Technical University of Crete

Electrical and Computer Engineering

3D Visualization of Biological data in Ultra High Definition Virtual Reality

Michail Nikiforakis

Thesis Committee

Professor Katerina Mania (ECE)

Professor Zervakis Michail (ECE)

Professor Anastasios Pavlopoulos(FORTH/IMBB)

Chania |Crete , July 2024

Abstract

Visual computing has become an indispensable tool in the field of biology, revolutionizing the way researchers and practitioners understand and interact with complex biological data. The integration of high-resolution Virtual Reality offers an immersive and detailed visualization experience, crucial for advancing our knowledge in areas such as genomics, proteomics, and cellular biology. Recent advancements in VR-based biological visualization often deploy lower-resolution Head Mounted Displays (HMDs), limiting the ability to observe fine structural detail. This work presents an interactive 3D visualization system for biological data in ultra-high-definition VR. The system supports volume rendering techniques, flexible rendering options, and the simultaneous visualization of multiple datasets, all within an 8K VR environment. In this work we collaborated with the Developmental Morphogenesis Lab, part of the Institute of Molecular Biology and Biotechnology of the Foundation for Research and Technology Hellas (IMBB-FORTH). They provided a dataset containing time-lapse microscopy recordings of live developing embryos from the crustacean model organism Parhyale hawaiensis, which was used in order to test and develop our application. Our system's performance is evaluated across varied metrics, including loading time, frames per second, and device latency, showcasing its capabilities in handling large biological datasets. A think aloud study provided expert feedback.

Acknowledgements

Initially, I would like to thank my supervisor professor Katerina Mania for her invaluable guidance and advice throughout this entire project.

I extend my sincere thanks to IMBB Group Leader Anastasios Pavlopoulos and his team for their contribution of the datasets, as well as for their excellent collaboration and feedback during the development of the application.

I am deeply thankful to the members of the Surreal team, whose continuous feedback and support were crucial during the everyday development hours.

Additionally, I want to specially thank my fellow friends Konstantina Mammou, Anna Minadaki, Christos Tsiaousis, Giorgos Protopapadakis and Stefanos Michailos for their love and effort, contributing with ideas and discussions while constructing this report as well as for their unwavering emotional support and moral boost.

Finally, I am profoundly grateful to my family for their endless support and encouragement throughout this journey.

Contents

1 Introduction **1**

1.1 Brief Introduction 1
1.2 Purpose of the Thesis 2
1.3 Brief Description 3
1.4 Structure of the Thesis 5

2 Research Overview **7**

2.1 Introduction 7
2.2 Virtual Reality 8
2.2.1 History of VR 8
2.2.2 Virtual Reality applications 12
2.3 Head Mounted Displays (HMDs) 13
2.3.1 Position Tracking 15
2.3.2 Head Tracking 16
2.3.3 Input methods 17
2.3.4 Other Specifications 18
2.4 Data Visualization 19
2.4.1 2D and 3D Data Visualizations 20
2.4.2 Biological Visualization 23
2.4.3 Virtual Reality Visualization 24
2.5 Volume Rendering 27
2.5.1 Introduction to Volume Rendering 27
2.5.2 Volumetric Data 28
2.5.3 Volume Rendering Techniques 30
2.5.4 Classification and Transfer Functions 33

2.6 Game Engines 34
2.6.1 Unity 35

3 Technological Background and Definitions 37
3.1 Introduction 37
3.2 Data Formats 37
3.2.1 Hierarchical Data Format 38
3.2.2 Nearly Raw Raster Data 39
3.3 Software tools 40
3.3.1 ImageJ 40
3.3.2 HDFview 40
3.3.3 Python 40
3.4 Graphics Pipelines 41
3.4.1 Unity Supported Pipelines 42
3.5 Basic Structure of Unity 43
3.5.1 Assets 43
3.5.2 Scenes 43
3.5.3 GameObjects 43
3.5.4 Components 44
3.5.5 Scripts 44
3.6 VR Structure in Unity 44
3.6.1 XR System 45
3.6.2 XR Origin 45
3.6.3 XR Interaction ToolKit 45
3.6.4 XR Plug-in Management 46
3.6.5 Open XR 46
3.7 Unity Architecture &Project Structure 47
3.7.1 Coordinates - Transform 47
3.7.2 Local Space vs World Space Coordinates 48
3.7.3 Mesh Component 48
3.7.4 Materials, Textures &Shaders 48
3.7.5 Lighting 49
3.7.6 User Interface (UI) 50

3.8 Pimax 8K X 50
3.8.1 Motion 51
3.8.2 Tracking 52
3.8.3 Simulation Sickness 53
3.9 Valve Index Controller 54
3.10 Vive Pro SteamVR Base Station 2.0 54

4 Users View **57**
4.1 Introduction 57
4.2 Controls 58
4.2.1 Navigation Controls 58
4.2.2 Interaction Controls 58
4.2.3 Menu Controls 59
4.3 Open - File 60
4.3.1 Single Instance 60
4.3.2 Multiple Instances 61
4.4 Manipulate/Observe 61
4.4.1 Light 62
4.4.2 Move 62
4.4.3 Rotation/scale 63
4.4.4 Render Mode 64
4.4.5 Visible Range 64
4.4.6 Cross Plane 64
4.4.7 Time moments mode 65
4.4.8 Channels mode 65
4.5 Transfer Function 65
4.5.1 Edit 66
4.5.2 Save/Load 67
4.6 Visualization Results 67

5 Implementation **69**
5.1 Introduction . . . 69
5.2 Data acquisition . . . 70
5.3 Preprocessing . . . 70
5.3.1 Data Structure . . . 70
5.3.2 Data Decomposing . . . 71
5.3.3 Data Formation . . . 72
5.3.4 Data Export . . . 73
5.4 Data Visualization . . . 74
5.4.1 File Import . . . 76
5.4.2 Object Creation . . . 76
5.4.3 Render Modes . . . 77
5.4.4 Transfer Function . . . 80
5.4.5 Finalize . . . 81
5.5 Virtual Environment . . . 82
5.5.1 Device connection . . . 82
5.5.2 VR setup in Unity . . . 82
5.5.3 Camera . . . 83
5.5.4 Controls . . . 83
5.5.5 Object Interaction . . . 85
5.5.6 UI interaction . . . 85
5.6 Tools and Features . . . 85
5.6.1 Edit/Load/Save Transfer Functions . . . 86
5.6.2 Cross Plane . . . 87
5.6.3 Open Modes . . . 87
5.7 User Interface . . . 88
5.7.1 UI Activation . . . 88
5.7.2 Main Panels . . . 89

6 Evaluation ,Results &Future Work **93**
6.1 Introduction . . . 93
6.2 Evaluation Method . . . 93
6.2.1 Performance Metrics . . . 93

6.2.2 Think Aloud Methodology 95
6.3 Evaluation Results 96
6.3.1 Performance Metrics results 96
6.3.2 Think Aloud Methodology results 97
6.4 Future Work 98

References **112**

List of Figures

1.1 Use of PiMax and Valve index controllers 3
1.2 3D Visualized Data results 4

2.1 Sensorama Prototype 9
2.2 Head Mounted Display Prototype 9
2.3 Oculus Rift campaign 11
2.4 Head Mounted Displays 14
2.5 Position Tracking with Sensors 15
2.6 Degrees Of freedom 16
2.7 Field of View Comparison 20
2.8 2D Visualizations Examples 21
2.9 Ray Marching technique 27
2.10 Volume Data 29
2.11 CT head rendered in the four main volume rendering modes: (a) X-ray; (b) MIP; (c) Iso-surface; (d) DVR. 30

3.1 Pimax 8KX 51
3.2 Controllers and Tracking bases 55

4.1 Open File menu 60
4.2 Object Manipulation Panel 62
4.3 Light in Application 63
4.4 Rotation and Scale sliders 63
4.5 DVR and MIP Examples 64
4.6 Cross Section plane 65
4.7 Transfer function editing panel 66

4.8 Results of 3d rendered object for different angles using Direct Volume Rendering technique 67
4.9 Membrane and Cells with DVR technique 68
4.10 Membrane and Cell with MIP technique 68

5.1 HDF file 71
5.2 File Tree 72
5.3 File decomposition 73
5.4 Data Formation 73
5.5 Data Export 74
5.6 Image Importer 75
5.8 MIP implementation 79
5.9 DVR implementation 80
5.10 The main function for creating the object 81
5.11 Ui Panels flow chart 92

6.1 Time measurement code 94
6.2 Frame Pes Second measurement script 95

Chapter 1

Introduction

1.1 Brief Introduction

In recent years, Virtual Reality (VR) technology has undergone a significant transformation, making it more accessible and user-friendly. The evolution of VR devices, from cumbersome and costly equipment to streamlined and affordable headsets, has expanded its accessibility, providing immersive and interactive experiences for a broader audience [1, 2].

The proliferation of VR technology goes beyond entertainment and gaming; it has begun to permeate various domains of our world. From education to healthcare, from architecture to art, VR presents limitless opportunities for innovation and exploration [3, 4]. With the increasing prevalence of VR devices, new horizons open up for investment in the sector, leading to the development of more advanced tools and technologies that can be harnessed by scientists and researchers [5].

With technology rapidly advancing, VR has the potential to impact every sector of our world. It transcends traditional two-dimensional screens, ushering us into three-dimensional, interactive worlds that transform how we perceive, interact with, and comprehend our surroundings [6].

In the research domain, VR offers unparalleled opportunities. It facilitates interactivity, enabling researchers to explore and manipulate data in novel ways. The immersive nature of VR enhances data analysis and visualization, enabling comprehensive examination of complex datasets. Researchers can leverage VR to break new ground, make significant contributions in their respective fields, and advance problem-solving and data exploration [7, 8].

In the field of biology, VR opens exciting avenues for exploration. By visualizing biological data in ultra-high definition within a VR environment, researchers gain invaluable tools for studying intricate biological structures [9, 10]. Through advanced volume rendering techniques and interactive experiences, biologists can delve into microscopic worlds, gaining insights that were once challenging to obtain [11]. This thesis explores the integration of 3D visualization techniques for biological data within ultra-high-definition virtual reality, offering new opportunities for exploration and discovery in biology.

1.2 Purpose of the Thesis

This thesis embarks on a journey to harness the transformative potential of Virtual Reality (VR) technology in the realm of biology. The primary goal is to bridge the gap between the intricate biological data generated in the field and the understanding of these datasets by researchers and scientists. VR technology, with its enhanced accessibility and immersive capabilities, offers a unique platform to visualize and analyze biological data in ways previously unattainable.

The central purpose of this thesis is to explore the use of ultra-high-definition virtual reality HMDs for 3D visualization techniques applied to biological data. By leveraging high fidelity VR technologies, the thesis aims to offer biology researchers a new dimension in data analysis and exploration. Through volume rendering techniques, it seeks to create interactive, three-dimensional representations of biological specimens, thus enhancing the ability to study intricate structures in high detail.

Furthermore, this thesis aims to provide a comprehensive overview of the research objectives and methodology underlying the utilization of VR technology in biology. It explores the development of user-friendly interfaces and interaction methods, ensuring that researchers can navigate the VR environment seamlessly and with precision.

Moreover, this research strives to uncover the potential benefits of integrating VR into the field of biology. By allowing biologists to immerse themselves in biological data and interact with it in a 3D virtual space, it holds the potential to advance research in this domain. Through the use of VR, this study seeks to explore novel avenues for enhanced precision, interactivity, and visualization, with the ultimate goal of contributing to a deeper understanding of complex biological structures.

1.3 Brief Description

In this thesis, we developed an application for the 3D visualization of biological data in ultra-high definition virtual reality. The application takes Nearly Raw Raster Data (NRRD format) as input and generates a 3D visualization of this data within a virtual reality environment using Volume Rendering techniques. Additionally, the application provides tools for users to configure the visual representation of the data, enabling more accurate and detailed observations.

The hardware used for this project includes the Pimax 8KX, an ultra-high fidelity Head Mounted Display (HMD), to present the virtual environment to users. The Pimax 8KX, known for its exceptional resolution and wide field of view, ensures a highly immersive and detailed visualization experience. In combination with the Valve Index controllers, users can interact with the virtual environment in an intuitive and interactive manner. This setup allows for a seamless and engaging experience when observing and analyzing the 3D biological data, enhancing the overall effectiveness of the application.

Figure 1.1: Use of PiMax and Valve index controllers

The datasets used for the development of this work were provided by the Developmental Morphogenesis Lab of the Institute of Molecular Biology and Biotechnology and were originally in Hierarchical Data Format (HDF). To facilitate their use in our application, these datasets were converted into Nearly Raw Raster Data (NRRD) format through a Python script. This pre-processing step was essential to standardize the data format, ensuring compatibility with the visualization tools and enabling efficient handling and rendering of the biological data within the virtual reality environment.

The Unity game engine was then used to visualize the data. The visualization techniques employed include volume rendering algorithms such as direct volume rendering and maximum intensity projection. These techniques were implemented through custom

shaders in Unity, resulting in a 3D reconstructed object with various visual configuration options. Users can configure the transforms functions of the 3D model and adjust visibility density windows, allowing for a customizable and detailed exploration of the biological data.

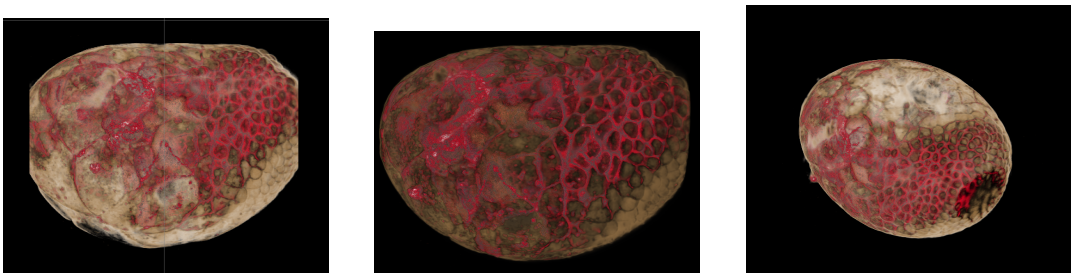


Figure 1.2: 3D Visualized Data results

After completing the visualization step in the desktop environment, Unity's array of virtual reality tools were employed to transition to a virtual reality (VR) setting. Specifically, the OpenXR standard was implemented using the XR plug-in, and tools such as XR Origin and XR Interaction Toolkit were utilized to shape the VR experience. This transition enabled the application to seamlessly integrate with various VR hardware setups, enhancing immersion and interaction capabilities for users. By leveraging Unity's robust VR development ecosystem, we ensured that the 3D biological data could be explored and analyzed within a fully immersive virtual environment.

To enhance the application experience, several tools and functionalities were integrated. A graphical editor for transfer functions was added to facilitate easier configuration, enabling real-time adjustments and fine-tuning of the visual representation of objects. Additionally, the ability to add a cross-section plane was implemented, facilitating the study of internal structures within the objects. Furthermore, additons were made to the opening modes, allowing users to open multiple objects simultaneously for comparative analysis.

After completing the implementation, we employed performance metrics to evaluate our application. These metrics included the loading times of datasets, the frames per second at which the application ran, and the latency of the Head Mounted Display (HMD) during use. Additionally, a think-aloud methodology was conducted with expert Virtual Reality developers, where their feedback and thoughts were gathered during their use

of the application. This qualitative evaluation provided valuable insights into usability issues, interface clarity, and overall user experiencel.

The primary contribution of this thesis is the development of a system for 3D biological visualization in ultra-high-definition virtual reality, offering significantly higher fidelity visualizations compared to existing similar works. Additionally, another significant contribution is the implementation of the ability to open multiple datasets simultaneously, facilitating comparative analysis within the virtual reality environment.

1.4 Structure of the Thesis

- Chapter 1: Introduction

  This chapter sets the stage for the research by discussing the current state of VR devices, their accessibility and potential benefits. It introduces the wide applicability of VR in various domains and highlights the opportunities it offers in the research domain, particularly in biology.

- Chapter 2: Research Overview

  In this chapter, we provide an overview of the research objectives and goals. We outline the scope of the study and present a comprehensive overview of the research. This chapter serves as a roadmap for the entire thesis.

- Chapter 3: Technological Background and Definitions

  This chapter delves into the essential technological concepts and definitions related to the thesis. It provides the necessary context for readers to understand VR technology, volume rendering, and key terms used throughout the study.

- Chapter 4: Users View

  This chapter provides a comprehensive insight into the application and its capabilities. It highlights what users can accomplish within the 3D environment, offering detailed explanations and images of the application's functionalities. The chapter covers essential controls, explains how to manipulate and observe data and details the procedures adjusting transfer functions, providing users with a thorough understanding of the application's tools and features.

- Chapter 5: Implementation

  This chapter focuses on the practical implementation of the VR application. It covers the step-by-step process of integrating volume rendering techniques, data preparation, and user interaction elements. We discuss the technical aspects of the application's development.

- Chapter 6: Evaluation,Results and Future Work

  In this final chapter, we present the results and findings of the research. We evaluate the effectiveness of the VR application in visualizing biological data and discuss any limitations encountered. Additionally, we explore potential areas for future research and development in the field of VR and biology.

Chapter 2

Research Overview

2.1 Introduction

Chapter 2 of this thesis embarks on a comprehensive exploration of the historical evolution and contemporary state of virtual reality (VR) technology and head-mounted displays (HMDs), with a particular emphasis on their interactive aspects. VR technology, along with its interactive capabilities, has undergone significant transformations, shaping immersive experiences and redefining human-computer interactions [2]. This chapter aims to provide insights into the development trajectories and current landscape of VR and HMDs, acknowledging their pivotal roles in facilitating immersive and interactive environments. By exploring the nuances of VR and HMDs, Chapter 2 establishes a foundational understanding of the technological context that surrounded the development of the VR application within the research framework.

Furthermore, the exploration extends to encompass the intricate realm of data visualization, particularly focusing on its applications within biology. Building upon the interactive capabilities inherent in VR environments, Chapter 2 delves into various visualization techniques, including an in-depth exploration of volume rendering methodologies. The chapter discusses the decision to leverage Unity as the game engine for the development of the VR application, considering its versatility and accessibility in creating interactive experiences. Moreover, Chapter 2 elucidates the nature and significance of the biological datasets utilized, aiming to contextualize their role within the research framework. Through this exploration, Chapter 2 seeks to shed light on the technological context during the developmental period of the VR application, bridging the interactivity

of VR environments with the intricate visualization techniques that can be used in the field of biology.

2.2 Virtual Reality

2.2.1 History of VR

2.2.1.1 Introduction to VR

Virtual reality (VR) traces its roots back to the mid-20th century when pioneering efforts in the field of computer graphics laid the groundwork for immersive digital experiences (Brooks, 1999). These early endeavors set the stage for the development of VR technology, envisioning computer-generated environments that users could interact with in real-time. The conceptualization of VR represented a paradigm shift in human-computer interaction, opening up new possibilities for creating simulated worlds and immersive experiences. As technology advanced, VR evolved from theoretical concepts to tangible applications, reshaping how we perceive and interact with digital content. The historical origins of VR underscore its enduring significance as a transformative technology with diverse applications across various domains.

2.2.1.2 Early Concepts and Theoretical Foundations

The concept of virtual reality (VR) was first articulated by Ivan Sutherland in his seminal paper "The Ultimate Display" in 1965, marking a significant milestone in the history of computer science and human-computer interaction [12]. In this visionary work, Sutherland proposed the idea of a computer-generated environment that users could interact with in real-time, thereby laying the theoretical foundations for VR technology. The notion of the "Ultimate Display" captured the imagination of researchers and engineers, sparking interest in creating immersive digital experiences that transcended the constraints of traditional media. Sutherland's pioneering vision paved the way for subsequent developments in VR research and innovation, inspiring generations of scientists, engineers, and artists to explore the possibilities of simulated environments and interactive storytelling.

2.2.1.3 Sensorama and Early Prototypes

In the 1960s, Morton Heilig pioneered the development of immersive multimedia experiences with the creation of the Sensorama [13]. This groundbreaking device represented an early prototype of virtual reality technology, offering users a multisensory journey through stereoscopic 3D visuals, surround sound, and motion effects as shown in Figure 2.1. The Sensorama aimed to provide an immersive and engaging experience that transcended traditional forms of entertainment and storytelling. Through its innovative design and integration of various sensory stimuli, the Sensorama foreshadowed the potential of VR to transport users to simulated worlds and evoke visceral experiences. Heilig's visionary contributions laid the groundwork for subsequent advancements in VR technology, inspiring future generations of researchers and engineers to explore new frontiers in immersive storytelling and digital interaction. The Sensorama stands as a testament to the ingenuity and creativity of early pioneers in the field of virtual reality, heralding a new era of sensory-rich experiences and interactive entertainment.

Source: https://www.historyofinformation.com/image.php?id=2078

Figure 2.1: Sensorama Prototype

2.2.1.4 The Birth of Head-Mounted Displays

The emergence of head-mounted displays (HMDs) in the 1960s and 1970s marked a significant milestone in the development of virtual reality (VR) technology, ushering in a new era of immersive digital experiences [14]. These early prototypes of HMDs (Figure 2.2) represented a breakthrough in human-computer interaction, enabling users to engage with virtual environments in a more intuitive and immersive manner. By mounting display screens directly onto the user's head, HMDs provided a seamless interface for exploring simulated

Source: https://www.researchgate.net/figure/The-first-head-mounted-display-designed-by-Dr-I-E-Sutherland-Reproduced-with_fig2_341058477

Figure 2.2: Head Mounted Display Prototype

worlds and interacting with digital content. This novel form of display technology laid the foundation for future innovations in VR hardware, paving the way for more sophisticated and immersive VR experiences. The advent of HMDs democratized access to VR technology, making it more accessible to researchers, developers, and enthusiasts alike. As HMD technology continued to evolve, it would go on to revolutionize industries ranging from gaming and entertainment to healthcare and education, catalyzing a paradigm shift in how we perceive and interact with digital information.

2.2.1.5 Military Applications and Flight Simulation

In the 1980s, virtual reality (VR) technology gained substantial traction through the development of flight simulators tailored for military training purposes, exemplifying the practical applications of immersive environments in real-world scenarios [15]. These pioneering flight simulators served as invaluable tools for training military personnel in complex and high-stakes situations, providing a safe and controlled environment for pilots to hone their skills and simulate challenging flight conditions. By leveraging VR technology, military organizations could offer trainees realistic and dynamic training experiences that closely mirrored actual flight operations, thereby enhancing readiness and proficiency levels. The adoption of VR-based flight simulators underscored the transformative potential of immersive technologies in revolutionizing traditional training methodologies and preparing military personnel for the rigors of combat and operational missions. Moreover, the success of VR-based flight simulation systems laid the groundwork for broader applications of VR technology across diverse industries, setting the stage for continued innovation and exploration in the field of immersive digital experiences.

2.2.1.6 Winter and Rebirth

Despite the initial enthusiasm surrounding virtual reality (VR), the industry encountered a significant downturn in the early 2000s, referred to as the "VR winter" [16]. This period was characterized by a dearth of viable applications and technological limitations that hindered the widespread adoption and commercialization of VR technology. The VR winter prompted a reevaluation of the potential of VR and underscored the need for significant advancements in hardware, software, and content development to reignite interest in the technology.

The 2010s witnessed a remarkable resurgence of interest in VR technology, driven by notable advancements across various fronts, including hardware, software, and content development [17] . This revitalization of the VR industry heralded a new era of innovation and investment, as developers and enthusiasts explored the untapped potential of immersive digital experiences. The renewed interest in VR technology paved the way for the development of cutting-edge VR hardware, sophisticated software applications, and compelling content, positioning VR as a transformative force in entertainment, education, healthcare, and beyond.

The launch of the Oculus Rift Kickstarter campaign in 2012 by Palmer Luckey marked a pivotal moment in the history of VR. The overwhelming success of the campaign, coupled with the subsequent acquisition of Oculus by Facebook in 2014, signaled the dawn of modern VR and catalyzed a wave of excitement and investment in the industry. The Oculus Rift emerged as a trailblazer in the VR landscape, showcasing the potential of high-quality, immersive VR experiences and inspiring a new generation of developers and enthusiasts to explore the possibilities of the medium. The Oculus Rift's success served as a catalyst for the broader adoption and commercialization of VR technology, ushering in an era of unprecedented innovation and creativity in the realm of immersive digital experiences.

Source: https://www.ncfacanada.org/kickstarter-backers-of-oculus-rift-angrily-reject-facebook-takeover/

Figure 2.3: Oculus Rift campaign

2.2.1.7 New Era

Today, virtual reality (VR) technology is experiencing widespread adoption across a spectrum of industries, underscoring its versatility and transformative potential [18]. In the realm of gaming and entertainment, VR offers immersive experiences that transport users to fantastical worlds and redefine interactive storytelling. The healthcare sector leverages VR for medical training, surgical simulations, pain management, and therapy, revolutionizing patient care and medical education [19]. In education, VR serves as a powerful tool for experiential learning, enabling students to explore historical sites, engage with

complex concepts, and interact with immersive educational content [16]. Additionally, architects and designers utilize VR for visualizing architectural plans, conducting virtual walkthroughs, and showcasing design concepts to clients, streamlining the design process and enhancing collaboration [20]. Across these diverse industries, VR technology is driving innovation, enhancing productivity, and reshaping how we experience and interact with digital content. As VR continues to evolve and mature, its impact on society, culture, and industry is poised to deepen, unlocking new possibilities and redefining the way we perceive and engage with the world around us [21].

2.2.2 Virtual Reality applications

Virtual Reality has evolved beyond its initial function as a visualization tool, emerging as a dynamic medium for immersive experiences. In this section, we explore the concept of Interactive Virtual Reality, emphasizing its significance and applications. Interactive VR allows users to actively engage with virtual environments in real-time, enabling them to shape the narrative, dynamics, and outcomes of their virtual experiences. Unlike passive forms of media consumption, Interactive VR empowers users to become active participants in the immersive worlds they inhabit. This interactivity is facilitated through a variety of input devices, including hand controllers, motion trackers, and gesture recognition systems, which enable users to navigate, interact with objects, and manipulate elements within the virtual environment.

Virtual Reality has found applications across various domains, including gaming, education, training, therapy, research, and medical tools, where its ability to engage users in meaningful and interactive experiences has proven invaluable. In gaming, Virtual Reality offers players unprecedented levels of immersion and agency, enabling them to interact with virtual worlds in ways that were previously unimaginable [18]. Educational institutions are harnessing Virtual Reality to create immersive learning environments, where students can explore scientific concepts, historical events, and cultural landmarks firsthand [16]. In training simulations, Interactive VR allows trainees to practice real-world tasks and scenarios in a safe and controlled environment, facilitating skill acquisition and knowledge transfer [22]. Moreover, Interactive VR has demonstrated promise in therapeutic settings, where it is utilized to treat phobias, PTSD, and other psychological

disorders through exposure therapy and immersive relaxation techniques [23]. Additionally, in research, Virtual Reality is employed to conduct experiments, simulations, and data visualization, enabling scientists and researchers to explore complex phenomena and analyze data in immersive environments [24]. In medical tools, Virtual Reality is utilized for surgical training, patient education, and medical imaging analysis, providing healthcare professionals with innovative tools for diagnosis, treatment, and patient care [25, 10, 8].

In summary, Virtual Realityrepresents a significant transformation in our interaction with digital content. By empowering users to actively engage in and shape their virtual experiences, Interactive VR provides a revolutionary platform for entertainment, education, training, therapy, research, and medical tools. As technology advances and interactive VR experiences become more sophisticated and accessible, the opportunities for innovation and creativity in this emerging field continue to expand, promising exciting developments and advancements in the future.

In this thesis, we developed an application that integrates interactive VR with the domain of biological research, aiming to provide advanced tools to facilitate and enhance the field of biological research.

2.3 Head Mounted Displays (HMDs)

Head-mounted displays (HMDs) are wearable devices that immerse users in virtual environments by presenting stereoscopic images to each eye (Figure 2.4). They typically consist of a head-worn unit containing display screens, lenses, and motion sensors, along with straps or frames for secure attachment to the user's head. HMDs vary in design, from lightweight and compact models for casual use to more sophisticated systems designed for professional applications.

HMDs play a pivotal role in integrating human senses into the virtual reality experience. By providing visual and auditory stimuli, they create a sense of immersion and presence, transporting users to virtual worlds that engage multiple sensory modalities. The quality of the visual and auditory feedback delivered by HMDs significantly influences the realism and effectiveness of the virtual experience [26].

Advancements in hardware technology have propelled the evolution of HMDs, making them lighter, more comfortable, and capable of delivering higher-resolution graphics and

smoother motion tracking [27, 28]. The development of high-fidelity displays, improved optics, and advanced motion sensors has enhanced the level of immersion and realism achievable in virtual environments [29]. These technological innovations have expanded the possibilities for applications ranging from entertainment and gaming to education, training, and healthcare [30].

Source: https://medium.com/echo3d/the-best-virtual-reality-headsets-out-there-7e7153155b31

Figure 2.4: Head Mounted Displays

Head-mounted displays (HMDs) boast a diverse range of hardware capabilities designed to enhance the immersive virtual reality (VR) experience. These capabilities encompass high-resolution displays that render lifelike visuals with crisp detail and vibrant colors, immersing users in visually stunning virtual environments. Advanced optics and lens systems ensure optimal clarity and field of view, minimizing distortion and maximizing the user's visual perception within the virtual space [29, 31]. Additionally, ergonomic design features, such as adjustable straps and cushioned padding, ensure comfort during extended VR sessions, promoting user engagement and endurance. Furthermore, integrated motion sensors and inertial measurement units (IMUs) enable precise tracking of head movements, allowing for seamless interaction with virtual objects and environments [32]. HMDs also integrate audio technologies, including built-in speakers or headphone

jacks, to deliver immersive spatial soundscapes that complement the visual experience. Collectively, these hardware capabilities contribute to the compelling and immersive nature of VR interactions, setting the stage for the exploration of advanced tracking systems and technical specifications that further enhance the VR user experience.

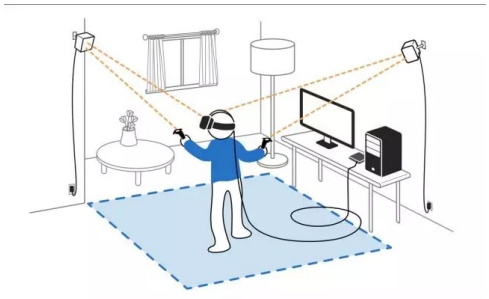
In this thesis, we utilized a cutting-edge VR headset equipped with ultra-high-resolution screens and an expansive field of view to provide an immersive and high-quality experience for biological data analysis.

2.3.1 Position Tracking

Position tracking is a fundamental aspect of virtual reality (VR) technology that enables precise monitoring of the user's physical movements within the virtual environment. This tracking mechanism allows VR systems to accurately determine the user's location and orientation in three-dimensional space, facilitating natural and intuitive interactions with virtual objects and environments. Various position tracking techniques have been developed to achieve this goal, ranging from basic to advanced implementations.

Inertial Measurement Units (IMUs): One common approach to position tracking involves the use of inertial measurement units (IMUs), which comprise sensors such as accelerometers and gyroscopes. IMUs measure changes in velocity and rotational motion, enabling real-time tracking of the user's head movements and orientation [33]. While IMUs provide low-latency tracking and are suitable for standalone VR experiences, they may suffer from drift and accuracy issues over time, especially during rapid movements or extended usage [34].

External Tracking Systems: External tracking systems employ external sensors or cameras placed around the user's environment to track the position and movement of VR hardware, such as headsets or controllers. These systems use markers or visual cues to triangulate the user's position in space accurately [35]. Examples include optical tracking systems like the Oculus Constellation and the HTC Vive's Lighthouse tracking system, which use infrared sensors and lasers, respectively,



Source: https://4experience.co/vr-tracking-meet-degrees-of-freedom/

Figure 2.5: Position Tracking with Sensors

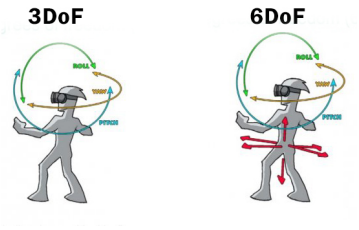
to track the precise location of VR devices within a designated play area (Figure 2.5).

Hybrid Tracking Solutions: Some VR platforms utilize hybrid tracking solutions that combine multiple tracking technologies to achieve optimal performance and accuracy. These solutions may integrate IMUs for low-latency tracking and external sensors for absolute positioning, offering a balance between responsiveness and precision [36]. By combining different tracking modalities, hybrid systems aim to mitigate the limitations of individual tracking methods and provide a more robust and immersive VR experience.

In this thesis, we employed a hybrid system consisting of IMUs and an external tracking system. Although the headset used is equipped with the necessary sensors for internal position tracking, an external tracking system was also used due to the superior accuracy of the combined system.

2.3.2 Head Tracking

Head tracking is a pivotal component of VR technology that enables users to control their viewpoint within virtual environments by tracking the movement of their head in real-time [37]. By accurately capturing head orientation and position, VR systems can adjust the perspective displayed to the user, creating a sense of immersion and presence in the virtual world. Head tracking is crucial for maintaining visual consistency and ensuring that the virtual environment reacts to the user's movements naturally and responsively [38].



Source: https://blog.bigimmersive.com/positional-tracking-in-vr-a-review-of-tracking-systems-their-features/

Figure 2.6: Degrees Of freedom

Head tracking can be implemented using various technologies, including inertial measurement units (IMUs), optical sensors, and depth-sensing cameras. IMUs, comprising accelerometers and gyroscopes, offer low-latency tracking and are commonly integrated into VR headsets to capture rotational movements of the user's head [39]. Optical tracking systems, such as those employed by the Oculus Rift and HTC Vive, utilize external sensors or cameras to precisely monitor head position and orientation within a defined tracking volume [37, 40].

Head tracking systems are characterized by their degrees of freedom (DoF), which determine the range of movements they can accurately capture. Basic head tracking systems may offer three degrees of freedom (3DoF), tracking rotational movements around the vertical, horizontal, and lateral axes. Advanced systems, known as six degrees of freedom (6DoF) tracking, additionally track translational movements along the X, Y, and Z axes (Figure 2.6), allowing users to move freely within the virtual space [37].

Accurate head tracking is essential for creating compelling and immersive VR experiences that respond seamlessly to user input. By synchronizing virtual camera movements with the user's head movements, VR applications can enhance the sense of presence and realism, making users feel as if they are truly inhabiting the virtual environment [41]. Smooth and responsive head tracking is crucial for preventing motion sickness and discomfort, ensuring a comfortable and enjoyable VR experience for users [42].

Despite advancements in head tracking technology, challenges such as latency, jitter, and occlusion may still affect the quality of tracking in certain scenarios [43]. Minimizing latency and ensuring consistent tracking accuracy are ongoing areas of research and development in the VR industry [42]. Additionally, optimizing head tracking algorithms and sensor fusion techniques can help mitigate issues related to sensor drift and environmental interference, enhancing the overall performance and reliability of VR head tracking systems [41].

In this thesis, head tracking is implemented through a hybrid system. While the IMU-based internal system offers only 3 degrees of freedom, combining it with an external tracking system enables the hybrid solution to achieve 6 degrees of freedom.

2.3.3 Input methods

In virtual reality (VR) environments, hand tracking, controllers, and eye tracking serve as primary input methods, enabling users to interact with virtual objects and environments with natural gestures, movements, and gaze. These input devices play a crucial role in enhancing immersion and facilitating intuitive interaction within VR experiences.

Hand Tracking Technology: Hand tracking technology enables the detection and analysis of hand movements and gestures in real-time, allowing users to manipulate virtual objects using their hands directly [44]. Optical sensors, depth cameras, and machine learning algorithms are commonly employed to capture and interpret hand movements

accurately. Advanced hand tracking systems can recognize intricate gestures and finger movements, providing users with precise and intuitive control over virtual interactions [45].

Controller-Based Input: VR controllers, such as handheld devices equipped with buttons, triggers, and joysticks, offer users tactile feedback and enhanced control over virtual environments. These controllers enable users to perform a wide range of actions, including grabbing objects, navigating menus, and interacting with virtual interfaces [46]. The ergonomic design and tactile feedback mechanisms of VR controllers enhance the sense of presence and immersion, making interactions within VR environments more intuitive and engaging.

Eye Tracking Integration: Eye tracking technology enables the precise measurement and analysis of a user's gaze and eye movements within VR environments. By tracking the direction of the user's gaze, eye tracking systems provide valuable insights into attention and visual focus, allowing for more natural and responsive interactions [47, 48]. Eye tracking also enables features such as foveated rendering, where graphics processing power is allocated based on the user's gaze, optimizing performance and visual quality [49].

Hybrid Input Solutions: Some VR systems integrate hand tracking, controller-based input, and eye tracking technologies, offering users flexibility in how they interact with virtual environments. Hybrid solutions leverage the strengths of each input method, allowing users to switch seamlessly between hand gestures, controller inputs, and gaze-based interactions based on the task or context. This versatility enhances user comfort and accommodates different interaction preferences, catering to a diverse range of VR experiences and applications [50].

In this thesis, a controller-based input system was preferred. Controllers provide the necessary range of motion and accuracy required for effective interaction with the virtual environment and biological data analysis.

2.3.4 Other Specifications

In addition to hand tracking, controllers, and eye tracking, several other specifications are essential considerations in the design and functionality of head-mounted displays (HMDs)

for virtual reality (VR) experiences. These specifications include display resolution, refresh rate, field of view, optics, and lenses.

Display resolution refers to the number of pixels displayed on the screen, directly impacting image clarity and detail within the VR environment. A higher resolution display enhances visual fidelity and reduces the visibility of individual pixels, resulting in a more immersive experience for users [51].

The **refresh rate** of an HMD determines how frequently the display updates images per second, measured in Hertz (Hz). A higher refresh rate reduces motion blur and flickering, leading to smoother and more comfortable viewing experiences, particularly during fast-paced VR interactions [31].

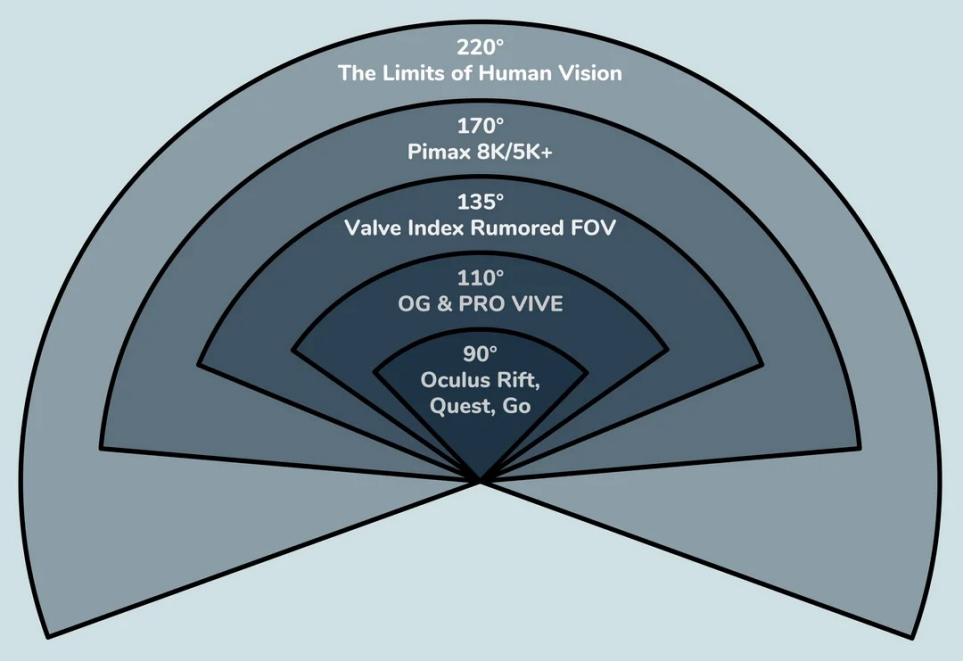
Field of view (FOV) defines the extent of the virtual environment visible to the user through the HMD lenses. HMDs with wider FOV (Figure 2.7) provide a more expansive and immersive view of the virtual world, enhancing the sense of presence and spatial awareness for users [52].

Optics and lenses play a crucial role in shaping the quality and clarity of the visual experience in VR. High-quality optics and precision-engineered lenses help minimize distortions, aberrations, and color fringing, ensuring sharp and accurate image reproduction across the entire field of view [51]. Additionally, lens coatings and anti-glare treatments improve light transmission and reduce reflections, enhancing visual comfort and reducing eye strain during extended VR sessions [31].

In this thesis, the HMD used is equipped with two screens (one for each eye) with a resolution of 3840x2160 each. Additionally, it uses Fresnel lenses that help achieve a significantly wide field of view (170+ degrees).

2.4 Data Visualization

Data visualization refers to the graphical representation of data to extract meaningful insights and patterns. It involves the use of visual elements such as charts, graphs, and maps to present complex datasets in a clear and understandable format [53]. Data visualization enables users to explore data, identify trends, and communicate findings effectively. It plays a crucial role in various fields, including business analytics, scientific research, and decision-making processes [54, 55, 56].



Source: https://www.reddit.com/r/OculusQuest/comments/b9veej/fov_comparison/

Figure 2.7: Field of View Comparison

2.4.1 2D and 3D Data Visualizations

2.4.1.1 2D Visualization

Two-dimensional (2D) data visualizations are graphical representations that display information using two axes, typically x and y coordinates. They are fundamental tools in data analysis and presentation, allowing users to visualize relationships, patterns, and trends within datasets. The historical development of 2D data visualizations traces back to the early methods of charting and graphing data, which were essential for conveying information in a visual format [57]. Over time, advancements in technology and design have led to the evolution of 2D visualizations into a diverse array of chart types and graphical representations [57].

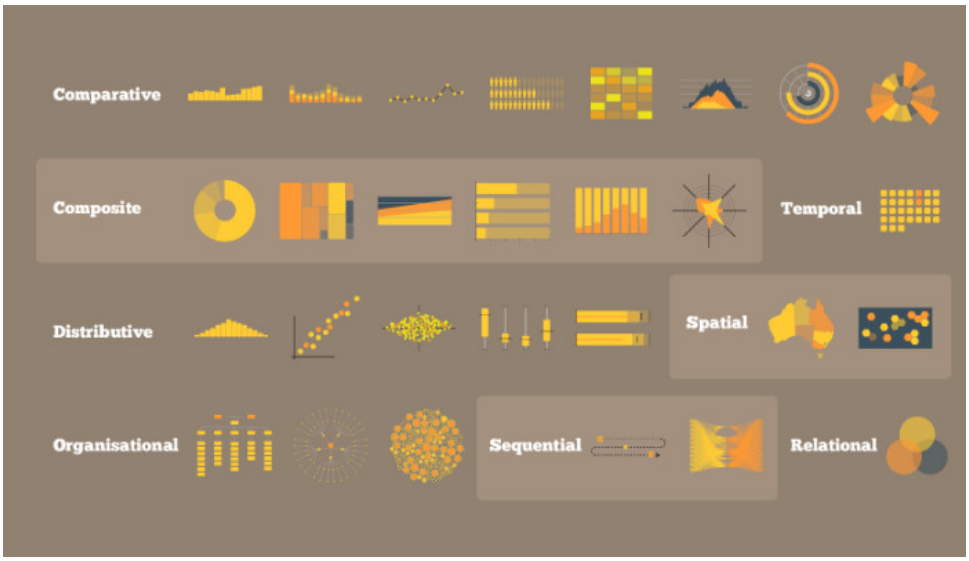
Two-dimensional data visualizations include bar charts, line graphs, pie charts, scatter plots, and histograms (Figure 2.8). Each type of 2D visualization offers unique advantages and is suited to different types of data analysis and presentation. For example, bar charts

are effective for comparing discrete categories, while line graphs are ideal for visualizing trends and patterns over time [54].

In data analysis and communication, 2D visualizations offer several advantages. They provide a clear and concise representation of data that is easy to interpret and understand [58]. Additionally, 2D visualizations can simplify complex datasets and highlight key insights, making them valuable tools for decision-making processes and data-driven decision-making [54].

However, 2D visualizations also have limitations. While they are versatile and widely used, they may struggle to represent multidimensional data or complex relationships effectively. In some cases, 2D visualizations may oversimplify data or obscure important details, leading to potential misinterpretation [58].

Recent advancements in technology have paved the way for innovative approaches to 2D data visualization. Interactive and dynamic 2D visualizations, for example, allow users to explore data in real-time and customize their viewing experience [55]. Furthermore, advancements in data visualization software and programming libraries have democratized the creation of 2D visualizations, making them more accessible to a wider audience [55].



Source: https://www.linkedin.com/pulse/15-most-common-types-data-visualisation-john-otto-ottinger

Figure 2.8: 2D Visualizations Examples

2.4.1.2 3D Visualization

Three-dimensional (3D) data visualizations are graphical representations that incorporate depth perception along with height and width dimensions, allowing for the depiction of spatial relationships and complex data structures. Unlike their two-dimensional counterparts, 3D visualizations provide an immersive experience that closely mirrors the physical world, enabling users to explore data from multiple perspectives [59]. The concept of 3D data visualization has evolved alongside advancements in computer graphics and visualization techniques, offering innovative solutions for analyzing and presenting multidimensional datasets [60].

Several types of 3D visualizations are commonly used across various domains to represent complex data. 3D scatter plots visualize data points in three-dimensional space, allowing for the examination of relationships between multiple variables [61]. Surface plots depict mathematical functions or empirical data on a three-dimensional surface, enabling the visualization of spatial patterns and gradients [62]. Volumetric rendering techniques reconstruct three-dimensional structures from volumetric data, such as medical imaging scans, enabling detailed exploration of internal anatomical structures [63]. Additionally, 3D bar charts represent categorical data using three dimensions, offering a visually compelling way to compare values across multiple categories [61].

Despite their advantages, 3D visualizations also present challenges and limitations. Occlusion, or the obstruction of objects by others, can occur in complex 3D scenes, making it difficult to discern all elements simultaneously [64]. Clutter may arise when visualizing large datasets, leading to information overload and diminished clarity [64]. Moreover, the interpretation of 3D visualizations can be subjective, as users may perceive depth and spatial relationships differently. Addressing these challenges requires careful design and optimization of 3D visualization techniques [65].

Recent advancements in technology have fueled the development of innovative approaches to 3D data visualization. Interactive and immersive 3D visualization tools enable users to manipulate and explore data in real-time, fostering deeper insights and understanding [66]. Virtual reality (VR) and augmented reality (AR) technologies offer new avenues for immersive data exploration, allowing users to interact with 3D visualizations in immersive environments [67]. Additionally, advancements in rendering techniques and hardware capabilities continue to push the boundaries of what is possible in

3D data visualization, paving the way for more sophisticated and impactful applications [68].

2.4.2 Biological Visualization

Visualizations have played a pivotal role in advancing the field of biology, offering researchers powerful tools to explore, analyze, and communicate complex biological phenomena. The integration of visualizations into biology can be traced back to early scientific illustrations and anatomical drawings, which provided researchers with visual representations of biological structures and processes. These visualizations served as essential tools for documenting observations, understanding anatomical features, and disseminating scientific knowledge [69].

Over time, technological advancements have revolutionized biological visualizations, enabling researchers to probe deeper into the intricacies of living organisms and biological systems. The emergence of microscopy techniques, such as light microscopy and electron microscopy, allowed scientists to visualize cellular structures and subcellular components with unprecedented clarity and resolution [70]. These advances facilitated groundbreaking discoveries in cell biology, microbiology, and molecular biology, driving our understanding of fundamental biological processes [71].

The significance of biological visualizations extends beyond mere observation; they serve as indispensable tools for hypothesis generation, experimentation, and data interpretation. Visualization techniques, such as molecular modeling and structural biology, enable researchers to visualize molecular interactions, protein structures, and DNA sequences, aiding in drug discovery, molecular biology research, and bio-informatics [59]. Furthermore, advances in bio-imaging technologies, including fluorescence imaging and live-cell imaging, allow researchers to observe dynamic biological processes in real-time, providing insights into cellular dynamics and physiological functions [72].

In the current landscape, biological visualizations continue to evolve rapidly, driven by advancements in imaging technologies, computational methods, and data analysis techniques. High-throughput imaging platforms, such as confocal microscopy and super-resolution microscopy, enable the acquisition of large-scale, high-resolution datasets, challenging researchers to develop innovative approaches for data visualization and analysis

[73]. Moreover, the integration of bio-informatics tools and computational modeling techniques has facilitated the visualization of complex biological networks, genomic data, and evolutionary relationships, fostering interdisciplinary collaborations and driving discoveries in systems biology and computational biology [74].

In summary, biological visualizations have been instrumental in shaping our understanding of the living world, from the microscopic realm of cells and molecules to the macroscopic complexity of ecosystems and biodiversity. As technology continues to advance and interdisciplinary collaborations flourish, the future of biological visualizations holds immense promise for unraveling the mysteries of life and driving transformative discoveries in the biological sciences [60].

2.4.3 Virtual Reality Visualization

2.4.3.1 Introduction to VR Visualizations

Virtual reality (VR) technology has ushered in a new era of data visualization, offering unprecedented opportunities to explore and interact with complex datasets in immersive environments. Unlike traditional visualization methods, VR enables users to experience data in three dimensions, providing depth and spatial context that enhance understanding and insight [75].

2.4.3.2 Applications of VR Visualizations

The current state of VR visualizations encompasses a diverse array of applications across various domains, including scientific research, engineering, healthcare, education, and entertainment. For example, in scientific research, VR visualization platforms such as CAVE systems and immersive virtual environments have been employed to visualize and analyze complex datasets in fields such as molecular biology, astronomy, and neuroscience [76]. In engineering, VR visualization tools are utilized for product design, simulation, and prototyping, allowing engineers to visualize and manipulate 3D models with greater precision and interactivity [56]. In healthcare, VR-based medical imaging platforms enable physicians and surgeons to visualize patient-specific anatomical structures and medical data for surgical planning and medical education [10] . These examples underscore the transformative impact of VR visualizations across diverse domains, driving advancements in research, education, and industry [73].

2.4.3.3 VR for Large Datasets

The advent of virtual reality (VR) technology has significantly impacted the visualization of large datasets, offering solutions to the inherent challenges posed by the scale and complexity of modern datasets. Traditional two-dimensional visualizations often struggle to effectively convey the richness of information contained in large datasets. VR provides a compelling alternative by immersing users in a three-dimensional space, allowing them to navigate and explore vast datasets seamlessly. This spatial representation not only enhances the comprehension of intricate details within the data but also enables the identification of patterns and correlations that might be overlooked in conventional visualizations [60]. Researchers are increasingly turning to VR to address the visualization needs of large-scale datasets, whether in fields such as genomics, climate modeling, or financial analytics [71]. The synergy between large dataset visualization and VR technology holds the potential to unlock valuable insights and drive innovations across various disciplines [64].

2.4.3.4 VR Visualization Techniques

Various techniques are employed in virtual reality (VR) visualization to represent and interact with data in immersive environments. Surface rendering techniques focus on representing the surfaces of objects or datasets in VR environments. These techniques include polygonal mesh rendering and point cloud rendering, which are commonly used in architectural visualization, geological modeling, and molecular modeling applications [61]. Surface rendering allows users to interactively manipulate and explore surface details with high fidelity and realism.

Dynamic data visualization techniques are crucial for representing time-varying or streaming data in VR environments. Techniques such as particle systems, flow visualization, and temporal interpolation enable users to observe and analyze changes in data over time, facilitating the exploration of dynamic phenomena in fields such as fluid dynamics, climate modeling, and financial markets [70].

Multimodal visualization techniques integrate multiple data modalities, such as visual, auditory, and haptic feedback, to enhance the immersive experience and convey information more effectively to users. By combining different sensory modalities, multi-

modal visualization techniques cater to users with diverse preferences and accessibility needs, enabling a richer and more engaging VR experience [74].

Finally, volume rendering stands as a cornerstone technique in VR visualization, allowing users to visualize volumetric data, such as medical images or scientific simulations, by rendering translucent three-dimensional structures directly from the data. Volume rendering techniques employ methods such as ray casting or texture-based rendering, enabling users to explore complex internal structures and spatial relationships within the data [72].

In this thesis, various volume rendering techniques are utilized to bring the provided biological data into the virtual environment. These techniques allow for the visualizations to be configured in real time tailored to the needs of each dataset.

2.4.3.5 Challenges and Future Directions

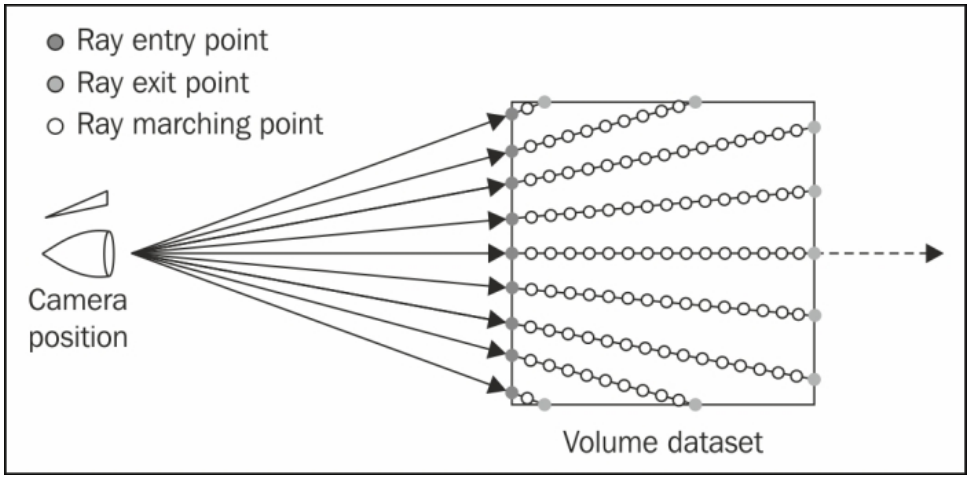
Despite its promising capabilities, visualizing large datasets in virtual reality (VR) environments presents several challenges and limitations. One major challenge is the processing and rendering of massive amounts of data in real-time, which requires robust computational resources and efficient rendering algorithms. The complexity of large datasets can strain system resources, leading to performance bottlenecks and latency issues that hinder the user experience [56]. Additionally, maintaining data integrity and preserving interactivity while navigating through voluminous datasets remain ongoing concerns. Another challenge lies in the design and implementation of user interfaces that facilitate intuitive interactions with large datasets in VR environments. Ensuring that users can effectively explore and analyze data while avoiding cognitive overload and spatial disorientation is a crucial aspect of VR visualization design [73]. Addressing these challenges requires interdisciplinary collaboration among computer scientists, data analysts, human-computer interaction specialists, and domain experts to develop scalable, efficient, and user-friendly VR visualization solutions [76].

2.5 Volume Rendering

2.5.1 Introduction to Volume Rendering

Volume rendering is a sophisticated technique used in computer graphics to visualize complex three-dimensional (3D) volumetric data. Unlike traditional 2D imaging methods that represent objects as surfaces or edges, volume rendering enables the visualization of data throughout a volume, capturing the intricate details within [77]. This method is particularly important for data that cannot be easily reduced to simple surface representations, such as medical imaging data or scientific simulations [78].

The core idea behind volume rendering is to project 3D data onto a 2D plane in a way that preserves the depth, density, and internal structures of the data. This is achieved by using various algorithms that handle the data directly without first converting it into geometric shapes like polygons or meshes [79]. Techniques such as ray marching (Figure 2.9), where rays are projected through the volume and accumulate color and opacity based on the voxel values they encounter, are fundamental to this process [80]. The result is a detailed and comprehensive image that conveys more information than surface renderings, albeit with increased computational complexity.



Source: https://www.oreilly.com/library/view/opengl-build/9781788296724/ch07s03.html

Figure 2.9: Ray Marching technique

Volume rendering has a wide range of applications across multiple fields due to its

ability to provide detailed visualizations of volumetric data.

- In medical imaging, it is extensively used to visualize data from MRI, CT, and PET scans, allowing clinicians to see inside the human body in 3D, which aids in diagnosis, treatment planning, and surgical simulations [10, 7]. This technique provides detailed views of tumors, blood vessels, and internal organs, which are crucial for accurate medical assessments.

- In scientific research, volume rendering helps visualize data from simulations and experiments, including computational fluid dynamics, meteorological data, and astrophysical phenomena . By rendering these complex data sets in 3D, scientists can better understand fluid flows, weather patterns, and the structure of the universe [77].

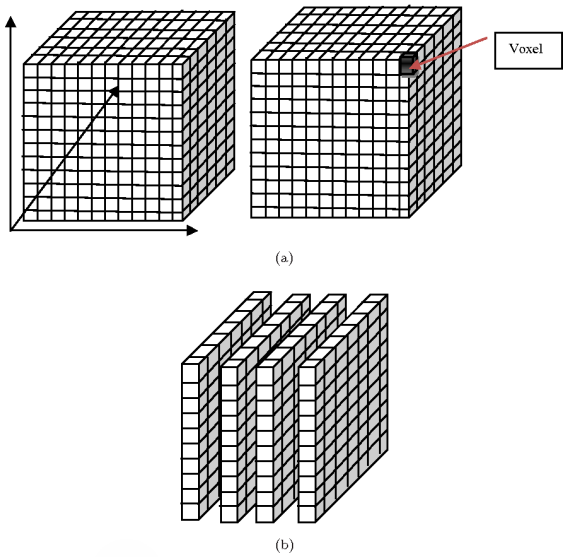
- In the industrial sector, volume rendering is applied in non-destructive testing and inspection, where techniques like industrial CT scans use volume rendering to inspect the internal structures of materials and mechanical parts without damaging them, making it useful for quality control and failure analysis [81].

- Geologists utilize volume rendering to visualize seismic data and underground structures, aiding in the exploration and analysis of natural resources such as oil, gas, and minerals. This 3D view of subsurface formations assists in making informed decisions in resource extraction [82].

- In the entertainment industry, volume rendering is employed to create realistic effects like clouds, smoke, and fire in movies and video games, effectively representing amorphous phenomena [83]. Overall, volume rendering enhances our ability to visualize and interpret complex 3D data, providing invaluable insights across various domains.

2.5.2 Volumetric Data

Volumetric data refers to data that exists in three dimensions, often captured through methods like sampling, simulation, or modeling techniques. These data sets consist of a collection of values, known as voxels, that represent the value of a particular property at specific coordinates within a 3D space (Figure 2.10). This property could be binary,



(a) 3D volume dataset and (b) slice images.

Source: https://www.semanticscholar.org/paper/Gpu-Accelerated-Interactive-Visualization-of-3D-Kumar-Agrawal/77d38adf7fd453bad591e45f79618e607047fa49

Figure 2.10: Volume Data

multi-valued, or even a vector representing complex characteristics like velocity or color [77].

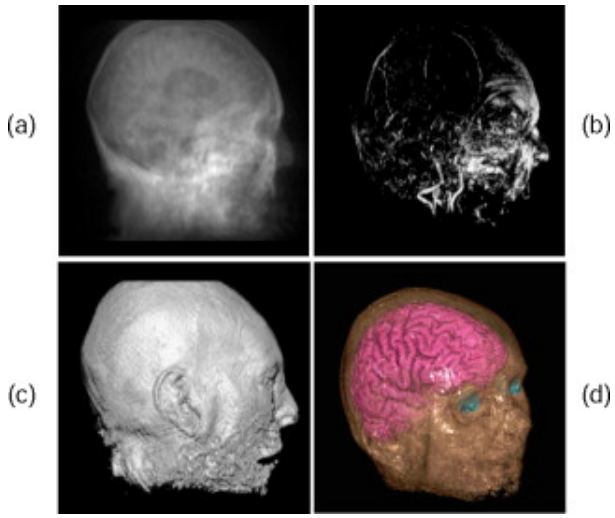
The organization of volumetric data can vary, but it is typically stored in a structured grid format. These grids can be isotropic, with uniform spacing between samples along each axis, or anisotropic, with varying spacing constants. More complex structures include rectilinear grids, where cells are axis-aligned with arbitrary spacing, and curvilinear grids, which are non-linearly transformed while preserving grid topology [79]. Unstructured or irregular grids consist of cells of arbitrary shapes like tetrahedra or hexahedra, requiring explicit specification of their connectivity [77].

One of the fundamental challenges in handling volumetric data is interpolation, which estimates the value of data at non-grid locations. Several interpolation methods exist, ranging from zero-order (nearest neighbor) interpolation, which simply assigns the value of the nearest voxel, to higher-order methods like tri-linear and cubic interpolation that provide smoother transitions between voxel values [84]. These interpolation techniques are crucial for producing visually appealing and accurate renderings of volumetric data,

as they significantly impact the final image quality.

In this thesis, the data used for the visualizations were structured in a grid format. To address the interpolation challenge, cubic interpolation was employed to achieve the most accurate results.

2.5.3 Volume Rendering Techniques



Source: https://www.sciencedirect.com/topics/computer-science/volume-rendering

Figure 2.11: CT head rendered in the four main volume rendering modes: (a) X-ray; (b) MIP; (c) Iso-surface; (d) DVR.

Volume rendering techniques can be broadly categorized into direct volume rendering, iso-surface rendering, and maximum intensity projection (MIP) [77]. These techniques, which are primarily image-order methods, differ in how they handle the volumetric data and the type of images they produce (Figure 2.11). Each technique has its unique advantages and application areas, contributing to the diverse field of volume visualization.

- **Direct Volume Rendering (DVR)**

Direct volume rendering (DVR) is a powerful method that visualizes volumetric data without first extracting geometric surfaces. Instead, DVR directly maps voxel data to

color and opacity values, producing images that represent the entire volume's internal structure. This technique uses the volume rendering integral to accumulate color and opacity along rays cast through the volume as it shown in Figure 2.9 [77]. Mathematically, the volume rendering integral can be expressed as:

$$I_\lambda(x,r) = \int_0^L C_\lambda(s)\mu(s)exp(-\int_0^s \mu(t)dt)ds$$

Here, $I_\lambda(x,r)$ is the intensity of light of wavelength λ received at point x along direction r, $C_\lambda(s)$ is the color, μ(s) is the mass density (or opacity) at position s along the ray, and L is the length of the ray. The exponential term accounts for the attenuation of light as it passes through the volume.

In practice, this integral is often approximated using discrete sampling and compositing techniques, where rays are cast from the image plane through the volume, accumulating color and opacity values at each sample point. The discrete sum equation is given by:

$$I_\lambda = \sum_{i=0}^{N-1} C_\lambda(s_i)\alpha(s_i)prod_{j=0}^{i-1}(1-\alpha(s_j))$$

where:

$$\alpha(s_i) = 1 - \exp(-\mu(s_i)\Delta s)$$

In this equation I_λ is the light intensity of wavelength λ received at the image plane, $C_\lambda(s_i)$ is the light emission at position s_i, $\alpha(s_i)$ is the opacity at position s_i, $\mu(s_i)$ is the attenuation coefficient at position s_i, Δs is the distance between consecutive sample points.

- **Maximum Intensity Projection (MIP)**

Maximum intensity projection (MIP) is a simpler volume rendering technique that projects the highest intensity value encountered along each ray onto the image plane. This method is particularly useful for highlighting the brightest structures within the volume, such as blood vessels in medical imaging [77]. The MIP operation can be mathematically described as:

$$I_{MIP}(x,r) = max_{s\in[0,L]}f(s)$$

where $I_{MIP}(x, r)$ is the intensity value at point x along direction r, and f(s) is the intensity function of the volumetric data. MIP is computationally efficient because it only requires finding the maximum value along each ray, but it can result in the loss of depth information and may produce less informative images for complex structures.

- **Iso-Surface Rendering**

Iso-surface rendering involves extracting a surface within the volume where the data values equal a specified iso-value v_{iso}. This technique is useful for visualizing boundaries within the volume, such as anatomical structures in medical imaging [77]. The iso-surface is defined by the function:

$$B(v) = \begin{cases} 1 & \text{if } v \geq v_{\text{iso}} \\ 0 & \text{otherwise} \end{cases}$$

Common algorithms for iso-surface extraction include Marching Cubes, which approximates the surface using triangles. The algorithm evaluates the function at the corners of each grid cell and interpolates the positions where the iso-surface intersects the cell edges. The vertex positions are determined using linear interpolation:

$$p = p_1 + \frac{v_{iso} - v_1}{v_2 - v_1}(p_2 - p_1)$$

where p_1 and p_2 are the positions of the grid points, and v_1 and v_2 are their corresponding values. This results in a polygonal mesh representing the iso-surface, which can be rendered using standard graphics techniques.

In summary, DVR, MIP, and Iso-Surface Rendering each provide unique methods for visualizing volumetric data. DVR captures the full complexity of the volume by simulating light transport, MIP highlights the brightest structures, and Iso-Surface Rendering extracts and displays specific surfaces within the volume. Each technique offers distinct advantages depending on the application and the type of information that needs to be visualized.

In this thesis, both Direct Volume Rendering and Maximum Intensity Projection techniques were fully implemented to produce accurate visualizations of the data. The Iso-Surface rendering technique was initially implemented and tested but was ultimately removed. This decision was due to the fact that, except for the low-quality datasets, the 3Dtexture required to represent them exceeded the size that the Unity Game engine can handle. The Unity Game engine has a maximum 3Dtexture size limit of 2GB.

2.5.4 Classification and Transfer Functions

Classification and transfer functions are essential components in volume rendering, playing a pivotal role in transforming raw volumetric data into meaningful visual representations. The primary objective of classification is to differentiate various structures within the volumetric data by assigning distinct color and opacity values to different voxel values. This is accomplished through transfer functions, which map data values to visual properties such as color and transparency. By carefully designing these functions, users can emphasize areas of interest, highlight specific features, and suppress irrelevant or occluding structures, thereby enhancing the overall interpretability of the data [85].

Transfer functions typically operate in a multi-dimensional space, considering not only the voxel intensity but also additional attributes such as gradients. The most common approach is the use of 1D transfer functions, where a single scalar value (the voxel intensity) is mapped to color and opacity. However, more sophisticated methods employ 2D or higher-dimensional transfer functions that also take into account the gradient magnitude [86]. This allows for more nuanced classifications, such as distinguishing between smooth regions and regions with sharp boundaries, which is particularly useful in medical imaging where it is important to differentiate between different tissue types [87].

A practical implementation of transfer functions involves using graphical editors, where users interactively adjust the mapping of data values to visual properties. These editors often provide tools for creating and manipulating transfer function curves, allowing users to dynamically explore and refine the visual representation of the volume data [86]. For example, in medical imaging, transfer functions can be adjusted to selectively enhance the visibility of bone structures, soft tissues, or blood vessels within a single dataset, providing a comprehensive view of the anatomy [87].

In summary, classification and transfer functions are fundamental to effective volume rendering. They enable the selective highlighting of important features within volumetric data, providing clear and informative visualizations. By leveraging multi-dimensional mappings and interactive editors, users can achieve precise control over the visualization process, facilitating deeper insights and better decision-making across various applications, from medical imaging to scientific research and industrial inspection [77].

In this thesis, a graphical editor was developed to facilitate the configuration, saving, loading, and editing of transfer functions for use with the direct volume rendering

algorithm. This tool enables real-time adjustments to the transfer functions, thereby simplifying the precise adjustment of visual representations of datasets.

2.6 Game Engines

Game engines serve as software frameworks designed to simplify the process of game development by providing developers with a suite of tools, libraries, and functionalities to create interactive digital experiences [88].

Historically, the concept of game engines traces back to the early days of video game development in the 1970s and 1980s. During this period, developers programmed games from scratch, writing code to handle graphics rendering, physics simulation, input processing, and other essential functionalities. However, as game development became increasingly complex and resource-intensive, the need for more efficient development tools became apparent [89].

The evolution of game engines gained momentum in the 1990s with the emergence of 3D graphics technology and the growing popularity of personal computers and gaming consoles. Game developers began to adopt proprietary engines developed by major gaming companies like id Software, Epic Games, and Valve Corporation to create groundbreaking titles such as Doom, Quake, and Half-Life [90].

The late 1990s and early 2000s witnessed the commercialization and democratization of game engines, with the release of licensed engines like Unreal Engine and CryEngine to third-party developers. These engines offered comprehensive toolsets, robust rendering pipelines, and powerful scripting languages, empowering developers to create high-quality games with reduced development time and resources.

Over the years, game engines have evolved to become versatile platforms not only for traditional game development but also for a wide range of interactive applications, including virtual reality, augmented reality, simulations, and multimedia experiences. Modern game engines boast advanced features such as real-time rendering, physics simulation, artificial intelligence, networking, and cross-platform compatibility, making them indispensable tools for developers across various industries [91, 92].

In summary, game engines represent a pivotal component of contemporary digital interactive experiences, providing developers with the tools and resources needed to bring their creative visions to life. The historical evolution of game engines reflects the ongoing

advancements in technology and the growing demand for immersive and interactive digital content across diverse platforms and industries [93].

2.6.1 Unity

Unity is renowned for its versatility, offering developers the flexibility to create apps that seamlessly transition between 2D, 3D, and virtual reality (VR) environments. This adaptability enables developers to start with a 2D app and later integrate 3D elements or VR functionality as the project evolves. Such flexibility is invaluable, allowing developers to explore different dimensions and interaction paradigms within the same development environment [94]. With Unity, developers can leverage a single codebase and asset pipeline to build apps that cater to diverse user experiences and preferences, from traditional 2D interfaces to immersive 3D worlds and VR simulations.

Unity prioritizes ease of use and accessibility, making it suitable for developers of all skill levels. Its intuitive user interface and visual editor empower developers to bring their creative ideas to life quickly and efficiently, without the need for extensive programming knowledge. The drag-and-drop functionality, combined with robust scripting support, streamlines the development process and enables developers to focus on crafting engaging app experiences. Moreover, Unity's comprehensive documentation and active community support provide developers with the resources they need to overcome challenges and optimize their app development workflows.

Unity boasts a rich feature set and a wide range of customization options, empowering developers to create immersive and feature-rich apps tailored to their specific requirements. From advanced graphics and rendering capabilities to physics simulations, audio effects, and animation tools, Unity offers a comprehensive suite of features that enable developers to push the boundaries of app development. Its extensible architecture and support for third-party plugins and assets provide developers with the flexibility to integrate additional functionality and expand the capabilities of their apps as needed [95, 96].

Chapter 3

Technological Background and Definitions

3.1 Introduction

Chapter 3 serves as a comprehensive exploration of the technological background and essential definitions crucial to understanding the development of our virtual reality (VR) application. This chapter lays the groundwork for our research journey by introducing key concepts, software tools, and hardware components integral to the creation and deployment of our VR application for visualizing biological data. Throughout Chapter 3, we will delve into various aspects, including data formats, software tools like ImageJ and Python, the Unity game development engine, graphics pipelines, VR structure in Unity, and hardware components such as the Pimax 8K X headset and Valve index Controllers.

3.2 Data Formats

In the context of the project, the biological data initially provided, arrived in a format unsuitable for direct integration into the virtual reality (VR) visualization application. The nature of the data, structured within the Hierarchical Data Format (HDF5), presented challenges in its direct utilization for volume rendering implementation purposes. HDF5, while efficient for organizing and storing large volumes of complex data, required preprocessing and conversion to a format more compatible with the application's require-

ments. Consequently, to facilitate seamless integration and effective utilization within the application, a conversion process was necessary to adapt the data into a format better suited for volume rendering. This necessitated the conversion of the original HDF5 format to the Nearly Raw Raster Data (NRRD) format, which offers advantages in terms of compatibility and efficiency within the application's volume rendering framework.

3.2.1 Hierarchical Data Format

Hierarchical Data Format version 5 (HDF5) is a data model, library, and file format for storing and managing large and complex datasets [97]. It is designed to address the challenges of handling massive volumes of data efficiently and effectively. HDF5 organizes data into a hierarchical structure, allowing users to organize, manage, and access data in a flexible and scalable manner.

At its core, HDF5 operates as a container format, capable of storing diverse data types, including numeric data, images, text, and metadata. It supports multidimensional arrays, allowing for the representation of complex data structures commonly encountered in scientific and engineering applications. HDF5 files can store vast amounts of data in a single file, making it well-suited for applications that require efficient data storage and retrieval.

One of the key features of HDF5 is its ability to support data compression and chunking, enabling efficient storage and retrieval of large datasets. Data compression reduces the storage footprint of datasets, while chunking allows for the selective loading of data, improving access times for specific regions of interest within a dataset.

HDF5 is widely used in various fields, including scientific research, engineering, finance, and healthcare. In scientific research, HDF5 is commonly used to store experimental data, simulation results, and computational models. In engineering applications, HDF5 is employed to store sensor data, simulation output, and design specifications. In finance, HDF5 is utilized for storing market data, financial models, and risk analysis results. In healthcare, HDF5 is used to store medical images, patient records, and clinical trial data.

The versatility, scalability, and performance of HDF5 make it a preferred choice for managing and analyzing large and complex datasets in diverse fields. Its open-source

nature, robust documentation, and support for various programming languages, including C, Python, MATLAB, and Java, further contribute to its widespread adoption and popularity in the scientific and engineering communities.

3.2.2 Nearly Raw Raster Data

Nearly Raw Raster Data (NRRD) is a file format designed for the storage and exchange of n-dimensional raster data. It was developed to address the need for a simple, flexible, and human-readable format that could accommodate a wide range of scientific and medical imaging data.

NRRD files consist of a concise header section containing metadata describing the data dimensions, types, and other properties, followed by the raw data itself. The header is written in plain text, making it easy to inspect and modify using standard text editors or command-line tools. This simplicity and transparency make NRRD well-suited for both manual inspection and automated processing workflows.

The NRRD format supports various data types, including scalar, vector, and tensor data, allowing it to represent a diverse array of image and volume datasets. Additionally, NRRD files can store metadata such as physical units, axis orientations, and spacing information, providing essential context for the interpretation and analysis of the data.

One of the key advantages of the NRRD format is its versatility and extensibility. NRRD files can encode not only traditional two-dimensional images but also higher-dimensional datasets such as volumetric images, time-series data, and multi-channel images. This flexibility makes NRRD suitable for a wide range of applications, including medical imaging, scientific research, computational biology, and computer graphics.

Overall, the NRRD format offers a balance of simplicity, flexibility, and expressiveness, making it a valuable tool for data management, analysis, and visualization in scientific and medical applications. Its open and well-documented specification, along with robust support in various software libraries and tools, further enhances its utility and accessibility in the research community.

In this thesis, both file formats mentioned above were utilized. Initially, the dataset was in the HDF5 format, which was subsequently converted to the NRRD format to facilitate processing and loading into the Unity Game Engine, aligning more effectively with the objectives of this work.

3.3 Software tools

3.3.1 ImageJ

Fiji suite, a distribution of ImageJ was used as a tool to explore the contents of the original HDF5 file. ImageJ, a widely-used open-source image processing program, provides functionality for visualizing and analyzing multidimensional images [98]. Leveraging the capabilities of the Fiji distribution, utilizing ImageJ to open the original HDF5 file, allowed the exploration of its contents and the visualization of discrete images comprising each volume.

3.3.2 HDFview

HDFview serves as a user-friendly program designed to open HDF files in a manner akin to browsing through a file directory [99]. By providing a visual interface for navigating the contents of an HDF file, it enables users to explore the internal structure and organization of the data. This capability was essential for comprehending the intricacies of the file and determining the most effective approach for decomposing its contents accurately. This functionality was useful for understanding the complexities of the file and identifying a correct method for accurately decomposing its contents.

3.3.3 Python

Python is a widely adopted programming language known for its versatility and extensive library support, particularly in the field of image processing [100]. With its robust ecosystem, Python offers a plethora of libraries tailored for array manipulation, including multidimensional arrays commonly encountered in image processing tasks. One notable library is H5Py, which provides functionalities to interact with HDF5 files, enabling users to open and read HDF files seamlessly through code. Additionally, Python offers the PyNRRD library, which, when combined with the NumPy library, equips developers with all the tools for creating and NRRD file. Effectively Python offers everything needed for creating an HDFtoNRRD file converter.

In this thesis, ImageJ and HDFview software tools were employed initially to examine the dataset. Subsequently, a Python script was utilized for converting the dataset from

HDF format to NRRD format.

3.4 Graphics Pipelines

The graphics pipeline is a fundamental component of modern computer graphics systems, responsible for transforming geometric data into rendered images displayed on screen [101]. It consists of a series of stages or steps, each handling specific tasks such as geometry processing, rasterization, shading, and rendering. The purpose of the graphics pipeline is to efficiently process and manipulate graphical data, ultimately producing visual outputs that accurately represent the virtual scene being rendered. By breaking down the rendering process into distinct stages, the graphics pipeline allows for parallelization and optimization of tasks, leading to improved performance and visual fidelity in real-time rendering applications such as video games, simulations, and virtual reality experiences.

The graphics pipeline typically consists of several common stages that are essential for generating rendered images. These stages include geometry processing, vertex shading, primitive assembly, rasterization, fragment shading, and frame buffer operations.

- In the geometry processing stage, raw geometric data such as vertices and polygons are transformed and manipulated to define the shape and position of objects in the virtual scene.
- Vertex shading involves applying transformations and computing vertex attributes such as color and texture coordinates.
- The primitive assembly stage organizes vertices into geometric primitives such as triangles or lines for further processing.
- Rasterization converts these primitives into pixels on the screen, determining which pixels are covered by the primitives and their corresponding attributes.
- Fragment shading calculates the final color and other properties of each pixel, taking into account lighting, materials, and textures.
- Finally, frame buffer operations involve writing the computed pixel values to the frame buffer for display.

Each stage of the graphics pipeline plays a crucial role in the overall rendering process, contributing to the creation of visually compelling and immersive graphics.

3.4.1 Unity Supported Pipelines

In Unity, various rendering pipelines determine how graphics are processed and displayed in a scene [102]. Unity provides support for multiple rendering pipelines, each offering different features, performance characteristics, and compatibility with different platforms. The built-in rendering pipelines include the Built-In Render Pipeline (also known as the Legacy Render Pipeline), the Universal Render Pipeline (URP), and the High Definition Render Pipeline (HDRP). The Built-In Render Pipeline is a straightforward and versatile pipeline suitable for rendering 2D and 3D graphics across a wide range of platforms, with support for features such as dynamic lighting, shadows, and post-processing effects. The Universal Render Pipeline (URP) is a lightweight and customizable rendering solution optimized for performance and compatibility with a broad range of devices, including mobile and low-end hardware. It offers features such as Shader Graph support, improved rendering quality, and built-in support for forward and deferred rendering. The High Definition Render Pipeline (HDRP) is a high-fidelity rendering pipeline designed for creating realistic and visually stunning graphics in high-end applications. It offers advanced features such as physically based rendering, high-quality lighting and shadows, volumetric effects, and advanced post-processing effects. Each rendering pipeline in Unity caters to different project requirements and target platforms, allowing developers to choose the most suitable pipeline for their specific needs.

In this thesis, the default Built-In Render Pipeline (Legacy Render Pipeline) was employed. While the Universal Render Pipeline (URP) offers lightweight and versatile rendering solutions, it was deemed unsuitable for this project due to lower quality results. Initially considered, the High Definition Render Pipeline (HDRP) offers numerous advantages in quality and fidelity of results but imposes significantly higher demands on computing resources, ultimately leading to compromised overall performance.

3.5 Basic Structure of Unity

Unity, as a game engine and development platform, provides a comprehensive set of tools and features for creating interactive 3D experiences, including virtual reality (VR) applications [103]. Understanding the basic structure of Unity is essential for navigating the development environment and effectively building VR projects. Unity projects are organized into several key components. Each game or application created in Unity is called a Project. Each Project consists of the used Assets and one or more Scenes. The Scenes consists of GameObjects and Prefabs, each of which has one of more Components and Scripts attached to it.

3.5.1 Assets

Assets in Unity refer to the various files and resources used in creating a project. These can include 3D models, textures, audio clips, animations, scripts, and more. Assets serve as the building blocks of the virtual environment, providing the visual, auditory, and interactive elements necessary for the project.

3.5.2 Scenes

Scenes in Unity represent different levels or environments within a project. Each scene contains the objects, characters, and elements that make up that particular part of the game or application. Scenes are used to organize and structure the project, allowing developers to focus on specific areas of the virtual world during the design and development process.

3.5.3 GameObjects

GameObjects are the fundamental objects in Unity's scene hierarchy. They can represent characters, props, lights, cameras, and other elements within the virtual environment. GameObjects can be customized and manipulated using components to define their appearance, behavior, and interactions.

3.5.4 Components

Components are reusable pieces of functionality that can be attached to GameObjects to add specific behaviors or properties. Examples of components include colliders for detecting collisions, rigidbodies for simulating physics, scripts for defining custom behaviors, and audio sources for playing sounds. Components allow developers to extend the functionality of GameObjects without having to write code from scratch.

3.5.5 Scripts

Scripts in Unity are written in programming languages such as C# or UnityScript (JavaScript) and are used to define custom behaviors and functionality for GameObjects and other elements in the project. Scripts can access and modify the properties of GameObjects, respond to user input, implement application logic, and interact with other components and systems within the Unity environment. They provide a flexible and powerful way to create dynamic and interactive experiences in Unity projects.

3.6 VR Structure in Unity

In Unity, transitioning a desktop project into a virtual reality (VR) experience involves integrating specialized libraries, tools, and components tailored for VR interaction [104]. Unlike traditional desktop applications, VR projects require additional elements to enable immersive experiences and seamless interaction within virtual environments. These components include VR-specific libraries and tools designed to handle input from VR devices, manage VR rendering and performance, and facilitate spatial interactions in 3D space.

To transform a desktop project into a VR experience, developers need to incorporate VR-specific components that cater to the unique requirements of VR environments. These components manage various aspects of VR integration, such as headset tracking, hand controllers, user interfaces optimized for VR displays, and interaction systems tailored for immersive interactions. Additionally, VR projects often leverage specialized rendering techniques and optimizations to ensure smooth performance and high fidelity in VR simulations.

In this thesis, the virtual environment was constructed using a combination of the systems, plugins, and frameworks that are analysed below.

3.6.1 XR System

The XR System in Unity, short for Cross-Reality System, is a framework designed to facilitate the development of immersive experiences across various reality platforms, including virtual reality (VR), augmented reality (AR), and mixed reality (MR) [105]. This system provides developers with a unified interface for building applications that can seamlessly transition between different reality modes, offering users a versatile and engaging experience. One of the key advantages of the XR System is its ability to abstract the underlying hardware complexities, allowing developers to focus on creating compelling content without worrying about platform-specific optimizations. However, while the XR System simplifies cross-reality development, it may introduce some overhead and compatibility challenges, particularly when targeting multiple reality platforms with varying capabilities and requirements.

3.6.2 XR Origin

The XR Origin is a critical component in transitioning a virtual camera from desktop use to VR use. Essentially, it serves as the anchor point or reference frame for all XR (Extended Reality) interactions within a Unity project. In the context of VR, the XR Origin defines the position and orientation of the virtual camera relative to the physical world or play space. This transformation is crucial for ensuring that the user's movements in the real world are accurately reflected within the virtual environment, enhancing immersion and minimizing motion sickness. By aligning the XR Origin with the user's physical location and orientation, developers can create seamless VR experiences that respond naturally to the user's movements and interactions.

3.6.3 XR Interaction ToolKit

The XR Interaction System within Unity serves as a framework crafted to streamline the development process of interactive VR experiences [106]. Tailored specifically for VR applications, this system equips developers with a diverse array of tools and components

essential for implementing a wide spectrum of VR interactions. From fundamental tasks like object manipulation and teleportation to more intricate interactions such as navigating menus and initiating animations, the XR Interaction System offers a versatile toolkit. Developers can seamlessly integrate features like object interaction, hand gesture recognition, locomotion mechanics, and menu navigation into their VR projects using readily available prefabs, scripts, and other resources provided by the system.

3.6.4 XR Plug-in Management

XR Plug-in Management in Unity refers to the system that allows developers to manage and integrate external XR (Extended Reality) plug-ins into their Unity projects. These plug-ins enable Unity applications to support various VR, AR, and MR (Mixed Reality) devices and platforms beyond the default capabilities provided by Unity's built-in XR framework.

With XR Plug-in Management, developers can easily switch between different XR plug-ins, depending on the target platform or device they are developing for. This flexibility allows for broader compatibility and support across a wide range of XR hardware and software ecosystems.

Additionally, XR Plug-in Management provides features for configuring and optimizing XR settings, such as input mappings, rendering configurations, and performance optimizations. This ensures that Unity applications can deliver optimal XR experiences tailored to the specific requirements of different XR devices and platforms.

3.6.5 Open XR

OpenXR is an open standard created by the Khronos Group [107], aimed at unifying the development of virtual reality (VR) and augmented reality (AR) applications across multiple hardware platforms and software ecosystems. This standard provides a common API (Application Programming Interface) that enables developers to write XR applications once and deploy them across a wide range of XR devices and platforms seamlessly. By adopting OpenXR, developers can create XR applications that are compatible with various XR hardware, including VR headsets, AR glasses, and other immersive devices, without the need for extensive platform-specific code modifications. OpenXR offers a unified approach to handling XR features such as tracking, input handling, rendering, and

device interactions, abstracting away the differences between different XR platforms and devices. Furthermore, OpenXR promotes interoperability and collaboration within the XR ecosystem by providing a common framework for device manufacturers, platform developers, and content creators. This facilitates the development of a diverse range of XR applications and ecosystems while ensuring consistent user experiences across different devices and platforms.

3.7 Unity Architecture & Project Structure

In Section 3.7, the fundamental components that contribute to the architecture and project structure of Unity will be explored [103]. These components are crucial in defining the behavior, appearance, and functionality of game objects within Unity projects.

3.7.1 Coordinates - Transform

In Unity, the coordinate system serves as a fundamental framework for positioning and orienting objects within a three-dimensional space. This system is based on a Cartesian coordinate grid, where each axis—X, Y, and Z—represents a different direction in space. The X-axis typically denotes horizontal movement, with positive values extending to the right and negative values to the left. The Y-axis represents vertical movement, with positive values moving upward and negative values downward. Lastly, the Z-axis signifies depth or forward-backward movement, with positive values extending forward and negative values extending backward. Together, these axes form a right-handed coordinate system, where rotations follow the right-hand rule, ensuring consistency in orientation throughout the scene.

The Transform component in Unity serves as an entity for managing the spatial properties of game objects. Attached to every object within the scene, the Transform component encapsulates attributes such as position, rotation, and scale. Through the Transform component, developers can precisely control the placement and orientation of objects within the Unity environment. Position coordinates determine the location of an object in relation to the scene's origin point, while rotation coordinates specify its angular orientation around each axis. Additionally, scale coordinates define the object's size along each axis, allowing for uniform or non-uniform scaling.

3.7.2 Local Space vs World Space Coordinates

In Unity, game objects can have their transformations specified in either local space or world space coordinates. Local space coordinates are relative to the object itself, meaning that the object's position, rotation, and scale are defined with respect to its own axes. This allows for transformations that are independent of the object's parent or the overall scene.

On the other hand, world space coordinates are absolute and defined with respect to the global coordinate system of the scene. In world space, the position, rotation, and scale of objects are determined relative to the origin of the scene and the global axes (X, Y, Z). This means that transformations are affected by the parent-child hierarchy and the position of the object within the scene.

3.7.3 Mesh Component

Mesh component defines the geometry of a game object, specifying its shape and structure. A mesh is composed of vertices, edges, and faces that collectively form the visual representation of the object. Vertices represent the points in 3D space, edges connect pairs of vertices, and faces define the surfaces of the object.

The Mesh component contains information about the vertices, such as their positions, normals (directions perpendicular to the faces), UV coordinates (for texture mapping), and optionally tangents and vertex colors. By manipulating the vertices and other properties of the Mesh component, developers can create a wide variety of shapes and forms for their game objects.

Meshes can be created procedurally using scripts or imported from external modeling software such as Blender or Maya. Unity provides tools for importing and manipulating meshes, including options for optimizing and modifying the geometry to improve performance and visual quality.

3.7.4 Materials, Textures & Shaders

Materials, Textures, and Shaders are integral components of Unity projects, contributing significantly to the visual appearance and rendering effects of game objects.

Materials define the surface properties of objects, including their color, reflectivity, transparency, and shininess. Each material can be assigned different properties to create diverse visual effects. Unity provides a wide range of built-in materials, and developers can also create custom materials tailored to their specific project requirements.

Textures are 2D images applied to the surfaces of objects to add detail and realism. They can be used to simulate various surface characteristics such as wood, metal, fabric, or skin. Textures can be imported into Unity and assigned to materials to enhance the visual fidelity of game objects.

Shaders are programs written in a shading language (such as Cg/HLSL for Unity) that define how light interacts with materials. Shaders determine the rendering behavior of objects, including their lighting, shadows, and reflections. Unity supports both surface shaders and vertex/fragment shaders, offering developers flexibility in creating complex visual effects.

Together, Materials, Textures, and Shaders allow developers to create visually stunning and immersive experiences in Unity projects.

3.7.5 Lighting

In Unity, lighting is managed through various components and settings that control the behavior and appearance of light sources within a scene. These components include:

Light Sources: Unity supports different types of light sources, such as directional lights, point lights, spotlights, and area lights. Each light source has unique properties that affect how it illuminates the scene.

Light Probes: Light probes are used to capture and approximate the lighting conditions in a scene, especially for dynamic objects or characters that move within the environment. Light probes help ensure consistent lighting across different areas of the scene.

Global Illumination (GI): Unity's Global Illumination system simulates the indirect lighting effects caused by bouncing light rays in the environment. GI enhances the realism of a scene by accounting for diffuse inter-reflections and color bleeding between objects.

Lightmaps: Lightmaps are precomputed textures that store the lighting information for static objects in the scene. Lightmaps help improve rendering performance by reducing the computational overhead of real-time lighting calculations.

Light Cookies: Light cookies are textures that can be applied to light sources to shape or modify the appearance of the light emitted. This allows developers to create custom lighting effects, such as casting patterns or projecting textures.

Light Settings: Unity provides a range of settings and parameters for fine-tuning the lighting in a scene, including intensity, color, range, and attenuation. These settings allow developers to achieve the desired visual effects and mood for their projects.

3.7.6 User Interface (UI)

In Unity, UI components include elements such as buttons, sliders, text fields, images, and panels. These elements can be arranged and styled to create custom user interfaces tailored to the specific requirements of the application.

UI components are typically created and managed using Unity's UI system, which provides a set of tools and components for designing and implementing user interfaces. Developers can use Unity's built-in UI editor to create and configure UI elements directly within the Unity Editor, allowing for rapid prototyping and iteration. Unity's UI system supports various interaction methods, including mouse, touch, and VR controllers, making it suitable for a wide range of applications across different platforms and devices.

3.8 Pimax 8K X

At the heart of the Pimax 8K X (Figure 3.1) is its remarkable display system, boasting an astonishing 8K resolution (3840x2160 pixels per eye), which is four times the resolution of traditional VR headsets [108]. This ultra-high resolution, combined with a wide field of view (FOV) of up to 170 degrees (Figure 2.7), ensures stunning visual clarity and an expansive view of virtual worlds, surpassing the limitations of conventional VR displays.

The Pimax 8K X also excels in motion tracking and responsiveness, thanks to its advanced tracking system and high refresh rate. With precise motion tracking technology and a refresh rate of up to 90Hz, the headset delivers smooth and accurate movement tracking, minimizing motion sickness and enhancing the sense of presence in VR experiences.

The decision to utilize the Pimax 8K X VR headset for this project was driven by its specifications and performance capabilities. Its display resolution, combined with a

Source: https://www.amazon.com/Virtual-Reality-Headset-Display-Version/dp/B095P5SSMC

Figure 3.1: Pimax 8KX

wide FOV and high refresh rate, promised to deliver high visual fidelity and immersion, essential for creating realistic and engaging biological visualization tools.

Furthermore, the Pimax 8K X's compatibility with popular VR development platforms, such as Unity and SteamVR, streamlined the integration process as well as the implementation of a variety of input methods.

3.8.1 Motion

In virtual reality (VR) environments, motion plays a critical role in creating immersive experiences that mimic real-world interactions. The perception of motion in VR is closely tied to the user's visual and vestibular systems, which are responsible for processing visual cues and maintaining balance, respectively. However, accurately simulating motion in VR presents several challenges and considerations.

Motion sickness, a common issue in VR experiences, arises from discrepancies between visual motion cues and physical movement, leading to discomfort for users. To address this, VR developers employ various locomotion techniques such as teleportation, snap turning, and comfort modes that minimize motion intensity while maintaining immersion. These techniques aim to reduce discomfort without compromising the overall experience.

Accurate motion tracking is fundamental for creating a seamless and responsive VR experience. Tracking systems, which utilize inertial sensors or external cameras, capture

the user's movements in real-time, allowing them to interact naturally with virtual environments. Advanced algorithms interpret these movements to ensure precise interaction, enabling users to navigate digital spaces with precision and interact with virtual objects seamlessly.

3.8.2 Tracking

One of the most common tracking methods used in VR is inside-out tracking, where sensors embedded within the VR headset or controllers detect changes in position and orientation relative to the user's surroundings. This approach offers the advantage of freedom of movement without the need for external sensors or markers. However, inside-out tracking may be susceptible to occlusion issues, where the sensors lose sight of the user's hands or body parts, resulting in tracking inaccuracies.

Alternatively, outside-in tracking relies on external sensors or cameras placed in the environment to track the user's movements. This method provides robust and precise tracking, as the external sensors have a comprehensive view of the user's position from multiple vantage points. However, it requires setting up dedicated tracking hardware and may limit the user's mobility within a predefined tracking area.

Hybrid tracking solutions combine elements of both inside-out and outside-in tracking to leverage their respective strengths while mitigating their weaknesses. For example, some VR systems employ a combination of onboard sensors for initial tracking and external cameras for additional positional data when needed. This hybrid approach aims to achieve a balance between accuracy, mobility, and ease of setup.

In addition to positional tracking, VR systems often incorporate other sensors to capture more nuanced movements, such as hand gestures and facial expressions. These sensors enhance the user's ability to interact with virtual objects and characters in a natural and intuitive manner, further enhancing immersion and presence in the virtual world.

Overall, tracking technology continues to evolve, driven by advancements in sensor technology, computer vision, and machine learning. As tracking systems become more sophisticated and reliable, they play a crucial role in shaping the future of VR by enabling more immersive and interactive experiences. However, addressing challenges such

as occlusion, latency, and power consumption remains a focus of ongoing research and development in the field.

3.8.3 Simulation Sickness

Simulation sickness, also known as motion sickness in virtual environments, is a common challenge faced in virtual reality (VR) experiences. It occurs when there is a disconnect between the motion perceived by the user's visual system and the motion sensed by their vestibular system, leading to symptoms such as nausea, dizziness, and discomfort.

The underlying cause of simulation sickness is often attributed to discrepancies between visual cues and proprioceptive feedback. In VR, users may perceive themselves moving through the virtual environment while remaining stationary in the physical world, leading to sensory conflict and disorientation.

Several factors contribute to simulation sickness, each influencing its likelihood and severity. The speed of movement and acceleration, for instance, plays a crucial role. Rapid movements and sudden changes in speed can trigger motion sickness more easily than gradual movements. Similarly, the degree of control the user has over their virtual avatar or viewpoint can affect their susceptibility to motion sickness. When users lack control or feel disoriented in the virtual environment, they're more likely to experience discomfort.

The duration of VR sessions also matters. Prolonged exposure to virtual environments, especially those involving continuous movement, can increase the risk of simulation sickness. Altitude-related factors, such as virtual heights and depth perception, can contribute to feelings of vertigo and nausea. Additionally, features of the VR display system, including binocular display, field of view, and latency, can impact motion sickness. High latency and lag between user actions and system responses can exacerbate feelings of disconnection and discomfort.

To mitigate simulation sickness, VR developers employ various techniques and design principles. Smooth locomotion methods, such as gradual acceleration and deceleration, can help reduce the sensation of vection (illusory self-motion) and alleviate discomfort. Providing visual aids, such as stationary reference points or virtual cockpits, can also enhance spatial orientation and reduce feelings of nausea.

Moreover, optimizing frame rates, minimizing latency, and ensuring consistent tracking accuracy are essential for maintaining a comfortable and nausea-free VR experience. High-quality displays with low persistence and minimal motion blur can further enhance visual comfort and reduce the risk of motion sickness.

3.9 Valve Index Controller

The Valve Index Controller (Figure 3.2a), developed by Valve Corporation, is a VR input device designed to provide users with precise and intuitive interaction within virtual environments [109]. Offering a wide range of motion tracking capabilities and ergonomic design, the Valve Index Controller stands out as a choice for immersive VR experiences.

The Valve Index Controller boasts a range of hardware specifications that contribute to its performance in VR applications. Featuring precise finger tracking technology, the Valve Index Controller allows users to interact with virtual objects and environments with dexterity and accuracy. Its ergonomic design ensures comfortable use during extended VR sessions, with adjustable straps and intuitive button placement enhancing user comfort and convenience. Additionally, the Valve Index Controller is equipped with high-resolution sensors for precise motion tracking, enabling fluid and natural movement within virtual environments. The controller's ergonomic grip and intuitive button layout provide users with seamless control and interaction in VR experiences, enhancing immersion and usability. Furthermore, the Valve Index Controller offers compatibility with a wide range of VR headsets and platforms, making it a versatile choice for both casual and professional users alike

3.10 Vive Pro SteamVR Base Station 2.0

The Vive Pro SteamVR Base Station 2.0 (Figure 3.2b) is a tracking device utilized in virtual reality (VR) setups to precisely track the position and movement of VR headsets and controllers within a defined space [110]. This base station was chosen due to its compatibility with both the Pimax 8K X headset and the Valve Index controllers, ensuring seamless integration and accurate tracking across all components.

Source: https://www.turbosquid.com/FullPreview/1550515

(a) Valve Index Controllers

Source: https://vortexvr.gr/en/products/valve-index-steam-vr-base-station-2-0-base-station-2-0

(b) Vive Pro SteamVR Base Station 2.0

Figure 3.2: Controllers and Tracking bases

Chapter 4

Users View

4.1 Introduction

This chapter aims to provide an exploration and presentation of the user experience within the application. Developed with the purpose of reconstructing large-scale biological datasets into three-dimensional objects and presenting them inside a virtual space, the app is offering the tools and the environment for a different approach in their examination.

Upon launching the application, users find themselves within an empty room, surrounded only by a gentle light source strategically placed to minimize disorientation. This minimalist environment serves as the backdrop for users to focus on their primary task: loading and examining specimens. At any point during their exploration, users can access the pop-up menu, as detailed later in this chapter, to initiate the loading process for any specimen that has been properly preprocessed.

After the loading process is complete, users gain the ability to interact with the reconstructed object. This interaction includes the freedom to manipulate the object's position within the virtual space, as well as to adjust its visualization parameters to suit their preferences and analytical needs. Users have the ability to adjust various parameters of the reconstructed object's appearance, including color, opacity, spectrum of representation, size, and rotation. To perform these changes, users utilize another pop-up menu that provides them with a variety of options for manipulating the object.

While all these parameters provide ways to interact with the object, it could be a repetitive task to set them all from scratch, especially when studying multiple objects. To

address this challenge, users have been equipped with the ability to save the configuration as a discrete file. These files can be loaded at any moment and applied to the current object, streamlining the analysis process and allowing users to maintain consistency across their observations.

4.2 Controls

Effective control mechanisms are crucial for providing users with a seamless and immersive experience within the virtual reality (VR) environment of the application. The Valve Index Controller serves as the primary input device, offering intuitive and responsive interaction capabilities. Through precise tracking of hand movements and fingers, coupled with ergonomic design considerations, the Valve Index Controller enables the users to navigate the virtual space, interact with objects, and access menus with ease.

The Valve Index Controller provides users with a versatile array of functionalities tailored to enhance their interaction with the virtual environment. Equipped with an array of buttons, triggers, and touch-sensitive surfaces, the controller offers precise input methods for executing various actions within the application. By leveraging these input mechanisms, users can navigate menus, manipulate objects and adjust settings.

4.2.1 Navigation Controls

For navigation controls, the application is primarily designed for a seated position, where users load and observe objects directly in front of them. While this is the intended usage scenario, some navigation controls have been implemented to accommodate users who may wish to explore the space around them. The locomotion system utilized in the application relies on tethering the Head-Mounted Display (HMD) to the base stations, allowing users to move their perspective accordingly. However, it's important to note that this system is limited by the length and flexibility of the cable, as the HMD used in this project is not wireless.

4.2.2 Interaction Controls

The Valve Index Controller provides users with a diverse range of input options, including buttons, triggers, grips, touchpads, joysticks, and spatial tracking capabilities. Within the

application, nearly all of these input methods are utilized to enhance the user experience and facilitate interactions.

The primary method of interaction with the loaded object is through the spatial tracking feature of the controller. This feature allows users to project a ray in front of the controller. By pointing the controller towards the desired object, users can interact with it. This same system is utilized for navigating through user interface elements. Essentially, this spatial tracking feature serves as the virtual environment's equivalent of a mouse cursor.

Continuing with the analogy, while the spatial tracking serves to move the cursor, the trigger button functions akin to a right-click action. It is utilized for selecting objects or executing actions within the virtual environment.

The controller also features a highly effective grip interface. By grasping and gently squeezing the controller, users can trigger the grab action. This allows them to point to the reconstructed object, physically grip the controller, and virtually grab the object. Once grabbed, the object moves along with the ray, maintaining a consistent distance between the hand model and the object. Importantly, when the user stops squeezing the controller, the grab action also ceases, releasing the object in the virtual world.

The joystick can be used during the grab action. While an item is grabbed, vertical motions of the joystick control the distance between the user and the object, enabling users to adjust the proximity as desired. Additionally, horizontal motions of the joystick facilitate rotation of the object, allowing users to rotate it clockwise or counterclockwise to achieve their preferred orientation.

4.2.3 Menu Controls

There are controls dedicated to UI and menu navigation. For instance, pressing down on the joystick activates the menu, causing it to appear above the hand model. The menu panel remains visible in this position until the joystick is pressed again to deactivate it. Menu navigation relies on spatial tracking and the trigger for selecting options. Notably, there is no dedicated button for the "back" action, which is why a back button is provided in every menu tab for easy navigation.

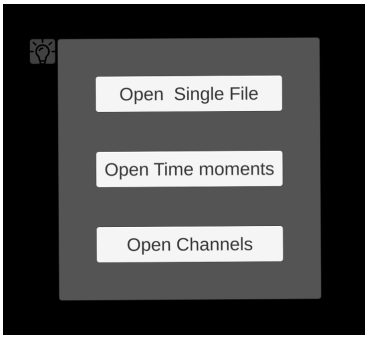
The A button serves the purpose of adding color control points and alpha value points within the transfer function editing interface, as will be elaborated in the subsequent

sections. The B button is used for removing control points in the same sections.

4.3 Open - File

Once the application starts, users are presented with the option to load a dataset, accessible through the menu. There are three different modes for opening files: single file, time moments, and channels (Figure 4.1).

After selecting an option, the virtual file explorer is activated, seamlessly integrated within the application. This feature enables users to explore the file system from within the virtual environment and select the appropriate file or files.



Source: Thesis application

Figure 4.1: Open File menu

4.3.1 Single Instance

The single file loading mode offers users the ability to load a single object for observation, editing, and manipulation. This mode capitalizes on the advantage of efficiently utilizing hardware resources, ensuring maximum quality data reconstruction, as only one file is loaded at a time.

Upon selecting this option, users are directed to the file explorer interface to navigate to their desired file. During navigation, only folders and reconstructable files are visible, while other non-reconstructable files remain hidden from view.

It's important to note that for the application to visualize a dataset, it must be in the NRRD format.

Once the file is selected, the application initiates the data loading process. The duration of this process varies depending on factors such as the size and quality of the file, as well as the specifications of the hardware components (RAM, GPU, CPU). During this loading period, the application remains inaccessible to users until the process is completed.

Upon completion of the loading process, users regain control of the application and are presented with the reconstructed object in front of them. However, it's worth men-

tioning that the visual appearance of the object may not be accurate at this stage, as the rendering mode and transfer function have not yet been configured.

4.3.2 Multiple Instances

Users have the capability to open multiple objects simultaneously, with the application offering two distinct modes: Time Moments and Channels. Although the process of opening files remains consistent across both modes, the treatment of the loaded objects differs significantly.

In the Time Moments mode, the application treats the reconstructed objects as iterations of the same object occurring at different time points. While only one object is visible at any given moment, all objects are loaded simultaneously, allowing users to seamlessly transition between them. Conversely, the Channels mode treats each loaded object as distinct parts or channels of the same object, with all objects active simultaneously within the virtual environment.

To select and open multiple files, users follow the same process as with single-file loading. After loading the first file, they have the option to either complete the loading phase or select another file, repeating the process as necessary.

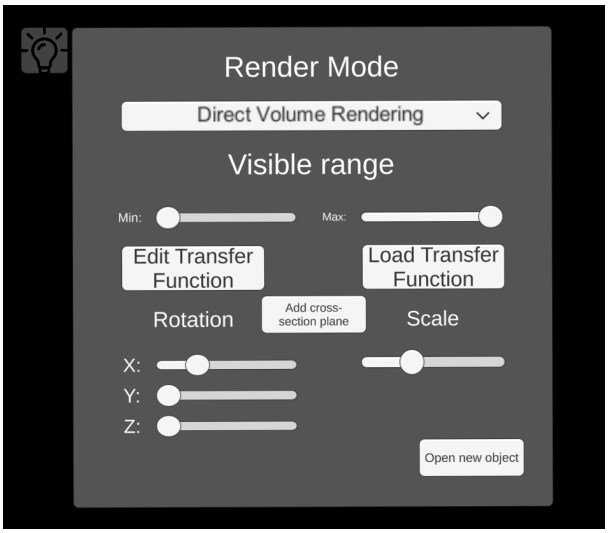
It is essential to note that all files loaded in these modes must be in the NRRD format. Additionally, the application allows users to open any NRRD file in any mode, providing flexibility but requiring users to ensure they select the appropriate mode for each object. For instance, opening two different channels of the same object in the Time Moments format will not pose a problem during the loading process.

However, hardware limitations can present challenges, particularly in the Channels mode where multiple objects are active concurrently. As file sizes increase, performance may suffer, potentially leading to longer loading times and reduced responsiveness. While the Time Moments mode mitigates some of these challenges by activating objects one at a time, loading times may still be impacted, especially with larger datasets.

4.4 Manipulate/Observe

At this stage, users are presented with fully reconstructed objects ready for observation and manipulation. As the objects are loaded into the virtual environment, users gain ac-

cess to a range of functionalities via the pop-up menu (Figure 4.2). This menu transforms to provide the necessary buttons and sliders essential for interacting with and examining the loaded objects.



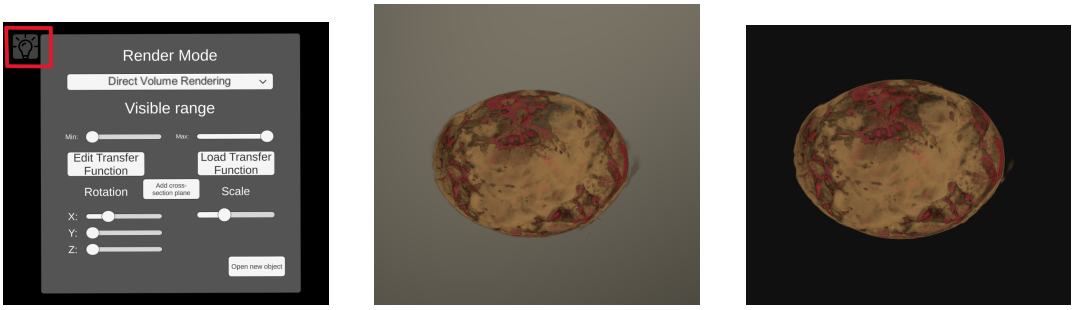
Source: Thesis application

Figure 4.2: Object Manipulation Panel

4.4.1 Light

The first function available from the outset of the application, even before loading is completed, is the light button. Designed to mitigate spatial disorientation, a directional light source has been integrated into the scene. Users can activate or deactivate this light as needed by simply pressing a button, located in the upper left corner of the menu interface (Figure 4.3a). It's worth noting that while this directional light contributes to the overall background lighting perception, the reconstructed objects themselves do not interact with it. Additionally, during object analysis, it is recommended to turn off the scene lighting for optimal visualization.

4.4.2 Move



Source: Thesis application

(a) Light switch

Source: Thesis application

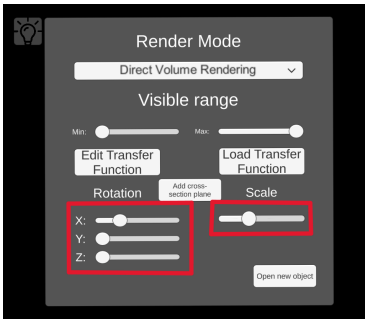
(b) Light active

Source: Thesis application

(c) Light inactive

Figure 4.3: Light in Application

Another method of interacting with the object is through the grab and move function. Users can point to the object, grip their controller handle, and effectively grab the object within the virtual environment. While the object is grabbed, it follows the user's movement, allowing them to reposition it within the virtual space. Once the user releases the object, it remains in its new location within the virtual world. This function enhances immersion and facilitates analysis, particularly when multiple objects are active simultaneously.



Source: Thesis application

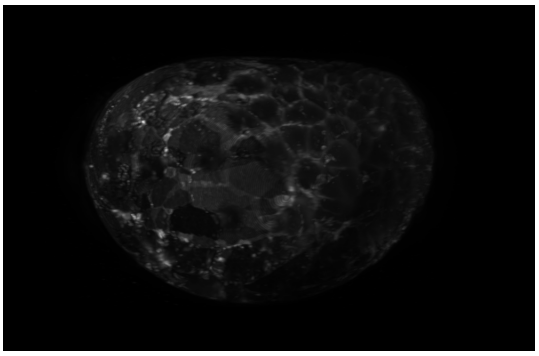
Figure 4.4: Rotation and Scale sliders

4.4.3 Rotation/scale

In addition to the rotation capability when the object is grabbed, users may require a more precise method of rotation. To cater to this need, users can access three sliders, each responsible for rotation along a specific axis (Figure 4.4). Furthermore, alongside the precise rotation sliders, there is a scale slider available, allowing users to adjust the scale of the object in real-time.

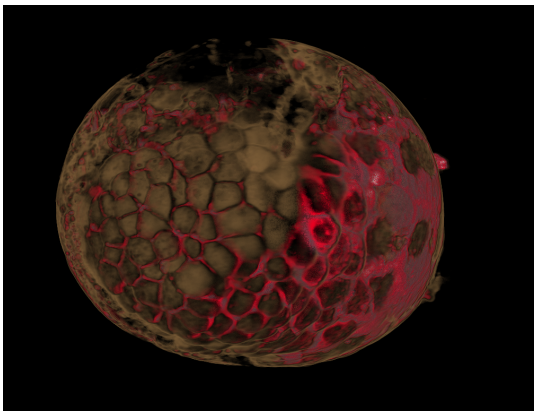
4.4.4 Render Mode

Next, users have the option to adjust how the object is rendered, with two available modes: Direct Rendering and Maximum Intensity Projection. In Direct Rendering mode, users can manipulate the opacity and color of the object for each sample used in its reconstruction through the transfer function. This allows users to selectively highlight or obscure different parts of the object, facilitating observation and analysis (Figure 4.5b). On the other hand, Maximum Intensity Projection mode presents a grayscale visualization of the object, ideal for extracting and examining its features (Figure 4.5a).



Source: Thesis application

(a) MIP example



Source: Thesis application

(b) DVR example

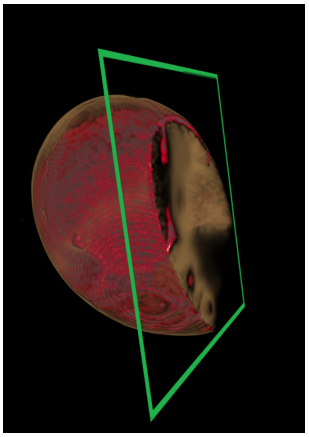
Figure 4.5: DVR and MIP Examples

4.4.5 Visible Range

Another visual functionality allows users to control the visible range of the object. From the user's perspective, this visible range can be likened to a radius cut of the object being visualized. Users have the flexibility to adjust this visible range, either excluding the upper layers of the object to focus more on its core features, or removing the core to examine features of the membrane in greater detail.

4.4.6 Cross Plane

The final functionality implemented is the ability to add a cross-section plane. As its name implies, the cross-section plane is a rectangular plane that enables users to peer inside the object (Figure 4.6). Users can manipulate the plane using the same method as the object, allowing for seamless positioning of either the reconstructed object around the plane or the plane around the object.



Source: Thesis application

Figure 4.6: Cross Section plane

4.4.7 Time moments mode

When users opt to open files in the time moments mode, an additional slider appears in the menu, enabling users to switch between active objects. This feature allows users to seamlessly change the active objects in real-time without any additional delays. Each object has its own settings profile, so when the active object is changed, the settings profile also changes accordingly.

4.4.8 Channels mode

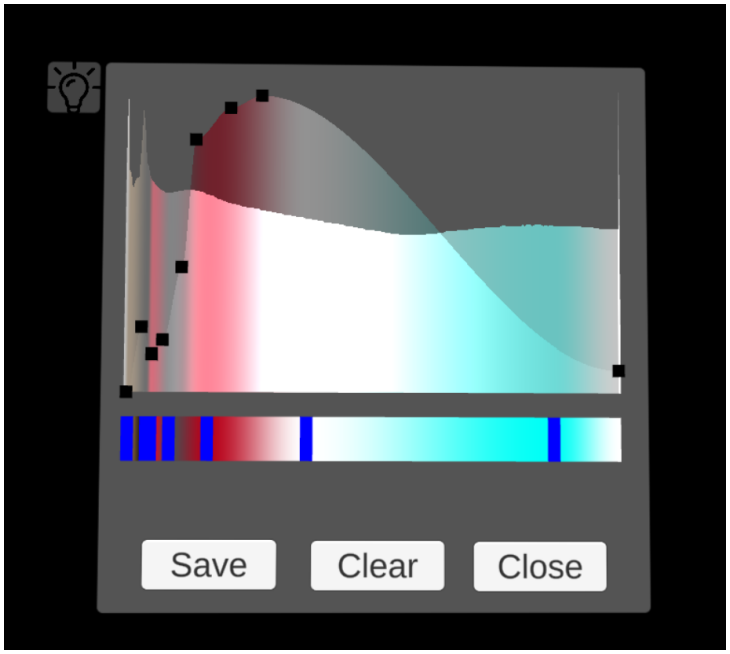
When the channels mode is selected, additional UI options become available. In this mode, all the reconstructed objects are active and can be interacted with. To facilitate the adjustment of settings for each object, a drop-down menu appears, listing all the available objects. This menu allows users to select and modify the settings for individual objects as needed.

4.5 Transfer Function

The transfer function offers users the ability to customize the appearance and visualization of their objects according to their preferences and analytical needs. When opting for direct volume rendering, users are empowered to finely adjust both the color and opacity of individual segments comprising the object.

4.5.1 Edit

The graphical interface for adjusting the transfer function is presented to the user in a distinct manner. It appears as a curve overlaid on a histogram, providing a visual representation of the data manipulation. On the x-axis of the histogram lies the visual range, while the y-axis denotes the intensity of the information. Correspondingly, within the transfer function, the x-axis still represents the visual range, but the y-axis reflects the alpha value of the color. Users have the ability to add nodes to the curve, enabling them to shape its trajectory. By moving the node along the vertical axis, users can adjust the alpha value, while horizontal movements determine the specific point along the visual range where this alpha value applies (Figure 4.7).



Source: Thesis application

Figure 4.7: Transfer function editing panel

Next to this graph, a colored spectrum is displayed, offering users additional control over the visualization. Users can add nodes to this bar and assign a color to each node. When two nodes are set adjacent to one another, the space between them defines a color spectrum transitioning smoothly from one color to the next. The position of each node

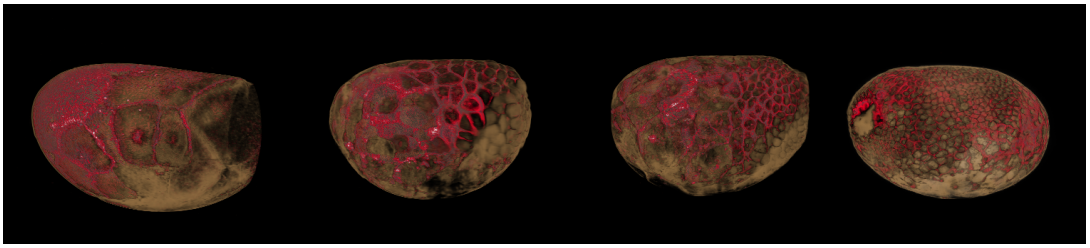
on this spectrum bar corresponds to the color assigned to that specific point along the visual range.

4.5.2 Save/Load

Configuring a transfer function can be a meticulous endeavor, involving the placement of numerous nodes to shape the function and assign colors to specific visual ranges. Given the potentially tedious nature of this process, users are provided with the option to save the transfer function to a separate file. By saving the entire transfer function to a file with the .tf suffix, users can preserve their configurations for future use, eliminating the need to recreate the function repeatedly.

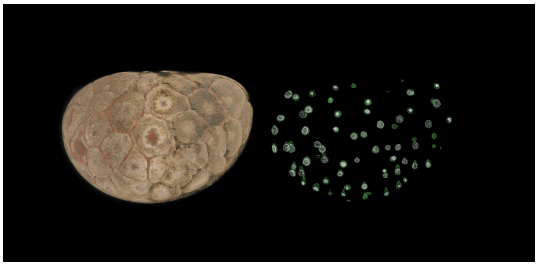
The saving process is seamlessly integrated into the application's interface. Located within the edit tab, users can easily access the save button. Upon activation, the file browsing tab opens, enabling users to navigate to their desired location and save the file. To input the file name, a virtual pop-up keyboard is provided, ensuring a user-friendly experience.

4.6 Visualization Results



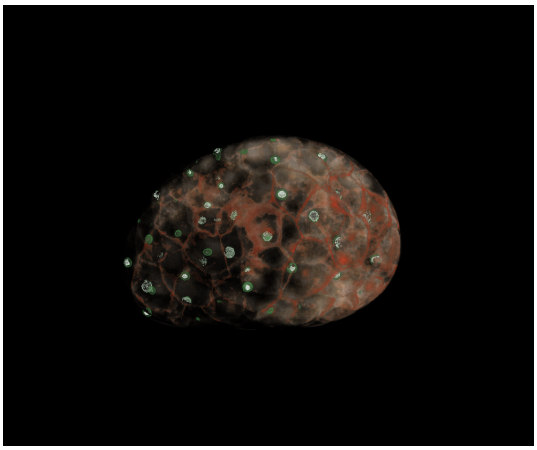
Source: Thesis application

Figure 4.8: Results of 3d rendered object for different angles using Direct Volume Rendering technique



Source: Thesis application

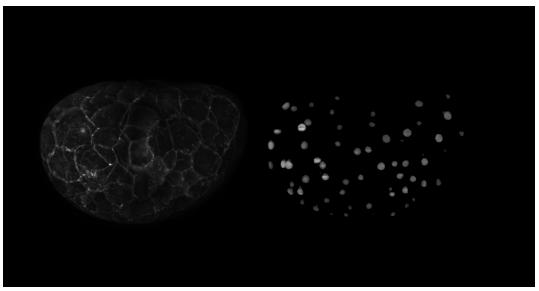
(a) Membrane next to cells



Source: Thesis application

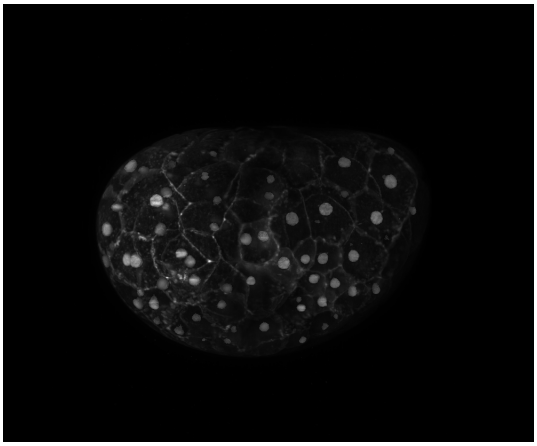
(b) Membrane and cells overlapped

Figure 4.9: Membrane and Cells with DVR technique



Source: Thesis application

(a) Membrane next to cells



Source: Thesis application

(b) Membrane and cells overlapped

Figure 4.10: Membrane and Cell with MIP technique

Chapter 5

Implementation

5.1 Introduction

Chapter 5 delves into the implementation details of the project, providing insights into how various components were developed and integrated to create the final application. From preprocessing the data to visualizing them within a virtual environment and incorporating analysis tools, this chapter offers a comprehensive exploration of the technical aspects behind the project's implementation.

The project was approached methodically by breaking it down into four distinct pillars, each encompassing a crucial aspect of the application's functionality. These pillars, namely Preprocessing, Data Visualization, Virtual Environment, and Analysis Tools, allowed for a systematic development process, wherein each section could be addressed individually and sequentially.

With the exception of the preprocessing stage, which was executed in Python, the entirety of the project's implementation was carried out within the Unity framework using C# scripting. From data visualization to creating a virtual environment and integrating analysis tools, nearly all functionalities were realized through scripts. These scripts were responsible for defining the behavior and logic of various elements within the application. Additionally, while the visual aspects of the application, excluding the reconstructed models, were crafted within the Unity scene editor, scripts played a crucial role in orchestrating interactions and user interfaces.

5.2 Data acquisition

Biological data were generated from time-lapse microscopy recordings of live developing embryos from the crustacean model organism Parhyale hawaiensis [111]. Parhyale embryos were fluorescently labelled with the membrane-tethered marker Lyn-mCherry marking the outlines of all cells (Figure 4.8). Imaging was performed on an adaptive SiMView light-sheet fluorescence microscope with four orthogonal optical arms, which simultaneously illuminated the embryo with two 594 nm laser light-sheets from two opposite sides and acquired serial optical sections from the two perpendicular sides with water-dipping 16x/0.8 detection objectives [112, 113]. Multi-view acquisitions of live Parhyale embryos were carried out every 10 min at $26°C$, over a period of 60 hrs covering the early cleavage, gastrulation and germband formation stages (Figure 4.8). TIFF files of acquired raw views (z-stacks) were first corrected and resaved in lossless compression, block-based file formats (HDF5 or KLB), were then spatiotemporally registered and fused into a single output 3D image volume per time point using established MATLAB and Fiji/ImageJ pipelines [114, 115].

5.3 Preprocessing

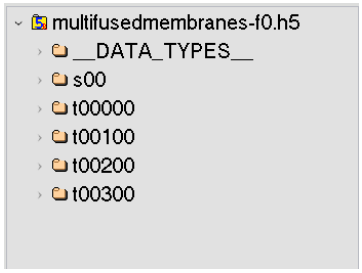
The data provided by the Foundation of Research and Technology Hellas was formatted in Hierarchical Data Format 5 (HDF5). This format, as previously discussed, is extensively employed for packaging large, complex, heterogeneous data. HDF5 utilizes a hierarchical structure similar to a file directory, enabling the organization of data within the file in many different structured ways.

Accordingly, the first step involved opening the file and examining its structure. To accomplish this, the HDFView 3.3.1 application developed by The HDF Group was utilized. This application functions as a file browser specifically designed for HDF files, facilitating the traversal and exploration of the dataset provided.

5.3.1 Data Structure

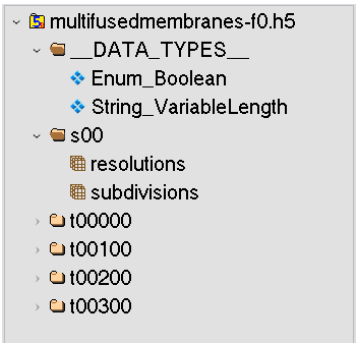
Opening the file with the HDFView app revealed the core structure of the dataset.

The top level of the file tree consisted of six folders (Figure 5.1a). The first two acted as "headers" (Figure 5.1b) for the file, providing information for the remaining structure.



Source: FORTH Datasets Open In HDFview

(a) File structure



Source: FORTH Datasets Open In HDFview

(b) Header folders

Figure 5.1: HDF file

The "__DATA_TYPES__" folder contained details about the type of each individual cell of the data, while "s00" provided information about the structure, characteristics, and subdivisions of each of the other folders.

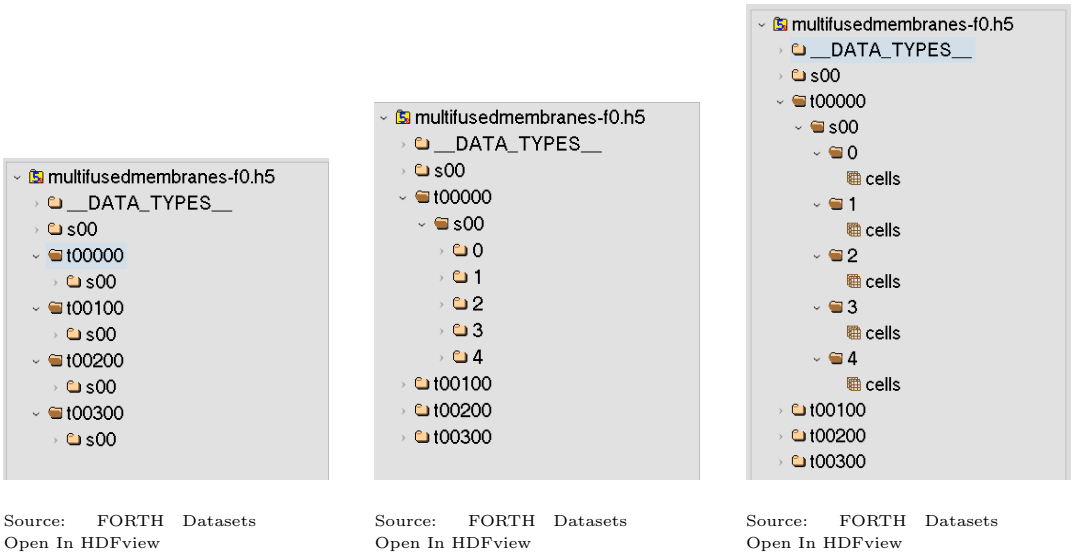
The remaining four folders represented four distinct datasets (Figure 5.2a), each available in multiple qualities (Figure 5.2b). Within each folder, a "s00" folder as defined in the header. Within these "s00" folders, there were five subfolders containing the actual data (Figure 5.2c).

After examining the file, it became apparent that it consisted of multiple numerical arrays organized within folders according to the described structure. The subsequent step involved extracting each individual dataset to isolate it for further processing.

5.3.2 Data Decomposing

Following the exploration of the file and understanding its structure, a Python script was developed to process and manipulate its contents. To accomplish this task, the h5py library, developed by The HDF Group, was utilized. The objective now is to use the script to read the file, navigate to the relevant folder, and isolate the numerical arrays representing the datasets (Figure 5.3).

To read the file, the script initially accepts the filename as the first input and proceeds to read it. Subsequently, it compiles all the folders, referred to as keys in the h5py library, into a list. In the latest version of the script, it also incorporates a second input denoting



(a) Timed Datasets (b) Dataset Qualities (c) Datasets

Figure 5.2: File Tree

the TimeSet (0-3) and the desired quality level of this TimeSet (0-4), where 0 represents the highest quality and 4 the lowest. Leveraging these additional inputs, the script navigates to the relevant folder until it accesses the dataset.

5.3.3 Data Formation

Each dataset comprises a three-dimensional array of numbers representing the captured information. In a 2D analysis, it resembles a stack of grayscale images (unsigned integer 16-bit), while in a 3D perspective, it forms a cube with the embryo contained within. Within this cube, there are areas devoid of information, where the values are either 0 or very close to it. Recognizing this characteristic, the script trims these empty cells to reduce the dataset's size without sacrificing any information.

The script processes the data variable, which holds the dataset, converting it into a numpy array of type uint16. During this conversion, it trims a portion of each dimension to optimize the dataset's size and enhance performance (Figure 5.4).

This optimization has proven invaluable for enhancing the application's performance, particularly in terms of loading times, and has enabled the handling of higher quality datasets within the same hardware specifications.

```
#Data Decomposition

filename_h5=sys.argv[1]                    #argv[1] is the file name
h5          =h5py.File(filename_h5,"r")
groupList   =list(h5.keys())               #keys are the folders in the top level

timeSet=[]                                 #Get all the timed folder in a list
timeSet.append(h5.get(groupList[2]))
timeSet.append(h5.get(groupList[3]))
timeSet.append(h5.get(groupList[4]))
timeSet.append(h5.get(groupList[5]))


s0=timeSet[ts].get(list(timeSet[ts])[0])   #Get the s00 folder
t=s0.get(list(s0)[scale])                  #Get the quality folder
data=t.get(list(t)[0])                     #Get the dataset
```

Source: Python script

Figure 5.3: File decomposition

```
#Data Formation
d=np.array(np.uint16(data[ int(data.shape[0]*StartCut) : int(data.shape[0]*StopCut),   #take a portion of each dimension
                           int(data.shape[1]*StartCut) : int(data.shape[1]*StopCut),
                           int(data.shape[2]*StartCut) : int(data.shape[2]*StopCut)]))
```

Source: Python script

Figure 5.4: Data Formation

5.3.4 Data Export

The last stage involves exporting the newly processed array into a nearly raw raster data (.nrrd) file format. For this task, the script utilizes the pynrrd library, which provides the functionality to write nrrd files.

The file name includes the TimeSet along with the quality of the file. This information is crucial because after processing all the data, 20 files would have been created, and there could be confusion between them. Additionally, the file name includes the word ”cut” and the new dimensions of the dataset to clarify whether the datasets have been trimmed or not (Figure 5.5).

In conclusion, the DataParser script in Python efficiently converts HDF file datasets into the necessary NRRD files, essential for later use within the application. With this

```
#Data Export
#FilenameFormat = t_timeSet_quality_cut_xsize_ysize_zsize.nrrd
filename_nrrd="t_"+str(ts)+"_"+str(scale)+"_(cut)_("+str(d.shape[0])+","  +str(d.shape[1])+","+str(d.shape[2])+ ").nrrd"
nrrd.write(filename_nrrd,d)
```

Source: Python script

Figure 5.5: Data Export

preprocessing phase complete, the groundwork is laid for addressing the subsequent pillars of development.

5.4 Data Visualization

With the datasets now in the suitable format, the subsequent phase involved transforming them into three-dimensional objects using volume rendering techniques. The fundamental approach was to perceive the datasets as a stack of images and apply volume rendering algorithms. As previously discussed, these algorithms excel in converting images into volumetric information.

To integrate volume rendering algorithms into Unity, the initial step involved researching available libraries and packages to determine which algorithms were supported. While Unity itself lacked native libraries for volume rendering, numerous projects were accessible through the Unity Asset Store. Further research revealed the "Unity Volume Rendering Master" project as a viable option. Developed and published by Matias Lavik under the MIT license [116], this project presented an open-source implementation of volume rendering techniques in Unity. It provided a structured data path that was adapted to suit the requirements of this application and modified to enhance performance, enable the loading of larger files and include tools and functionalities for users. Moreover, the project initially included a rudimentary virtual reality implementation, which was extensively overhauled to achieve superior performance and compatibility with VR headsets and controllers.

```
2 references
public class SimpleITKImageFileImporter : IImageFileImporter
{
    7 references
    public VolumeDataset Import(string filePath)
    {
        ImageFileReader reader = new ImageFileReader();

        reader.SetFileName(filePath);

        Image image = reader.Execute();

        // Cast to 32-bit float
        image = SimpleITK.Cast(image, PixelIDValueEnum.sitkFloat32);

        VectorUInt32 size = image.GetSize();

        int numPixels = 1;
        for (int dim = 0; dim < image.GetDimension(); dim++)
            numPixels *= (int)size[dim];

        // Read pixel data
        float[] pixelData = new float[numPixels];
        IntPtr imgBuffer = image.GetBufferAsFloat();
        Marshal.Copy(imgBuffer, pixelData, 0, numPixels);

        VectorDouble spacing = image.GetSpacing();

        // Create dataset
        VolumeDataset volumeDataset = new VolumeDataset();
        volumeDataset.data = pixelData;
        volumeDataset.dimX = (int)size[0];
        volumeDataset.dimY = (int)size[1];
        volumeDataset.dimZ = (int)size[2];
        volumeDataset.datasetName = "ImportedDataset";
        volumeDataset.filePath = filePath;
        volumeDataset.scaleX = (float)(spacing[0] * size[0]);
        volumeDataset.scaleY = (float)(spacing[1] * size[1]);
        volumeDataset.scaleZ = (float)(spacing[2] * size[2]);

        volumeDataset.FixDimensions();

        return volumeDataset;
    }
}
```

Source: Unity script

Figure 5.6: Image Importer

5.4.1 File Import

The process of importing files into the application begins with the user's interaction, typically initiated by pressing the "Open File" button within the application's interface. This action prompts the application to engage with the user, guiding them through menus and a file browser interface. These interactive elements provide the user with the means to navigate their local file system and select the desired dataset for import. Once the user has chosen a file and confirmed their selection, the application receives the file path, signaling the start of the import process.

Upon receiving the file path, the application initializes an ImageFileImporter object using the SimpleITK Library. This library, known for its efficacy in handling various medical image formats including NRRD, provides the necessary tools to read and interpret the selected file. Leveraging the capabilities of SimpleITK, the application gains access to the contents of the dataset, paving the way for further processing and visualization.

With the dataset successfully read into memory, the next phase involves initiating the creation of the 3D object. The final step in the Importer script is to instantiate a VolumeDataset object and provide it with the data for each dimension, as well as the scales of each dimension as it shown in figure 5.6. This pivotal moment marks the transition from raw data to a tangible, three-dimensional representation within the virtual environment.

5.4.2 Object Creation

At this moment, the data, once loaded into Unity, is stored as a float array. Now, the next step involves transforming this data into three-dimensional objects. The VolumeDataset object, which contains the data, serves as the input to the VolumeObjectFactory script. This script is responsible for processing the data and configuring all the necessary parameters to generate the final three-dimensional object.

In the project's assets, a prefab named volumeContainer is designed specifically for this purpose. This prefab includes a cube mesh and a meshRenderer with a custom shader called VolumeRendering. The initial task of the VolumeObjectFactory is to instantiate an empty GameObject and attach to it the meshRenderer and mesh from the volumeContainer prefab. Additionally, the VolumeObjectFactory adds the volumeRenderedObject

component, which will manage the settings of the reconstructed object. During this process, once the meshRenderer is attached to the GameObject, the volumeDatasetObject is also passed as a parameter. Within this step, the meshRenderer retrieves the data from the volumeDatasetObject and stores them not as a float array, but rather in a Texture3D format.

Once all these steps are completed, the volumeObjectFactory returns a GameObject comprising a cube container, a 3D texture representing the information of the dataset, a material featuring a custom shader to implement the volume rendering techniques, and the volumeRenderedObject component. This component is responsible for managing all the real-time modifications that will occur within the application.

5.4.3 Render Modes

In the previous steps, the data were successfully loaded into Unity and transformed into a 3D texture within a GameObject. However, at this stage, the object remains transparent, lacking the necessary visualization. To address this, a custom shader is applied to the material of the GameObject. This shader is crucial for implementing the raymarching technique, enabling the object to assign colors to each part of the 3D texture. Through this shader, the reconstructed object gains the ability to represent the volumetric information accurately, thus completing the Volume Rendering algorithm.

5.4.3.1 Raymarching

In order to implement raymarching in the custom shader, two structs and several functions were introduced. These components are essential for handling the raymarching process, providing the necessary framework and utilities to enable accurate volume rendering within the shader.

The first struct, RayInfo, is designed to hold the necessary information for the raymarching process. It includes startPos and endPos, which represent the starting and ending positions of the ray, Direction, indicating the ray's direction, and the intersection boundaries of the box. The second struct, RaymarchInfo, extends the functionality of RayInfo by also including the number of steps the ray will take and the step size.

To implement the raymarching algorithm, several functions are defined within the custom shader. The first function is responsible for creating the ray. This function

```
struct RayInfo
{
    float3 startPos;
    float3 endPos;
    float3 direction;
    float2 aabbInters;
};
```

Source: Unity script

(a) RayInfo struct

```
struct RaymarchInfo
{
    RayInfo ray;
    int numSteps;
    float numStepsRecip;
    float stepSize;
};
```

Source: Unity script

(b) RaymarchInfo struct

determines the direction of the ray by calculating the angle of the object relative to the camera. It identifies the start position of the ray, which is the closest or farthest point of the object to the camera, depending on whether the ray is cast from back-to-front or front-to-back. The function also locates all the intersection points where the ray cuts through the object, truncating the ray before and after it intersects the volume, and finally calculates the end position. The second function creates the RaymarchInfo struct. It calculates the step size to ensure it does not exceed the maximum number of steps and determines the number of steps that will fit within the object's boundaries. This encapsulates the setup required for the raymarching process, ensuring accurate and efficient rendering of the volume.

5.4.3.2 Maximum Intensity Projection (MIP)

In order to render the object using the Maximum Intensity Projection (MIP), the following steps were followed. First, the shader creates the RayInfo and RaymarchInfo using the functions mentioned earlier. It then follows a straightforward process for each step, starting from the beginning and continuing until the number of steps parameter is reached. At each step, the shader reads the density of the current point and keeps track of the maximum value encountered. At the end of this iteration, the shader colors only the point with the highest intensity (Figure 5.8).

5.4.3.3 Direct Volume Rendering (DVR)

In order to render the object using Direct Volume Rendering (DVR), the approach follows a logic similar to Maximum Intensity Projection (MIP) but with a crucial difference.

```
// Maximum Intensity Projection mode
frag_out frag_mip(frag_in i)
{
    #define MAX_NUM_STEPS 512

    RayInfo ray = getRayBack2Front(i.vertexLocal);
    RaymarchInfo raymarchInfo = initRaymarch(ray, MAX_NUM_STEPS);

    float maxDensity = 0.0f;
    float3 maxDensityPos = ray.startPos;
    for (int iStep = 0; iStep < raymarchInfo.numSteps; iStep++)
    {
        const float t = iStep * raymarchInfo.numStepsRecip;
        const float3 currPos = lerp(ray.startPos, ray.endPos, t);



        const float density = getDensity(currPos);
        if (density > maxDensity && density > _MinVal && density < _MaxVal)
        {
            maxDensity = density;
            maxDensityPos = currPos;
        }
    }

    // Write fragment output
    frag_out output;
    output.colour = float4(1.0f, 1.0f, 1.0f, maxDensity); // maximum intensity
```

Source: Unity script

Figure 5.8: MIP implementation

During DVR, the shader iterates through the raymarching process as described earlier, but instead of simply finding the highest density value along the ray, it consults a transfer function for each density encountered. The transfer function, which will be detailed further, dictates the color and opacity (alpha value) assigned to each density point within the volume. This allows for a more sophisticated visualization where the object's interior is colored and rendered based on the properties defined by the transfer function, providing greater flexibility in highlighting specific features and structures within the volumetric data (Figure 5.9).

```
// Direct Volume Rendering
frag_out frag_dvr(frag_in i)
{
    #define MAX_NUM_STEPS 512
    #define OPACITY_THRESHOLD (1.0 - 1.0 / 255.0)

    RayInfo ray = getRayFront2Back(i.vertexLocal);
    RaymarchInfo raymarchInfo = initRaymarch(ray, MAX_NUM_STEPS);

    float3 lightDir = normalize(ObjSpaceViewDir(float4(float3(0.0f, 0.0f, 0.0f), 0.0f)));

    // Create a small random offset in order to remove artifacts
    ray.startPos += (JITTER_FACTOR * ray.direction * raymarchInfo.stepSize) * tex2D(_NoiseTex, float2(i.uv.x, i.uv.y)).r;

    float4 col = float4(0.0f, 0.0f, 0.0f, 0.0f);

    float tDepth = raymarchInfo.numStepsRecip * (raymarchInfo.numSteps - 1);

    for (int iStep = 0; iStep < raymarchInfo.numSteps; iStep++)
    {
        const float t = iStep * raymarchInfo.numStepsRecip;
        const float3 currPos = lerp(ray.startPos, ray.endPos, t);

        // Get the dansity/sample value of the current position
        const float density = getDensity(currPos);

        // Apply visibility window
        if (density < _MinVal || density > _MaxVal) continue;
        float4 src = getTF1DColour(density);

        src.rgb *= src.a;
        col = (1.0f - col.a) * src + col;

        if (col.a > 0.15 && t < tDepth) {
            tDepth = t;
        }
    }

    // Write fragment output
    frag_out output;
    output.colour = col;
```

Source: Unity script

Figure 5.9: DVR implementation

5.4.4 Transfer Function

In the context of Direct Volume Rendering (DVR), as previously mentioned, the shader retrieves color and opacity information for each density point from a transfer function. This transfer function is represented as a 2D texture, which is stored within the object itself. Users have the capability to dynamically adjust this function by manipulating color and alpha points interactively. Each adjustment triggers a refresh of this texture to reflect the updated input, and subsequently, the new texture data is passed to the shader to dynamically update the visual output accordingly. This interactive capability allows

users to fine-tune the rendering of the volumetric data in real-time, facilitating detailed exploration and analysis of the object's internal structures.

5.4.5 Finalize

By this stage, the object reconstruction process is complete. The dataset, originally a stack of images, was imported into Unity, transformed into a 3D texture, and encapsulated within a game object equipped with a custom shader tailored for volume rendering techniques. This marks the end of the Data Visualization segment, paving the way for the subsequent exploration of the 3D environment.

Before transitioning to the next segment, some additional actions are performed on the object to integrate it further into the application. This includes adding the interact component, essential for interaction within the Virtual Reality environment (which will be discussed in subsequent sections). Furthermore, initialization tasks related to the user interfaces (UIs) are also carried out, setting the stage for their detailed analysis in upcoming sections (Figure 5.10).

```
IImageFileImporter importer = ImporterFactory.CreateImageFileImporter(ImageFileFormat.NRRD);
VolumeDataset dataset = importer.Import(result.path);

if (dataset != null)
{
    VolumeRenderedObject obj = VolumeObjectFactory.CreateObject(dataset);
    obj.gameObject.AddComponent<Interact>();
    Transform nextpanel=this.transform.Find("menu canvas").Find("Menu").Find("DataSettings");
    this.GetComponent<RuntimeVolumeEditor>().initialize(obj,nextpanel,m,mode);
}
else
{
    UnityEngine.Debug.LogError("Failed to import datset");
}
```

Source: Unity script

Figure 5.10: The main function for creating the object

5.5 Virtual Environment

The Virtual Environment is a crucial aspect of the application, providing users with an immersive experience for interacting with the reconstructed 3D objects. This section covers the integration of the hardware device, the setup and configuration of the camera system, the implementation of user controls, and the mechanisms for object and UI interaction.

5.5.1 Device connection

To connect the Pimax 8K headset to the PC, a proprietary software called Pitool is required. Pitool manages the necessary drivers and allows users to configure various settings of the device, such as resolution, refresh rate, and field of view. Additionally, Pitool provides tools for calibrating the room setup and position, ensuring optimal performance and accurate tracking within the virtual environment.

Furthermore, the Pitool software handles the tracking of the device. While the Pimax 8K supports self-tracking through its internal sensors, base station tracking was chosen for this application to enable the use of controllers, which are also tracked by the base stations. After installing the software and connecting the device to the PC, the base stations are set up and plugged into the wall, and the controllers are powered on, the Pitool service then recognizes these components and manages all the connections and tracking between them, ensuring seamless integration and accurate tracking within the virtual environment.

5.5.2 VR setup in Unity

After creating a new project in Unity, several packages need to be installed to support the loading, initialization, settings, and build support for a VR project. The first step involves installing the XR Plugin Management package through the Project Settings. Once this package is installed, the OpenXR interface is selected and installed. OpenXR is a crucial interface as it enables the development of a single project that can run on various input devices, including different HMDs and controllers. This flexibility ensures broad compatibility and simplifies the development process for VR applications

The final step involves downloading Steam and, through it, the SteamVR client. SteamVR acts as a launcher for VR applications and is set as the default system application for opening programs that utilize the OpenXR interface. In addition to launching VR applications, SteamVR provides extra functionalities such as advanced settings, and monitoring tools for device latency and performance.

After completing these steps, the external setup is finalized. The devices are now connected and configured correctly. The Pimax 8K headset, along with its base stations and controllers, are now fully integrated with Unity via the OpenXR interface and SteamVR. This setup ensures a seamless transition to VR development, allowing the project to leverage the full capabilities of the hardware.

5.5.3 Camera

The first step to creating the virtual environment is to change the rendering camera. With the XR package installed, Unity now offers the option to create an object called XR Origin. The XR Origin essentially represents the player when the application is running. Inside the XR Origin, there is a camera with an offset to enable stereoscopic rendering, ensuring the VR experience is immersive and realistic. Additionally, the XR Origin contains objects for the controllers, allowing for interaction within the virtual environment.

The XR Origin's camera also implements the tracking options from the HMD, allowing the camera to move in sync with the user's head movements. This ensures that the virtual environment updates in real-time, providing a seamless and immersive experience as the user navigates through the application.

5.5.4 Controls

Within the XR Origin setup, both the left and right controllers are represented as separate objects, mirroring the physical controllers in the virtual environment. These objects are seamlessly integrated via the OpenXR interface, allowing them to accurately track the movements of the real-life controllers. This synchronization ensures that users experience a natural interaction, where the virtual controllers move in sync with their physical counterparts.

To enhance immersion, the Oculus Hands Unity package was imported into the project. This package introduces 3D models of hands along with accompanying animations, which visually represent the controllers. While primarily serving aesthetic purposes, these hand models provide users with a more immersive experience by visually aligning their virtual actions with the movements of their physical controllers.

Moving to the functionality of the controllers, each controller object is equipped with essential components: XR Interactor Line Visual, Line Renderer, and XR Ray Interactor. These components work cohesively to extend the user's interaction capability within the virtual environment. The XR Interactor Line Visual and Line Renderer combine to draw a visible line in front of the controllers, providing users with a visual reference and indicating their interaction range. The XR Ray Interactor enables precise interaction by detecting when the ray intersects with objects or UI elements, facilitating seamless interaction even with further items.

In order to implement the functionality of the controllers within the virtual environment, the XR Interaction Toolkit was integrated into the project. This package plays a crucial role in simplifying the implementation of interactive actions by providing a comprehensive set of tools and components.

The final component integrated into the controller objects is the XR Controller (Action-Based). This component plays a crucial role in mapping a variety of virtual environment actions, such as selection, UI interaction, and rotation tracking, to predefined input actions provided by the XR Interaction Toolkit, the XRI default input actions.

The final step involves mapping the physical controller actions to the XRI input action list. This mapping process is facilitated by adding the controller scheme through the OpenXR tab within the project settings. By configuring the controller scheme in this manner, the application aligns the physical actions performed on the VR controllers with the corresponding virtual actions defined in the XRI input action list. This synchronization ensures that user interactions are accurately translated within the virtual environment, maintaining consistency and usability across different VR devices and configurations.

5.5.5 Object Interaction

As previously discussed, once a dataset is imported and reconstructed within Unity, the final step involves adding an interaction component to facilitate user interaction with the object. This interaction component, implemented as a custom script, ensures that the reconstructed object possesses the necessary components and settings for seamless interaction with VR controllers. The script begins by adding a Rigidbody component to the object, where the gravity parameter is disabled and the IsKinematic parameter is set to true. Additionally, a capsule collider is attached to the object along with the XRGrabInteractable component

The Rigidbody component is added to allow the XRGrabInteractable component to recognize and interact with the object, while configuring the gravity and kinematic parameters ensures the object behaves appropriately within the virtual environment. The collider component is essential for enabling interaction with the object using the controller rays. Lastly, the XRGrabInteractable component enables the object to respond to grab and release actions triggered by the controllers, as defined by the XRI input actions.

5.5.6 UI interaction

Implementing UI interactions was straightforward due to the predefined UI actions within the XRI input action list. To integrate UI interactions, the only requirement was to designate the objects within the virtual environment as UI elements. This straightforward approach leverages the existing XRI input actions, allowing users to interact seamlessly with UI components using the controllers within the virtual environment.

5.6 Tools and Features

This section covers the various tools and features implemented in the application to enhance the analysis and visualization of the reconstructed datasets. These tools are designed to provide users with comprehensive control over the data rendering process, enabling detailed exploration and customization of the visualized 3D objects.

5.6.1 Edit/Load/Save Transfer Functions

5.6.1.1 Edit Transfer Function

As mentioned in the 3D Visualization segment, when an object is rendered using Direct Volume Rendering (DVR), users can adjust the transfer function by moving color and opacity nodes. Additionally, users can choose the desired color from a spectrum presented next to the histogram of the transfer function.

As mentioned in the 3D Visualization segment, when an object is rendered using Direct Volume Rendering (DVR), users can adjust the transfer function by moving color and opacity nodes. Additionally, users can choose the desired color from a spectrum presented next to the histogram of the transfer function.

To solve this problem, a script was created to implement the cursor logic. This script contains functions that use the raycast method to check if the ray intersects with the histogram, the color palette, or the color spectrum. Finally, the cursor logic script returns the coordinates of the intersection point relative to the UI panel. Using the functionality of this script, each time a node is pressed, the 2D coordinates of the UI panel can be used to control the movement as if it were clicked by a cursor in a desktop environment. This approach allows users to intuitively manipulate the transfer function nodes using the VR controllers.

The color palette that appears next to the transfer function editing window allows users to choose their desired color. This palette is represented by a 2D texture in Unity. When a user selects a point on this image, the same logic is followed to obtain the 2D coordinates of that point. Once the coordinates are determined, the color value at this point on the 2D texture is read and used to update the transfer function accordingly.

5.6.1.2 Save Transfer Function

Users can press the save button on the transfer function editing panel to save the current state of the transfer function setup. To achieve this, the data is stored in a JSON file with the .tf suffix. When the save button is pressed, the application serializes the current state of the transfer function into a JSON format.

5.6.1.3 Load Transfer Function

When users press the load transfer function button, they are prompted to choose a .tf file from their directory system. Once a file is selected, the application reads the .tf file, deserializes the JSON data, and stores it into a new Transfer Function object. This allows the application to apply the previously saved transfer function settings, ensuring that users can seamlessly continue their work with their preferred configurations.

5.6.2 Cross Plane

Another feature implemented in the application is the ability to add a cross-section plane. When a cross-section plane is spawned in the virtual environment, it is tethered to a single reconstructed object. Technically, the cross-section plane is a game object with a custom shader. This shader renders the object as a transparent square, allowing the inner part of the object to be displayed through it.

All the functionality of the cross-section plane is implemented within the custom shader of the reconstructed object. When the plane is active, a flag in the Direct Volume Rendering (DVR) shader is set to true. The DVR shader then reads the position of the plane and ensures that it does not render the part of the object that is located behind the plane. This creates the effect of slicing through the object, revealing its internal structure.

The scale of the plane matches that of the object and changes accordingly as the object is resized or manipulated within the virtual environment. This synchronization ensures that the cross-section plane consistently represents a meaningful slice of the 3D dataset, enhancing the user's ability to explore and analyze the data from different perspectives.

5.6.3 Open Modes

The application offers three distinct file opening modes: Single File Open, Time Moments, and Channels mode. While Single File Open has been detailed in segment 5.3.1 File Import, Time Moments and Channels modes offer additional functionalities suited for different data exploration needs.

After users open the first file in either Time Moments or Channels mode, a UI panel appears, presenting options to either open another file or finalize the opening process.

Choosing "Open More" repeats the file loading process exactly as the first time, except for skipping the step where all existing objects are despawned.

The key functional difference between Time Moments and Channels mode lies in the finalization stage:

- Time Moments Mode: All loaded objects are stored in a list, but only the first object is initially set to active. Users can switch between different time moments using a slider control. Each time a new object is selected via the slider, the currently active object's "SetActive" value is set to false, while the newly chosen object's value is set to true. This allows users to navigate through different time frames of the dataset efficiently.

- Channels Mode: Similar to Time Moments mode, all loaded objects are stored in a list, but unlike Time Moments mode, all objects remain active simultaneously. To facilitate interaction with specific objects, a dropdown menu is provided. Selecting a different object from the dropdown dynamically adjusts the UI elements to affect the settings of the newly chosen object. This mode is ideal for scenarios where multiple channels of data need to be visualized and compared simultaneously.

5.7 User Interface

5.7.1 UI Activation

When the application launches, users find themselves in an empty virtual environment. To access UI windows, users simply press the thumbstick on either controller. This action toggles the visibility of the UI window—if it's already visible, pressing the thumbstick again will hide it, and vice versa. When the UI window appears, it resumes from the state it was in when last deactivated, starting at the Open File Choice Panel.

The placement of the UI window depends on which thumbstick was pressed (left or right). It appears above the hand that initiated the action and is oriented to always face towards the user's head. This immersive functionality is achieved through a custom button action added to the controllers' interaction list. Additionally, a script calculates the angle between the hand and the head, ensuring the UI window's proper alignment and orientation relative to the user's perspective.

5.7.2 Main Panels

The UI system of the app consists of five different panels, each offering separate information and functionalities (Figure 5.11). The visual part of these five panels was implemented in the Unity editor. This approach allowed control and management of the appropriate placements of all the elements while providing visual feedback on how they will appear inside the app.

All the panels are children of the UI manager game object. The UI manager object contains scripts for controlling the functionality of each panel and a manager script that controls which of the functionality scripts need to be used at each moment. This hierarchical structure ensures that all UI elements are centrally managed, allowing for efficient updates and interactions. The manager script acts as the central hub, coordinating the activation and deactivation of the various panel scripts based on user interactions and application state as it seen in Figure 5.11.

In order to connect the UI elements placed in the 3D environment with the scripts that control them, there are initialization steps (some occurring when the app starts and others when they need to be used). These steps utilize the panel.Find("Component_Name"). GetComponent<Component_Type>(); command of the Unity engine to get and setup the ui elements. This command searches the children list of the panel until it finds a game object with the name "Component_Name" of type "Component_Type" and returns this object. By utilizing this function, the separate development of the visual and functional design was facilitated. Going further, the five main panels will be explained.

- **Open File Choice Panel**

The first panel is a panel with three buttons allowing users to choose in which mode they want to open datasets. The options are Single File Open, Time Moments, and Channels mode. After any of these buttons are pressed, this panel deactivates, and the File Browser Panel is activated.

- **File Browser Panel**

This panel is a simplistic file browser that allows users to traverse their file directory system. This panel is used on four separate occasions: to open a dataset, open multiple datasets, open a transfer function, and save a transfer function. While the core functionality of the panel remains the same in all occasions, there are some small differences:

1. Single File Open: Only files with the .nrrd suffix are visible, while all other files do not appear inside folders. After the file is chosen and the confirm button is pressed, the next panel is the Data Manipulation panel.

2. Multiple Datasets Open: Only files with the .nrrd suffix are visible, but the next panel is the Open More panel.

3. Load Transfer Function: Only files with the .tf suffix are visible.

4. Save Transfer Function: A pop-up keyboard is added, allowing users to type the name of the file to be saved.

- **Open More Panel**

This panel features only two buttons:

1. Open More: This button returns users to the File Browser Panel to open another file.

2. Finished: Clicking this button proceeds to the Data Manipulation Panel.

The Open More Panel serves as an intermediary step when users choose to open multiple datasets sequentially. Its purpose is to streamline the process of loading additional files without returning to the main file selection screen each time.

- **Data Manipulation Panel**

The Data Manipulation Panel hosts a variety of UI elements designed to control and interact with the reconstructed object:

1. Sliders: These allow users to adjust parameters such as scale and rotation along each axis of the object.

2. Visible Range Controls: These sliders define the minimum and maximum values visible in the dataset.

3. Buttons: Options to edit or load transfer functions are accessible from this panel.

4. Mode-Specific Controls: Depending on the mode (e.g., Time Moments or Channels), additional UI elements like sliders or dropdown menus appear to select the active object or specific settings.

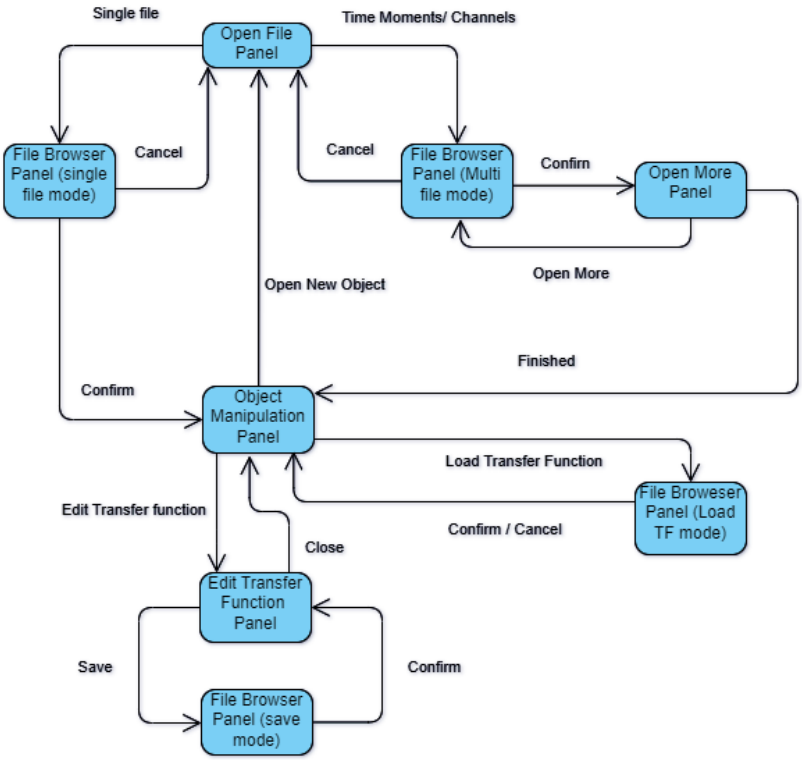
This panel consolidates essential controls for manipulating and fine-tuning the visual representation and behavior of the reconstructed object within the virtual environment.

- **Edit Transfer Function Panel**

The Edit Transfer Function Panel is dedicated to configuring the transfer function used in Direct Volume Rendering (DVR):

1. Histogram: Visualizes the distribution of voxel densities in the dataset.
2. Color Spectrum: Provides a spectrum for selecting colors to represent different densities.
3. Color Palette: Allows users to pick specific colors to assign to density ranges.

These elements collectively enable users to fine-tune how voxel densities are mapped to colors and opacities, thereby customizing the appearance of the rendered object in the virtual environment.



Source: Created in https://www.visual-paradigm.com/

Figure 5.11: Ui Panels flow chart

Chapter 6

Evaluation ,Results & Future Work

6.1 Introduction

In this chapter of the thesis, the evaluation method and results of the developed 3D Visualization application are presented, as well as potential future work discussed. Having implemented the whole application it ended up as an executable file which can be run through Windows, providing the users have installed an openXR launcher (SteamVR), any software the HMD needs (PiTool) and an OpenXR compatible HMD (PiMax 8k) is available.

6.2 Evaluation Method

6.2.1 Performance Metrics

To measure the performance of the application, three key metrics were utilized: the time it took to load a dataset, the running frames per second (FPS), and the latency of the device during operation.

- Loading Times:

The System.Diagnostics library of the Unity system was used to measure loading times. This library offers the stopwatch functionality, which was employed to measure the loading duration. The measurement started after reading the NRRD file and before

the reconstruction of the object began. The measurement stopped when the final object was ready and appeared in the virtual environment(Figure 6.1).

```
// open file

IImageFileImporter importer = ImporterFactory.CreateImageFileImporter(ImageFileFormat.NRRD);
VolumeDataset dataset = importer.Import(result.path);
// create and stard stopWatch
Stopwatch sw = new Stopwatch();
sw.Start();

if (dataset != null)
{
    VolumeRenderedObject obj = VolumeObjectFactory.CreateObject(dataset);
    obj.gameObject.AddComponent<Interact>();
    Transform nextpanel=this.transform.Find("menu canvas").Find("Menu").Find("DataSettings");
    this.GetComponent<RuntimeVolumeEditor>().initialize(obj,nextpanel,m,mode);
}
else
{
    UnityEngine.Debug.LogError("Failed to import datset");
}
// stop and print loading time
sw.Stop();
UnityEngine.Debug.Log("Loading Time "+ sw.ElapsedMilliseconds + " ms");
```

Source: Unity script

Figure 6.1: Time measurement code

- Frames Per Second (FPS):

A custom script was written to measure FPS (Figure 6.2). This script takes the timeScale-independent interval in seconds from the last frame to the current one and then prints its reciprocal in a UI textbox. This method provided a real-time display of the application's FPS, allowing for continuous performance monitoring.

- Latency Measurement:

The latency measurement was taken from SteamVR, which offers the ability to display a performance graph. This graph shows the real-time latency of the device while the application is running. This feature provided an essential insight into the application's responsiveness and the HMD's performance during use.

```
// open file

IImageFileImporter importer = ImporterFactory.CreateImageFileImporter(ImageFileFormat.NRRD);
VolumeDataset dataset = importer.Import(result.path);
// create and stard stopWatch
Stopwatch sw = new Stopwatch();
sw.Start();

if (dataset != null)
{
    VolumeRenderedObject obj = VolumeObjectFactory.CreateObject(dataset);
    obj.gameObject.AddComponent<Interact>();
    Transform nextpanel=this.transform.Find("menu canvas").Find("Menu").Find("DataSettings");
    this.GetComponent<RuntimeVolumeEditor>().initialize(obj,nextpanel,m,mode);
}
else
{
    UnityEngine.Debug.LogError("Failed to import datset");
}
// stop and print loading time
sw.Stop();
UnityEngine.Debug.Log("Loading Time "+ sw.ElapsedMilliseconds + " ms");
```

Source: Unity script

Figure 6.2: Frame Pes Second measurement script

6.2.2 Think Aloud Methodology

To evaluate the applicability of the system in the context of biological analysis and research, a formal analysis with the target audience (medical, biomedical, and biological practitioners and researchers) is necessary. However, conducting this form of evaluation was challenging. Instead, a think aloud methodology was employed with expert VR developers.

The think aloud process is a qualitative research technique used to gather insights into users' thought processes as they interact with a system or perform a task. In this methodology, participants are asked to verbalize their thoughts, feelings, and actions while they complete a task.

In this case, participants were asked to load two datasets into the app in Time Moments mode and configure their transfer functions with the goal of making as many details observable as possible. During this process, the comments made by the participants while they were trying to complete their tasks were collected. These comments provided direct insights into the user experience, highlighting any challenges or confusions faced, as well as any positive aspects of the system's design and functionality. This feedback was crucial for identifying areas that required further refinement and for validating the system's

usability within the context of VR development and potentially biological research.

6.3 Evaluation Results

6.3.1 Performance Metrics results

After a dataset was opened ten times, the average values for all the performance metrics were calculated. Additionally, measurements were taken for five different dataset resolutions, ranging from very low to very high (Table 6.1). The results are presented in Table 6.2:

	X	Y	Z
Lowest	140	131	108
Low	280	262	217
Medium	561	525	434
High	1122	1050	868
Highest	2244	2100	1736

Table 6.1: Dimension sizes for each dataset evaluated.

	Loading Time (ms)	FPS	Latency (ms)
Lowest	34	109	5.32
Low	161	106	5.33
Medium	1238	100	4.84
High	9970	100	5.42
Highest	81519	2	105.5

Table 6.2: Evaluation results

The first metric, loading time, demonstrated a nearly linear increase as the size of the dataset grew. This behavior is expected because the loading process is CPU-bound, meaning that larger files require more time to be processed and loaded into the application. The linear relationship between dataset size and loading time underscores the

importance of efficient data management and pre-processing steps to minimize delays in loading large datasets.

For the second metric, frames per second (FPS), the application consistently managed to run at an average of around 100 FPS for all resolutions, except the highest one. When the highest resolution dataset was loaded, the FPS dropped significantly, indicating that the application was exceeding the capabilities of our graphics card (Nvidia RTX 3070). This bottleneck at the GPU capacity suggests that while our system can handle high-resolution datasets, the highest resolutions require more powerful GPUs. Tests on systems with lower specifications showed similar trends, with bottlenecks occurring at correspondingly lower resolutions, further confirming that GPU capacity can be the limiting factor.

Regarding the third metric, latency, the Pimax 8K VR headset maintained a consistent latency of under 7 milliseconds across all tests, except for the highest resolution dataset. For the highest resolution, latency exceeded 100 milliseconds, which is attributed to the application's reduced performance, at this stage, rather than a limitation of the device itself. This indicates that while the headset is capable of low-latency performance, application optimization is crucial for maintaining smooth and responsive interaction, especially with extremely high-resolution data.

6.3.2 Think Aloud Methodology results

The feedback from the VR developer experts was categorized into comments about the hardware (HMD and controllers) and the app itself. Regarding the hardware, participants noted that the head-mounted display (HMD) felt heavy after extended use, and the grab action on the controllers was not always registering correctly. On a positive note, they appreciated the good external light cut-off, indicating that the HMD fit well to the face, reducing light intrusion. Concerning the app, users reported that the precision of interactions with the controller rays decreased as the distance to the object or user interfaces increased. When objects overlapped, such as in multi-instance open mode or when a cross-section plane overlapped with another object, it was unclear which object would be grabbed with the grab action. Additionally, while adjusting the transfer function, users found it challenging to place nodes accurately as they focused on the real-time changes

in the object and not focusing on the placement of the node. Finally, users mentioned that after prolonged use of the app they felt close to zero discomfort and disorientation.

6.4 Future Work

The future development of this application can be significantly enhanced by incorporating sophisticated tools for data manipulation. One such enhancement is the addition of advanced annotation tools that allow users to mark specific regions of interest within the 3D biological data. These tools would enable researchers to make detailed notes directly within the VR environment, thereby streamlining the process of documenting observations. Furthermore, implementing exporting tools would be highly beneficial, as these would allow users to save screenshots and measurements acquired during analysis. Such features would not only improve the utility of the application but also facilitate the sharing of insights and findings with other researchers and stakeholders.

In addition to data manipulation tools, improving support for collaborative VR environments is a critical area for future work. Enabling multiple users to interact with the same dataset in real time could revolutionize collaborative research and decision-making processes in medical and biological studies. This capability would allow researchers from different geographical locations to work together seamlessly, discussing findings and making joint decisions as if they were in the same room. Real-time collaboration could lead to more dynamic and interactive research sessions, fostering a deeper understanding of complex biological data and accelerating the pace of scientific discoveries.

Another promising avenue for future development is the integration of machine learning algorithms into the application. Machine learning can play a pivotal role in data analysis by automating the identification of patterns and anomalies within large datasets. By leveraging these algorithms, the application could provide researchers with insights that might be missed through manual analysis alone, thereby enhancing both the accuracy and efficiency of their work. Additionally, machine learning models could be trained to recognize specific biological structures or disease markers, offering predictive analytics that could further aid in research and diagnostic processes.

To keep pace with technological advancements, it is essential to leverage the latest hardware improvements. The use of more powerful GPUs and faster processors can significantly enhance the performance of the application, allowing for smoother and

more detailed visualizations. Furthermore, optimizing rendering algorithms and utilizing cutting-edge VR frameworks will be crucial for improving the overall user experience. As hardware continues to evolve, regularly updating the application to take advantage of these advancements will ensure that it remains at the forefront of VR-based biological data visualization.

Lastly, future work should also consider feedback from biological researchers, medical professionals, and other biology practitioners to continually refine and improve the application. Actively engaging with the scientific community will provide valuable insights into how the application is being used in real-world scenarios, identifying areas for improvement and new features that would be most beneficial. This iterative process of development, driven by expert feedback, will help ensure that the application meets the evolving needs of the scientific community and remains a vital tool for biological research in VR.

References

[1] Anthes, C., García-Hernández, R.J., Wiedemann, M., Kranzlmüller, D.: State of the art of virtual reality technology. In: 2016 IEEE Aerospace Conference. (2016) 1–19 1

[2] Jerald, J.: The VR Book: Human-Centered Design for Virtual Reality. Association for Computing Machinery and Morgan & Claypool (2015) 1, 7

[3] Mikropoulos, T.A., Natsis, A.: Educational virtual environments: A ten-year review of empirical research (1999–2009). Computers & Education **56**(3) (2011) 769–780 1

[4] Riva, G., Wiederhold, B.K.: What the metaverse is (really) and why we need to know about it (May 2022) 1

[5] Nee, A., Ong, S.: Virtual and augmented reality applications in manufacturing. IFAC Proceedings Volumes **46**(9) (2013) 15–26 7th IFAC Conference on Manufacturing Modelling, Management, and Control. 1

[6] Slater, M., Sanchez-Vives, M.V.: Enhancing our lives with immersive virtual reality. Frontiers in Robotics and AI **3** (2016) 1

[7] Abjigitova, D., Sadeghi, A.H., Peek, J.J., Bekkers, J.A., Bogers, A.J.J.C., Mahtab, E.A.F.: Virtual reality in the preoperative planning of adult aortic surgery: A feasibility study. Journal of Cardiovascular Development and Disease **9**(2) (2022) 1, 28

[8] Saleh, A., Mahmood, B.: Vr-based visualization for biological networks. **7** (05 2023) 48–68 1, 13

[9] Theart, R.P., Loos, B., Niesler, T.R.: Virtual reality assisted microscopy data visualization and colocalization analysis. BMC Bioinformatics **18**(2) (Feb 2017) 64 2

[10] Sadeghi, A., Bakhuis, W., Van Schaagen, F., Oei, F., Bekkers, J., Maat, A., Mahtab, E., Bogers, A., Taverne, Y.: Immersive 3d virtual reality imaging in planning minimally invasive and complex adult cardiac surgery. European heart journal (2020) 62–70 2, 13, 24, 28

[11] Gobbetti, E., Scateni, R.: Virtual reality: past, present and future. Studies in health technology and informatics **58** (1998) 3–20 2

[12] Sutherland, I.E.: The ultimate display. (1965) 8

[13] Heilig, M.: Sensorama. `https://en.wikipedia.org/wiki/Sensorama` (1962) 9

[14] Sutherland, I.E.: A head-mounted three dimensional display. In: Proceedings of the December 9-11, 1968, Fall Joint Computer Conference, Part I. AFIPS '68 (Fall, part I), New York, NY, USA, Association for Computing Machinery (1968) 757–764 9

[15] Baarspul, M.: A review of flight simulation techniques. Progress in Aerospace Sciences **27**(1) (1990) 1–120 10

[16] Hamad, A., Jia, B.: How virtual reality technology has changed our lives: An overview of the current and potential applications and limitations. International Journal of Environmental Research and Public Health **19**(18) (2022) 10, 12

[17] Baudisch, P.: Virtual reality in your living room: technical perspective. Commun. ACM **58**(6) (may 2015) 92 11

[18] Raji, M.A., Olodo, H.B., Oke, T.T., Addy, W.A., Ofodile, O.C., Oyewole, A.T.: Business strategies in virtual reality: A review of market opportunities and consumer experience. International Journal of Management & Entrepreneurship Research (2024) 11, 12

[19] Damaševičius, R., Maskeliūnas, R., Blažauskas, T.: Serious games and gamification in healthcare: A meta-review. Information **14**(2) (2023) 11

[20] Raman Kumar, S.: Overview of virtual reality, applications and impact on various industries. International Journal of Scientific Research in Engineering and Management (2024) 12

[21] Choukikar, H., Parte, S.: "transformative realities: The social impact of virtual reality". International Journal for Research in Applied Science and Engineering Technology **11** (12 2023) 650–663 12

[22] Pappa, D., Papadopoulos, H.: A use case of the application of advanced gaming and immersion technologies for professional training: The gamepharm training environment for physiotherapists. Electronic Journal of e-Learning **17** (06 2019) 12

[23] Zhang, M., Ding, H., Zhang, Y.: Virtual reality technology as an educational and intervention tool for autism spectrum disorder: Current perspectives and future directions. (02 2022) 13

[24] Sales, B., Machado, L., Moraes, R. In: Interactive collaboration for Virtual Reality systems related to medical education and training. (01 2011) 157–162 13

[25] Jiang, H., Vimalesvaran, S., Wang, J., Lim, K., Mogali, S., Tudor Car, L.: Virtual reality in medical students' education: A scoping review (preprint). JMIR Medical Education **8** (11 2021) 13

[26] Grassini, S., Laumann, K., Luzi, A.: Association of individual factors with simulator sickness and sense of presence in virtual reality mediated by head-mounted displays (hmds). Multimodal Technologies and Interaction **5(3)** (02 2021) 13

[27] Panzirsch, M., Weber, B., Bechtel, N., Grabner, N., Lingenauber, M.: Light-field head-mounted displays reduce the visual effort: A user study. Journal of the Society for Information Display **30** (03 2022) 14

[28] Choi, S.W., Siyeong, L., Seo, M.W., Kang, S.J.: Time sequential motion-to-photon latency measurement system for virtual reality head-mounted displays. Electronics **7** (09 2018) 171 14

[29] Zhao, C., Kim, A., Beams, R., Badano, A.: Spatiotemporal image quality of virtual reality head mounted displays. Scientific Reports **12** (11 2022) 14

[30] Sikorski, E., Palla, A.: Interactive demonstration of an ac-130 aircraft virtual reality part task trainer. Proceedings of the Human Factors and Ergonomics Society Annual Meeting **61**(1) (2017) 395–397 14

[31] Huang, Y.P.: Visual perception and fatigue in ar/vr head-mounted displays. Information Display **35** (03 2019) 4–5 14, 19

[32] Caserman, P., Garcia-Agundez, A., Zerban, A., Göbel, S.: Cybersickness in current-generation virtual reality head-mounted displays: systematic review and outlook. Virtual Reality **25** (12 2021) 1–18 14

[33] Ahmed, H., Ullah, I., Khan, U., Qureshi, M., Manzoor, S., Muhammad, N., Khan, M.U.S., Nawaz, R.: Adaptive filtering on gps-aided mems-imu for optimal estimation of ground vehicle trajectory. Sensors **19** (12 2019) 5357 15

[34] Shakerian, A., Eghmazi, A., Goasdoué, J., Landry, R.J.: A secure zupt-aided indoor navigation system using blockchain in gnss-denied environments. Sensors **23**(14) (2023) 15

[35] Gheorghiu, S., Nagy-Betegh, K., Molnar, R., Grammenos, R.: Wallsy: The uwb and smartmesh ip enabled wireless ad-hoc low-power localization system. (06 2021) 15

[36] Almousa, O., Prates, J., Yeslam, N., Gregor, D.M., Zhang, J., Phan, V., Nielsen, M., Smith, R., Qayumi, K.: Virtual Reality Simulation Technology for Cardiopulmonary Resuscitation Training: An Innovative Hybrid System With Haptic Feedback. Simulation & Gaming **50**(1) (February 2019) 6–22 16

[37] he, C., Kazanzides, P., Sen, H.T., Kim, S., Liu, Y.: An inertial and optical sensor fusion approach for six degree-of-freedom pose estimation. Sensors **15** (07 2015) 16448–16465 16, 17

[38] Koniaris, B., Huerta, I., Kosek, M., Darragh, K., Malleson, C., Jamrozy, J., Swafford, N., Iglesias Guitián, J., Moon, B., Israr, A., Mitchell, K.: Iridium: immersive rendered interactive deep media. (07 2016) 1–2 16

[39] Franček, P., Jambrošić, K., Horvat, M., Planinec, V.: The performance of inertial measurement unit sensors on various hardware platforms for binaural head-tracking applications. Sensors **23**(2) (2023) 16

[40] Filippeschi, A., Schmitz, N., Miezal, M., Bleser, G., Ruffaldi, E., Stricker, D.: Survey of motion tracking methods based on inertial sensors: A focus on upper limb human motion. Sensors **17**(6) (2017) 16

[41] Kim, M., Lee, S.: Fusion poser: 3d human pose estimation using sparse imus and head trackers in real time. Sensors **22** (06 2022) 4846 17

[42] Palmisano, S., Allison, R., Teixeira, J., Kim, J.: Differences in virtual and physical head orientation predict sickness during active head-mounted display-based virtual reality. Virtual Reality **27** (12 2022) 17

[43] Rico, M., Botero-Valencia, J., Hernández García, R.: Vertical jump data from inertial and optical motion tracking systems. Data **7** (08 2022) 116 17

[44] Reimer, D., Podkosova, I., Scherzer, D., Kaufmann, H.: Evaluation and improvement of hmd-based and rgb-based hand tracking solutions in vr. Frontiers in Virtual Reality **4** (2023) 17

[45] Hye Sung Moon, G.O., Jeon, M.: Hand tracking with vibrotactile feedback enhanced presence, engagement, usability, and performance in a virtual reality rhythm game. International Journal of Human–Computer Interaction **39**(14) (2023) 2840–2851 18

[46] Ganias, G., Lougiakis, C., Katifori, A., Roussou, M., Ioannidis, Y., Ioannidis, I.: Comparing different grasping visualizations for object manipulation in vr using controllers. IEEE Transactions on Visualization and Computer Graphics **PP** (05 2023) 1–10 18

[47] Fernandes, A.S., Murdison, T.S., Proulx, M.J.: Leveling the playing field: A comparative reevaluation of unmodified eye tracking as an input and interaction modality for vr. IEEE Transactions on Visualization and Computer Graphics **29**(5) (may 2023) 2269–2279 18

[48] Drakopoulos, P., Koulieris, G.a., Mania, K.: Eye tracking interaction on unmodified mobile vr headsets using the selfie camera. ACM Trans. Appl. Percept. **18**(3) (may 2021) 18

[49] Hussain, R., Chessa, M., Solari, F.: Exploring Foveation Techniques for Virtual Reality Environments. In: 19th International Conference on Computer Graphics Theory and Applications, Rome, Italy, SCITEPRESS - Science and Technology Publications (February 2024) 321–328 18

[50] Rantamaa, H.R., Kangas, J., Kumar, S., Mehtonen, H., Järnstedt, J., Raisamo, R.: Comparison of a vr stylus with a controller, hand tracking, and a mouse for object manipulation and medical marking tasks in virtual reality. Applied Sciences **13** (02 2023) 2251 18

[51] Wilson, A., Hua, H.: High-resolution optical see-through vari-focal-plane head-mounted display using freeform alvarez lenses. (05 2018) 140 19

[52] Balram, N.: Solving the rendering puzzle. Information Display **33**(6) (2017) 4–34 19

[53] Mahesh, B., Komarasamy G, D.: Conception about the data visualization techniques including data stream mining and bioinformatics. TechnoareteTransactions on Intelligent Data Mining and Knowledge Discovery **2** (02 2022) 19

[54] Sirait, N., Situmeang, S., Zaluku, W., Panjaitan, W.: Data visualization using the rapid miner application to evaluate sales patterns. INFOKUM **11** (08 2023) 48–58 19, 21

[55] Rachmat, F., Kiagus, Agus, P., Abdul, A.: Analysis of financial data descriptions about investment and the stock market based on gender and age attributes. Jurnal E-Komtek (Elektro-Komputer-Teknik) (2023) 19, 21

[56] Eddy, J., Lewis, K.: Multidimensional design visualization in multiobjective optimization. (09 2002) 19, 24, 26

[57] Rosenkranz, T., Jaillon, A.: Accessible and interactive: New methods of data visualization as tools for data analysis and information sharing in transitional justice research. Transitional Justice Review (2016) 20

[58] Gnanasekaran, R.K., Marciano, R.: Piloting data science learning platforms through the development of cloud-based interactive digital computational notebooks. PoS **ISGC2021** (2021) 018 21

[59] A.S.K., W., Ruvan, A., M.W.P, M.: A systematic review of 3d metaphoric information visualization. International Journal of Modern Education and Computer Science(IJMECS) (2023) 22, 23

[60] Zhao, H., Wang, S.: A coding basis and three-in-one integrated data visualization method 'ana' for the rapid analysis of multidimensional omics dataset. Life **12** (11 2022) 22, 24, 25

[61] Gall, H., Jazayeri, M., Riva, C.: Visualizing software release histories: The use of color and third dimension. (01 1999) 99–108 22, 25

[62] Lütjens, M., Kersten, T., Dorschel, B., Tschirschwitz, F.: Virtual reality in cartography: Immersive 3d visualization of the arctic clyde inlet (canada) using digital elevation models and bathymetric data. Multimodal Technologies and Interaction **3** (02 2019) 9 22

[63] Khayyal, H., Zidan, Z., Beshr, A.: Creation and spatial analysis of 3d city modeling based on gis data. Civil Engineering Journal **8** (01 2022) 105–123 22

[64] Nam, J.W., Isenberg, T., Keefe, D.F.: V-mail: 3d-enabled correspondence about spatial data on (almost) all your devices. IEEE Transactions on Visualization and Computer Graphics **30**(4) (2024) 1853–1867 22, 25

[65] Pietriga, E.: Engineering interactive geospatial visualizations for cluster-driven ultra-high-resolution wall displays. In: Companion of the 2022 ACM SIGCHI Symposium on Engineering Interactive Computing Systems. EICS '22 Companion, New York, NY, USA, Association for Computing Machinery (2022) 3–4 22

[66] Fouché, G., Argelaguet, F., Faure, E., Kervrann, C.: Immersive and interactive visualization of 3d spatio-temporal data using a space time hypercube. (06 2022) 22

[67] Masud, S.M.R.A., Adiba, H., Hossain, T., Saha, A.K., Rahman, R.: Development of interactive data visualization system in three-dimensional immersive space. International Journal of Advanced Computer Science and Applications **14**(10) (2023) 22

[68] Chen, M., Lal, D., Yu, Z., Xu, J., Feng, A., You, S., Nurunnabi, A., Shi, Y.: Large-scale 3d terrain reconstruction using 3d gaussian splatting for visualization and simulation. ISPRS (2024) 23

[69] Aviyente, S., Frangi, A.F., Meijering, E., Muñoz-Barrutia, A., Liebling, M., Van De Ville, D., Olivo-Marin, J.C., Kovačević, J., Unser, M.: From nano to macro: An overview of the ieee bio image and signal processing technical committee. IEEE Signal Processing Magazine **40**(4) (2023) 61–71 23

[70] Kim, K., Chung, J.M., Lee, S., Jung, H.: The effects of electron beam exposure time on transmission electron microscopy imaging of negatively stained biological samples. Applied Microscopy **45** (09 2015) 150–154 23, 25

[71] Conesa, P., Fonseca, Y., Morena, J., Sharov, G., Rosa-Trevín, J., Cuervo, A., Mena, A., de Francisco, B., Hoyo, D., Herreros, D., Marchan, D., Střelák, D., Gimenez Fernandez, E., Ramírez-Aportela, E., Isidro-Gómez, F., Sánchez, I., Krieger, J., Vilas, J., del Caño, L., Sorzano, C.: Scipion3: A workflow engine for cryo-electron microscopy image processing and structural biology. Biological Imaging **3** (06 2023) 1–22 23, 25

[72] Wei, L., Hu, F., Chen, Z., Shen, Y., Zhang, L., Min, W.: Live-cell bioorthogonal chemical imaging: Stimulated raman scattering microscopy of vibrational probes. Accounts of Chemical Research **49**(8) (2016) 1494–1502 PMID: 27486796. 23, 26

[73] Pawley, J.: Handbook of biological confocal microscopy. Volume 236. Springer Science & Business Media (2006) 24, 26

[74] Razansky, D., Klohs, J., Ni, R.: Multi-scale optoacoustic molecular imaging of brain diseases. European Journal of Nuclear Medicine and Molecular Imaging **48**(13) (Dec 2021) 4152–4170 24, 26

[75] Gervasi, O., Ranon, R.: Guest editors' foreword to the special issue on virtual reality in scientific application foreword. Virtual Reality **13** (12 2009) 219–220 24

[76] Li, S.: VE-Suite: Coupling Visualization and Computational Environments to Support on-the-fly Engineering Design. PhD thesis (08 2003) 24, 26

[77] Kaufman, A., Mueller, K. In: Overview of Volume Rendering. Volume 7. (12 2005) 127–XI 27, 28, 29, 30, 31, 32, 33

[78] Duran, A., Duran, M., Masood, I., Maciolek, L., Hussain, H.: The additional diagnostic value of the three-dimensional volume rendering imaging in routine radiology practice. Cureus **11** (09 2019) 27

[79] Kaufman, A.E.: Volume visualization. ACM Comput. Surv. **28**(1) (mar 1996) 165–167 27, 29

[80] Meißner, M., Pfister, H., Westermann, R., Wittenbrink, C.: Volume visualization and volume rendering techniques. (01 2000) 27

[81] Kruth, J.P., Bartscher, M., Carmignato, S., Schmitt, R., Chiffre, L., Weckenmann, A.: Computed tomography for dimensional metrology. Cirp Annals-manufacturing Technology - CIRP ANN-MANUF TECHNOL **60** (12 2011) 821–842 28

[82] Yilmaz, o.: Seismic Data Analysis: Processing, Inversion, and Interpretation of Seismic Data. Society of Exploration Geophysicists (01 2001) 28

[83] Ebert, D.S., Musgrave, F.K., Peachey, D., Perlin, K., Worley, S.: Texturing and Modeling: A Procedural Approach. 3rd edn. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (2002) 28

[84] Chen, S.E., Williams, L.: View interpolation for image synthesis. SIGGRAPH '93, New York, NY, USA, Association for Computing Machinery (1993) 279–288 29

[85] Ljung, P., Krüger, J., Groller, E., Hadwiger, M., Hansen, C.D., Ynnerman, A.: State of the art in transfer functions for direct volume rendering. Computer Graphics Forum **35**(3) (2016) 669–691 33

[86] Kindlmann, G.: Transfer functions in direct volume rendering: Design, interface, interaction. (07 2002) 33

[87] Mota, A., Clarkson, M., Orvalho, L., Almeida, P., Matela, N.: Calculation of transfer functions for volume rendering of breast tomosynthesis imaging. (05 2020) 12 33

[88] Anthropy, A., Clark, N.: A Game Design Vocabulary: Exploring the Foundational Principles Behind Good Game Design. 1st edn. Addison-Wesley Professional (2014) 34

[89] Totten, C.W.: An Architectural Approach to Level Design. CRC Press (2014) 34

[90] Kushner, D.: Masters of Doom : how two guys created an empire and transformed pop culture. Random House (2003) 34

[91] Chang, Y., Aziz, E., Esche, S., Chassapis, C.: Virtual mechanical assembly training based on a 3d game engine. Computer-Aided Design and Applications **12** (03 2015) 119–134 34

[92] Smith, M., Queiroz, F.: Unity 5.x Cookbook. (10 2015) 34

[93] Schell, J.: The art of game design : a book of lenses. Amsterdam ; Boston : Elsevier/Morgan Kaufmann, [2008] ©2008 ([2008]) Includes bibliographical references (pages 477-479) and index. 35

[94] Logothetis, I., Sfyrakis, M., Vidakis, N.: Eduardo—unity components for augmented reality environments †. Information (Basel) **14** (Apr 2023) 4. 35

[95] Nesbit, P., Boulding, A., Hugenholtz, C., Durkin, P., Hubbard, S.: Visualization and sharing of 3d digital outcrop models to promote open science. GSA Today **30** (06 2020) 4–10 35

[96] Kern, F., Kullmann, P., Ganal, E., Korwisi, K., Stingl, R., Niebling, F., Latoschik, M.: Off-the-shelf stylus: Using xr devices for handwriting and sketching on physically aligned virtual surfaces. **2** (06 2021) 35

[97] Folk, M., Heber, G., Koziol, Q., Pourmal, E., Robinson, D.: An overview of the hdf5 technology suite and its applications. (03 2011) 36–47 38

[98] Schindelin, J., Arganda-Carreras, I., Frise, E., Kaynig, V., Longair, M., Pietzsch, T., Preibisch, S., Rueden, C., Saalfeld, S., Schmid, B., Tinevez, J.Y., White, D., Hartenstein, V., Eliceiri, K., Tomancak, P., Cardona, A.: Fiji: An open-source platform for biological-image analysis. Nature methods **9** (06 2012) 676–82 40

[99] HDFGroup: Hdfview usersguide. `https://www.scribd.com/document/606318594/HDFView-UsersGuide` (2019) 40

[100] Millman, K., Aivazis, M.: Python for scientists and engineers. Computing in Science & Engineering **13** (05 2011) 9 – 12 40

[101] Marschner, S., Shirley, P.: The graphics pipeline. In: Fundamentals of Computer Graphics. AK Peters/CRC Press (2021) 177–204 41

[102] Technologies., U.: Render pipelines. `https://docs.unity3d.com/Manual/render-pipelines.html` (2024) 42

[103] Technologies., U.: Unity user manual. `https://docs.unity3d.com/2023.2/Documentation/Manual/UnityManual.html` (2024) 43, 47

[104] Technologies., U.: Vr development in unity. `https://docs.unity3d.com/Manual/VROverview.html` (2024) 44

[105] Technologies., U.: Unity xr. `https://docs.unity3d.com/Manual/XR.htmll` (2024) 45

[106] Technologies., U.: Xr interaction toolkit. `https://docs.unity3d.com/Packages/com.unity.xr.interaction.toolkit@3.0/manual/index.html` (2024) 45

[107] Technologies., U.: Openxr plugin. `https://docs.unity3d.com/Packages/com.unity.xr.openxr@1.11/manual/index.html` (2024) 46

[108] Pimax: Pimax 8k x. `https://pimax.com/products/outlet-8k-x-smas` (2019) 50

[109] Corporation, V.: Valve index controllers. https://store.steampowered.com/app/1059550/Valve_Index_Controllers/ (2019) 54

[110] Corporation, H.: Steamvr base station 2.0. https://www.vive.com/eu/accessory/base-station2/ (2019) 54

[111] Rallis, J., Kapai, G., Pavlopoulos, A.: 16. In: Handbook of Marine Model Organisms in Experimental Biology: Established and Emerging. CRC Press (2021) 70

[112] Tomer, R., Khairy, K., Amat, F., Keller, P.J.: Quantitative high-speed imaging of entire developing embryos with simultaneous multiview light-sheet microscopy. Nature Methods **9**(7) (Jul 2012) 755–763 70

[113] Royer, L.A., Lemon, W.C., Chhetri, R.K., Wan, Y., Coleman, M., Myers, E.W., Keller, P.J.: Adaptive light-sheet microscopy for long-term, high-resolution imaging in living organisms. Nature Biotechnology **34**(12) (Dec 2016) 1267–1278 70

[114] Amat, F., Höckendorf, B., Wan, Y., Lemon, W.C., McDole, K., Keller, P.J.: Efficient processing and analysis of large-scale light-sheet microscopy data. Nature Protocols **10**(11) (Nov 2015) 1679–1696 70

[115] Hörl, D., Rojas Rusak, F., Preusser, F., Tillberg, P., Randel, N., Chhetri, R.K., Cardona, A., Keller, P.J., Harz, H., Leonhardt, H., Treier, M., Preibisch, S.: Bigstitcher: reconstructing high-resolution image datasets of cleared and expanded samples. Nature Methods **16**(9) (Sep 2019) 870–874 70

[116] Lavik, M.: Unityvolumerendering. https://codeberg.org/matiaslavik/UnityVolumeRendering (2019) 74