjaalie and Leergaard, 2005, 2006; Papp et al., 2015; Kjonigsen et al., 2015; Zakiewicz et al., 2015). These methods are employed to investigate the architecture of large projection systems in the mammalian brain, as well as to study whole brain distribution patterns for gene, molecular, and circuit level data. Also in context of anatomy teaching for students of medicine, dentistry, and nutrition it is an important objective to achieve a 3D understanding of structures at the level of molecules, tissues and whole organs. In this context computational rendering of structures is an important aid for practical courses in microscopic and macroscopic anatomy.

Given the well-recognized value of 3D visualization biomedical research and teaching, it is expected that the emerging new technologies and hardware for virtual reality (VR) visualization will have significant value. So far the use of VR for teaching and research purposes is limited, and little research has been done to determine the added pedagogical value of VR relative to more traditional 3D visualization methods. With increasingly available hardware it is important to determine how VR technology can be applied in biomedical research and teaching.

A pilot virtual reality application was in 2016 developed to explore how VR technology can be used as an additive to teaching medical students the anatomy of the beating human heart based on a standard MRI acquisition of a normal heart. This work indicates a pedagogical potential, but also important limitations related to image quality, available 3D rendering technology, computational hardware, and need for multiple devices to accommodate large student groups (K.S. Åbjørsbråten and T.B. Leergaard, unpublished). In a different project, Magnus Elden (student of informatics, Dept. Informatics, Univ. Oslo) demonstrated VR visualization of geometrically rendered (mesh) files representing complex molecular structures and a volumetric anatomical rat brain reference atlas was explored. These results clearly demonstrate a pedagogical potential for 3D visualization of complex structures, including the rat brain and molecules. Compared to standard 3D visualization programs, VR is unparalleled in representation of figures and structures since it allows interactive stretching, rotation, and virtual immersion in the 3D models. This provides an advanced visual experience that is not achievable on a 2D screen. Functionalities for integrating other information elements into the same applications will likely open for strong interactive visualization tools. Further utilization of VR technology in a pedagogical context would benefit from use of more sophisticated geometrical data sets and implementation of
interactive functionalities tailored for specific pedagogical needs. VR based courses may be
developed as independent self-study modules, or as more complex modules to be used in
practical courses. For teaching of medical students at the Institute of Basic Medical Sciences
it would be desired to apply low cost visualization hardware (to serve large student groups)
and to develop applications allowing multiple simultaneous users (e.g. tutors with multiple
students).

Our conclusion so far is that this is a highly exciting and promising technology with
considerable potential value for biomedical teaching and research.

References

Bjaalie JG, Leergaard TB. Three-dimensional computerized reconstruction from serial sections: cell populations,

Bjaalie JG, Leergaard TB. Three-dimensional visualization and analysis of wiring patterns in the brain:
experiments, tools, models, and databases. In: 'Databasing the brain: From data to knowledge' (eds. SH

Kjonigsen LJ, Lillehaug S, Bjaalie JG, Witter MP, Leergaard TB. Waxholm Space atlas of the rat brain
hippocampal region: Three-dimensional delineations based on magnetic resonance and diffusion tensor

Leergaard TB, Bjaalie JG. Architecture of sensory map transformations: axonal tracing in combination with 3-D
reconstruction, geometric modeling, and quantitative analyses. In: ‘Computational Neuroanatomy:

Papp EA, Leergaard TB, Calabrese E, Johnson GA, Bjaalie JG. Waxholm Space atlas of the Sprague Dawley rat

Zakiewicz IM, Majka P, Wojcik D, Bjaalie JG, Leergaard TB. Three-dimensional histology volume
reconstruction of axonal tract tracing data: exploring topographical organization in subcortical projections