2019

A method for viewing and interacting with medical volumes in virtual reality

Jordan King Williams
Iowa State University

A method for viewing and interacting with medical volumes in virtual reality

by

Jordan Williams

A thesis submitted to the graduate faculty

in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Co-majors: Human Computer Interaction; Computer Engineering

Program of Study Committee:
Eliot Winer, Major Professor
Adarsh Krishnamurthy
James Oliver

The student author, whose presentation of the scholarship herein was approved by the program of study committee, is solely responsible for the content of this thesis. The Graduate College will ensure this thesis is globally accessible and will not permit alterations after a degree is conferred.

Iowa State University

Ames, Iowa

2019

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF JOURNAL FIGURES

LIST OF TABLES

ABSTRACT

The medical field has long benefited from advancements in diagnostic imaging technology. Medical images created through methods such as Computed Tomography (CT) and Magnetic Resonance Imaging (MRI) are used by medical professionals to non-intrusively peer into the body to make decisions about surgeries. Over time, the viewing medium of medical images has evolved from X-ray film negatives to stereoscopic 3D displays, with each new development enhancing the viewer's ability to discern detail or decreasing the time needed to produce and render a body scan. Though doctors and surgeons are trained to view medical images in 2D, some are choosing to view body scans in 3D through volume rendering. While traditional 2D displays can be used to display 3D data, a viewing method that incorporates depth would convey more information to the viewer. One device that has shown promise in medical image viewing applications is the Virtual Reality Head Mounted Display (VR HMD).

VR HMDs have recently increased in popularity, with several commodity devices being released within the last few years. The Oculus Rift, HTC Vive, and Windows Mixed Reality HMDs like the Samsung Odyssey offer higher resolution screens, more accurate motion tracking, and lower prices than earlier HMDs. They also include motion-tracked handheld controllers meant for navigation and interaction in video games. Because of their popularity and low cost, medical volume viewing software that is compatible with these headsets would be accessible to a wide audience. However, the introduction of VR to medical volume rendering presents difficulties in implementing consistent user interactions and ensuring performance.

Though all three headsets require unique driver software, they are compatible with OpenVR, a middleware that standardizes communication between the HMD, the HMD's controllers, and

VR software. However, the controllers included with the HMDs each has a slightly different control layout. Furthermore, buttons, triggers, touchpads, and joysticks that share the same hand position between devices do not report values to OpenVR in the same way. Implementing volume rendering functions like clipping and tissue density windowing on VR controllers could improve the user's experience over mouse-and-keyboard schemes through the use of tracked hand and finger movements. To create a control scheme that is compatible with multiple HMD's A way of mapping controls differently depending on the device was developed.

Additionally, volume rendering is a computationally intensive process, and even more so when rendering for an HMD. By using techniques like GPU raytracing with modern GPUs, real-time framerates are achievable on desktop computers with traditional displays. However, the importance of achieving high framerates is even greater when viewing with a VR HMD due to its higher level of immersion. Because the 3D scene occupies most of the user's field of view, low or choppy framerates contribute to feelings of motion sickness. This was mitigated through a decrease in volume rendering quality in situations where the framerate drops below acceptable levels.

The volume rendering and VR interaction methods described in this thesis were demonstrated in an application developed for immersive viewing of medical volumes. This application places the user and a medical volume in a 3D VR environment, allowing the user to manually place clipping planes, adjust the tissue density window, and move the volume to achieve different viewing angles with handheld motion tracked controllers. The result shows that GPU raytraced medical volumes can be viewed and interacted with in VR using commodity

hardware, and that a control scheme can be mapped to allow the same functions on different HMD controllers despite differences in layout.

CHAPTER 1: INTRODUCTION

Uses of Medical Imaging Technology in the Field

Doctors and surgeons often use medical images to diagnose patients and plan medical procedures [1, 2, 3]. Medical images including X-rays, computed tomography (CT), and magnetic resonance imaging (MRI) are useful because they provide a non-intrusive view inside the body. While a single X-ray image can give one view of the body, CTs and MRIs capture series of images in slices to produce volumetric images.

Doctors commonly use X-rays to identify bone fractures . MRI and CT scans are also used for radiation therapy cancer treatments, which require beams of radiation to be shaped in 3D in order to avoid damaging healthy tissue [4]. Surgeons use 3D medical images to plan a wide variety of procedures including liver [5] and craniofacial [6] surgeries. Medical imaging technology gives crucial insight to medical professionals and has led to the development of more effective treatments and more accurate patient diagnoses.

X-rays and early MRI and CT scans were initially viewed in 2D, meaning that doctors could only view a 2-dimensional subset of the volume data at a given time. Functions like contrast adjustment, coloring, and image scaling were used to make tissue types more distinct and examine areas of interest more closely [7]. Procedures requiring 3D positioning could still be planned by using multiple 2D views from different angles. With volumetric data, however, computers could be used to recreate image planes in any orientation [1]. 2D planar reconstruction still does not convey volume to the viewer, which is helpful when an understanding of the 3D structure of the body is desired. Doctors have even used 3D printed body scans for this purpose [8], as they can be manually rotated and moved to achieve any

viewing angle. Physical models may provide more information than a static image, but cannot be changed without repeated printing or sculpting, and fine detail is limited by the resolution of the 3D printer. These problems could be solved by displaying the body scan in 3D and in real-time, something made possible through volume rendering.

Volume Rendered Medical Images

Volume rendering is used to project volumetric data into a 2D image, and provides a better picture of 3D structures in a body scan than a single slice. Volume rendering methods typically involve using 3D geometry to represent a stack of body scan slices, or by simulating light interacting with the volume as seen by the virtual camera [9, 10, 11].Volume rendering is more computationally intensive than 2D slice viewing, but these rendering methods were created to render volumes at high framerates in the interest of real time interaction. Volume rendering has been demonstrated as effective for visualizing CT angiographies [12, 2] and effective for diagnosing tumors [2, 13]. Where doctors viewing 2D images use functions like contrast adjustment, volume rendered scans can be “clipped” to expose internal structures, and “windowed” to filter out or segment tissue types of interest.

VR HMDs

With the ability to render volumes at high framerates and expose different tissue types, a more immersive viewing and interaction method is desired as well. 3D viewing methods like 3D displays and CAVE systems have been implemented for interactive applications, including medical volume viewing [11]. Both methods offer increased immersion over a 2D display through stereoscopy and motion tracking. However, 3D displays limit the movement of the user, and CAVEs are impractical for most users due to their size and cost.

VR HMDs also allow for viewing of 3D images with depth information provided through stereoscopy. HMDs give the viewer the impression that they are inside of a 3D environment by occupying their entire field of view and using the viewer's head movements to recalculate the 3D scene. Because of this, HMDs have been demonstrated to give users a higher sense of immersion in [14]. Current headsets like the Oculus Rift [15], HTC Vive [16], and Samsung Odyssey [17] include screens with higher resolutions and refresh rates, better motion tracking, and are offered at a lower price than previous headsets. They also include motion-tracked handheld controllers designed for immersive applications like VR video games.

Similarly to CAVEs, the increased immersion offered by HMDs can also affect the user negatively through motion sickness. Some aspects of VR software that contribute to motion sickness are low framerates and unresponsive motion tracking, both potentially causing conflicts in the user's physical and visual senses of motion [14, 18, 19]. This leads to software requirements of high framerates and the avoidance of unnatural movements.

VR Controllers

As the primary interaction method for most computers, the mouse and keyboard are often used to control volume rendering software and interact with the volume. Viewers using a traditional display might wish to control multiple input values at once, something that is difficult using a mouse and keyboard. Gamepads like the Microsoft Xbox [20] and Nintendo Wii [21] controllers have been used instead [22], as they contain multiple buttons and joysticks that can be manipulated by one hand simultaneously. A gamepad is especially helpful to a viewer using a VR HMD, as they are likely to be moving around. Mouse-and-keyboard control schemes typically require the user to remain stationary, while gamepads are handheld and often wireless.

VR controllers offer the buttons and joysticks of a traditional gamepad with added motion tracking and a layout designed to be used without sight of the hands [14]. Controls can be assigned to movements, so physical gestures can be used for interactions like placing clipping planes and moving the volume. They have also been demonstrated as more effective in 3D manipulation tasks than non motion-tracked controllers [23]. However, the controllers for commodity HMDs are different. This presents difficulties for implementing viewing and interaction controls across the breadth of devices currently available and those planned for the future.

The work presented in this thesis addresses concerns of real-time volume rendering, mitigation of motion sickness in an HMD, and creating a consistent control scheme despite differences between HMD controllers. The methods developed are demonstrated in an application that renders medical volumes on current VR HMDs and uses VR controllers to interact with the volume by translating, rotating, clipping, and windowing.

Thesis Organization

This thesis first outlines concepts regarding computer graphics, medical imaging, and virtual reality that informed the development of a VR volume rendering and interaction methods. The software and hardware used is listed next, followed by a methodology of development of an in the form of a journal paper. Finally, the result is summarized and some possible improvements for a future project are discussed.

CHAPTER 2: BACKGROUND

Graphic Rendering Techniques

Producing computer graphics requires a series of operations to visually represent numerically defined objects. Most modern displays used to display these visual representations are made of millions of pixels, where the color and brightness of each pixel is manipulated independently to create an image. In the physical world, the eyes and brain work together to understand what is being looked at based on how light interacts with objects in the field of view and light receptors in the eye. For this reason, most rendering techniques attempt to simulate the behavior of light rays and the properties of the materials with which they interact.

One category of rendering techniques makes use of raytracing, where virtual light sources are described, and pixel color and brightness are found by simulating the interactions of light rays reaching the virtual eye. Whitted presented raytracing as an improved model for simulating light sources located within a scene [24] compared to the previous Phong shading model [25]. Through raytracing, simulated light rays reflect from and refract through geometric objects in the scene based on their material properties. This also applies to rendering volumetric objects, where raytracing can be used to display the intensities of light traveling through different areas of the volume rather than simulating how light interacts with the surfaces. Some 3D medical image viewing applications render CT and MRI images in this manner. Rather than simulating light bouncing off the volume, rays are cast out from camera, one per screen pixel, traveling through the volume until they terminate at the non-visible side. As each ray travels through the volume, samples of the volume's density are taken in small increments. Illustrated in Figure 1, these

samples are translated from density to color based on opacity and tissue transfer functions and accumulated into a final pixel value that is displayed on the screen [9, 11].

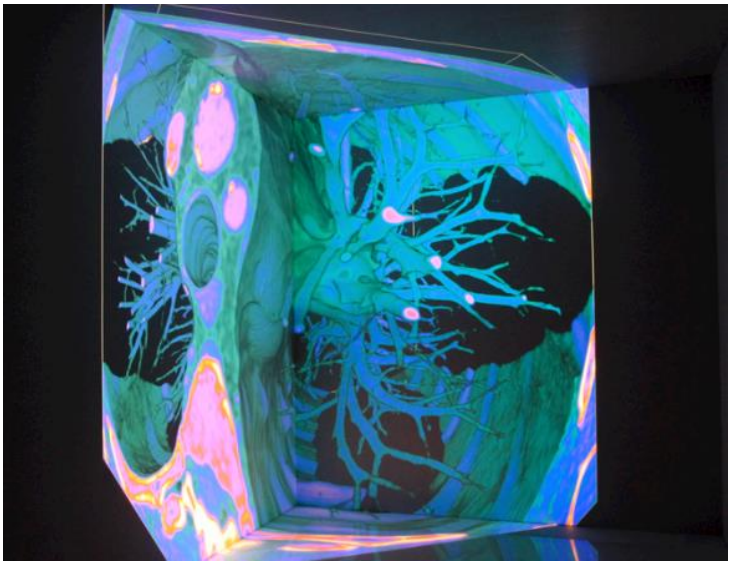

Figure 1: Raytraced volumetric chest cavity, courtesy of [11]

While raytracing can be used to produce accurate images, realistic light simulation is a computationally expensive process. For offline rendering of still images or video animations, it is suitable for renders to take hours or days to compute a single frame. Conversely, real-time rendering for games and other interactive applications generally requires 25 or more complete renders every second [14] with higher framerates desired for improved interaction. To fulfill this requirement, new techniques were developed with simplified lighting models, one of which is called rasterization. Instead of illuminating pixels by simulating light rays, sets of screen pixels, or fragments, occupied by each geometry element are found based on the object's position relative to the camera. These sets of screen pixels are often shaded based on ambient lighting and the angle between discrete lights and the geometry element [25]. Graphics libraries like OpenGL [26] were developed to facilitate rasterization, and computer graphics hardware has evolved

along with graphics libraries to streamline the rasterization process. Modern GPUs contain thousands of smaller processing units dedicated to performing operations in parallel, something useful when millions of pixel values are being shaded using repetitive algorithms. Rasterization was developed to render objects defined by geometry, but because of the inclination towards rasterized rendering in modern hardware, some rendering techniques use rasterization instead of raytracing to simulate volumetric objects at high framerates [9].

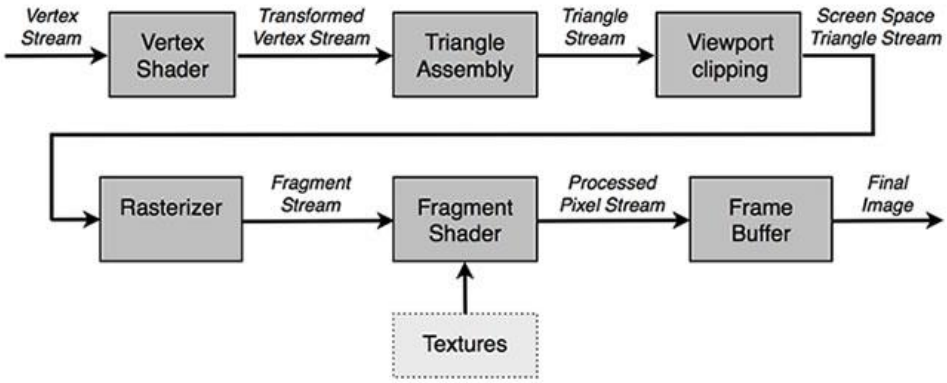

Figure 2: The programmable OpenGL Pipeline. Image from adobe.com/devnet/flashplayer/articles/vertex-fragment-shaders.html

The OpenGL graphics pipeline has become more programmable over time, now allowing programmers to write the code used to render each element of geometry in a scene. These programs, called shaders, are compiled and run on the GPU. For rasterization, it is common to write a vertex shader and a fragment shader. The relationship between the vertex and fragment shader stages is illustrated in Figure 2. The former applies a series of transformations to translate vertices from object space to screen space, and the latter applies color to the pixels identified during the rasterization stage as being occupied by geometry. One aspect of the OpenGL shader structure that is important to note is the fact that the fragment shader is a program that essentially runs in parallel on each pixel being rendered. It is possible to utilize the parallel nature of the

fragment shader and the GPU to render raytraced volumes faster than what is possible on the CPU [11, 27], and as a result raytracing algorithms have been able to run much faster on newer GPUs than CPUs.

Medical Imaging

Medical imaging is a term that can refer to two different classes of methods used to obtain images of the body. One relies on visible light and includes visual inspection and simple photography. The other creates a visible image from non-visible phenomena, including X-ray, CT, and MRI scans. As volumetric body scan rendering is desired, the latter will be the focus of this thesis.

2D medical imaging

X-ray images show the result of x-rays shot through the subject of the scan interacting with the subject before being detected after passing through. While visible light is mostly blocked by the skin and soft tissue, x-rays pass through most body structures except for bones. After passing through the subject the x-ray beams are detected using either film or a digital detector, creating an image showing a gradient of areas where x-ray beams could pass through the body. CT scans utilize x-rays, but collect multiple measurements rotating around the body, building tomographic “slices” of the body. These slices contain values regarding a tissue’s ability to transmit x-rays, and tissues in the image can be classified depending on previously recorded CT values. MRI creates similar image slices, instead placing the patient into a magnetic field, generating and sending radio wave pulses through the body, and examining the magnetic response of the tissue’s protons [1]. The values recorded are primarily influenced by proton density, from which tissue density can be inferred. As with CT, magnetic properties of different

tissue types are known and can be used to identify regions in the resulting MRI scan. Examples of x-ray, CT, and MRI scans are pictured in Figure 3.

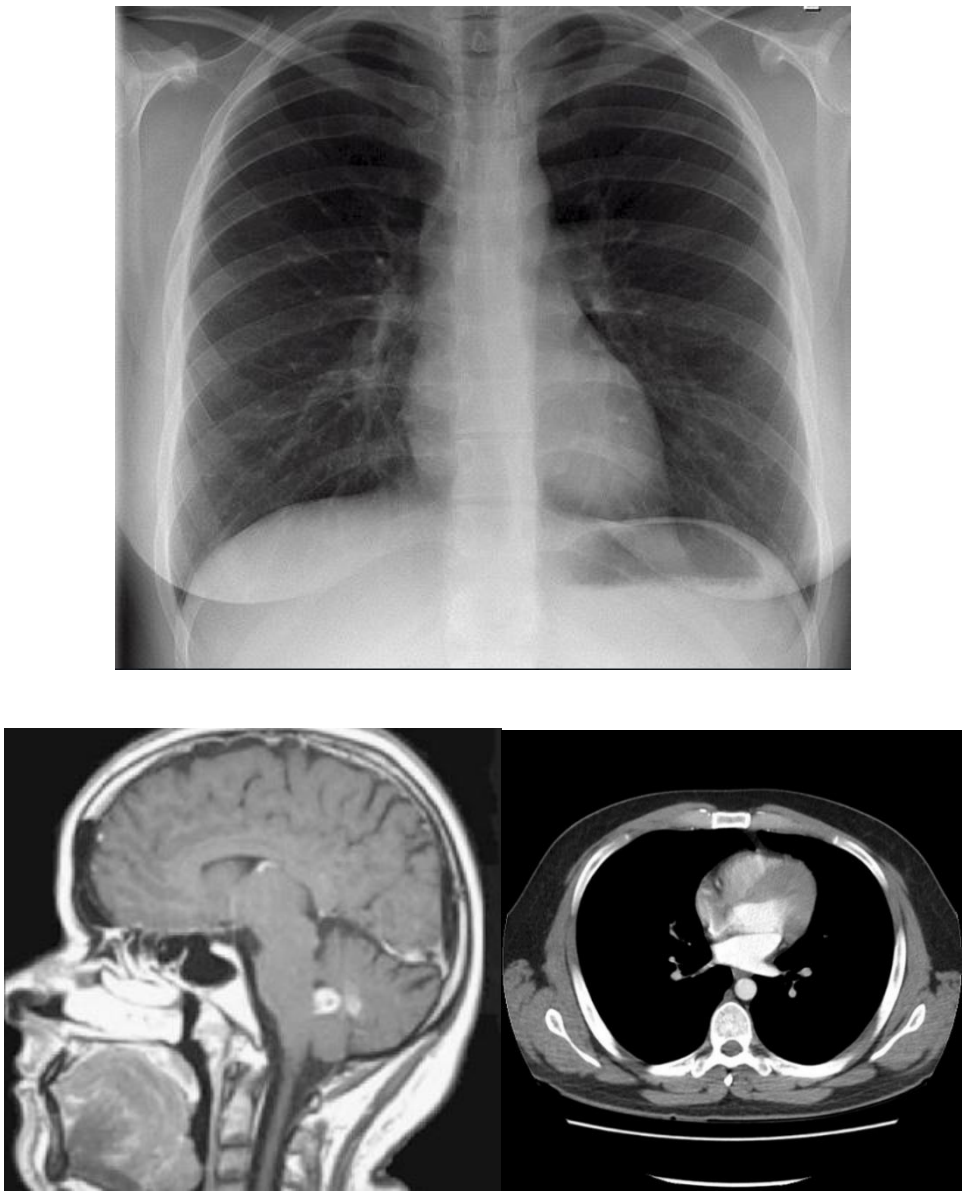

Figure 3: Clockwise from top: X-ray of chest, CT scan of chest, and MRI of head

To store and recall medical images digitally, a container file format is needed. To simplify the process of sharing files between radiologists and doctors with different equipment, the American College of Radiology and National Electrical Manufacturer's Association jointly

developed the Digital Imaging and Communications in Medicine (DICOM) standard [28, 29]. DICOM specifies the formatting of image slice data and patient metadata in a similar manner to existing media file formats such as MPEG and JPEG. An example chest cavity CT scan with a resolution of 512 x 512 x 256 would consist of 256 DICOM files, each containing patient information and a compressed 512 x 512 grayscale image representing a slice of the chest.

Many 2D medical image viewing applications exist for x-ray, MRI, and CT scans. Some of these include Osirix [30] and RadiAnt [31]. 2D rendering requires software capable of loading images and performing image processing operations like contrast adjustment, scaling, and rotation. Current computers are easily able to view and manipulate 2D scans, and can even load hundreds of slices into memory, enabling the viewer to traverse through an entire scan quickly. Also common is the use of a measuring tool, where the distance between two user-selected points is calculated [32].

HMDs provide immersive stereoscopic viewing which has been shown to positively affect users identifying tumors and fractures [13, 1]. Hands-free control of medical imaging software has been implemented using the Leap Motion hand tracker [33], Microsoft Kinect [34], and with a combination of the Leap Motion and Oculus Rift HMD [35]. Both [33] and [34] experimented with hand tracking control methods that could be suitable for VR, but did not utilize a VR viewing method as flat displays were used in both cases. Furthermore, the rendered medical images were 2D slices either read from a DICOM set directly or reconstructed in an off-axis plane. This means that the user receives no depth information about the body scan. Reinschluessel's [35] virtual operating room conveyed a full VR experience as the user's environment was fully constructed in 3D, but the scans displayed were 2D as well. The control

abilities given to the user in all cases included translation and rotation of the image, pointing and clicking regions of interest, and animating frames of the scan. These controls were enabled though hands-free tracking methods, which are useful for surgeons who need to remain sterile during a procedure. While planning a surgery or diagnosing a patient, however, hands-free tracking is unnecessary. A control method that uses physical controls like buttons, triggers, and joysticks while retaining full hand tracking would likely improve user experience for medical volume interaction.

2D medical image viewing and interaction has been implemented in VR, but without volumetric 3D rendering the stereoscopic capability of HMDs is underutilized. Though more computationally intensive, volume rendering of medical scans has recently become more common due to rendering techniques adapted to achieve real-time performance as computer hardware has improved.

Volume rendering methods

An early technique developed to render medical volumes, called “texture slicing”, simulates volume by substituting geometric data [9, 36]. A simple implementation of texture slicing involves creating a series of quadrilateral polygons, or quads, to represent slices of a body scan. The images from the scan are loaded into 2D textures and applied to each quad. The quads are rendered through rasterization but give the appearance of a volumetric structure due to their stacked arrangement. With static texture slices, assuming a viewing angle perpendicular to the plane normals will show gaps between slices. To fix this, slices must be recomputed in two different ways, each minimizing the angle between the camera and the textured planes of the volume. For one method, a set of textures is generated for each axis, and the set is chosen that

makes the shallowest angle with the camera. This produces a noticeable change in the image when the set of textures are switched, but after the recomputed slices are generated, no further recomputation is necessary. The other method requires a set of textures to be recomputed every time the camera's view changes. In this case, all slices are aligned to face the camera. This method produces images with less artifacts than alternative texture slicing methods, but comes with a performance cost for VR, where the view is expected to change every frame.

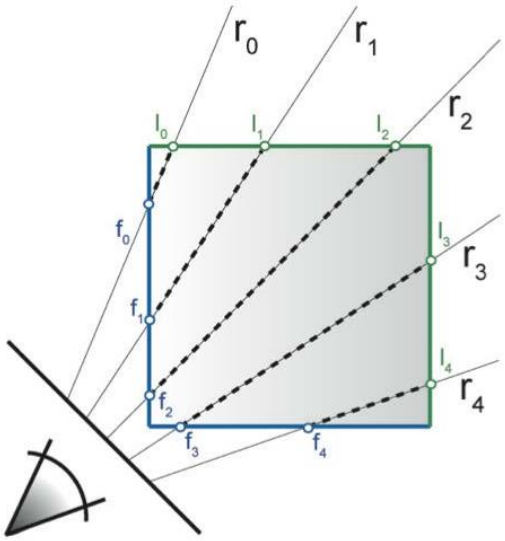

Figure 4: Diagram of ray computation for volume raytracing, courtesy of [37]

Another technique, iso-surface rendering, renders volumetric data as a geometric surface. Boundaries between the inside and outside of segments of the volume are determined and each region of the boundary is substituted for geometric primitives [10]. These boundaries are commonly found through a marching cubes algorithm or by raytracing. Iso-surface rendering makes the surface of the volume distinct and allows the volume to be lit using geometric shading models like Phong shading. However, since only surfaces of the volume are rendered, transparency cannot be used to mimic volumetric data.

Additionally, image splatting has been used to project a volumetric data set like that of a body scan onto a 2D image plane [38, 39]. This involves "splatting" each point onto a plane in the view direction. Each "splat" takes the form of a 3D kernel, often a Gaussian distribution as it approximates a volumetric sphere. Points can be weighted by density and appropriately colored and sized before accumulating, allowing for detailed renders and transparency effects not possible with iso-surface rendering. However, the order of point projection is important, as points near the viewer can be occluded if they are not "splatted" after points located behind them. Some splatting algorithms suffer from artifacts as a result. There are culling methods that eliminate non-visible points before they are projected, but this requires constant processing if the view is continuously changing.

As CPU and GPU power have increased, raytracing volume rendering methods have been considered more feasible for real-time VR. Shooting rays through a volume to simulate interactions encountered by light is considered to accurately mimic how the volume would be seen by the eye. The problem is then determining where each point on the ray lies with respect to the volume. Assembling tomographic slices from CT or MRI scans into a cuboid structure yields a three-dimensional dataset comprised of 3D pixels, or "voxels." Voxels are arranged in a grid analogous to pixels in a stacked set of images. Each voxel represents a volume of between 0.5mm and 2mm, depending on the scanning method, creating a grid of 3D "cells" in which voxel data values are centered. Rays traveling through the volume intersect with the voxels, but the sampled intersection points along the rays lie between voxel centers. For accurate rendering, the values sampled from the ray must be interpolated between cells in the neighborhood of the sampling point. This adds computing overhead to the already intensive sampling process.

The shear-warp algorithm was developed as a faster CPU-based alternative to traditional raytracing for volumetric rendering [10]. One area in which shear-warp improves rendering performance is voxel interpolation. Instead of shooting a ray through the volume from the angle of the eye, the volume is rotated so that the ray direction vector from the eye aligns with one of the volume's axes. In addition, the sampling interval is set to the thickness of the volume's slices. To account for the rotation, the slices are sheared, like a stack of paper. The shearing adjustment brings the alignment of the volume close to the original pre-rotation alignment. This process ensures that there is exactly one sample per volume slice, eliminating the need for interpolating between slices. The resulting image is then "warped" to counteract the distortion from shearing. This procedure is illustrated in Figure 5.A positive effect of using the shear-warp method is a framerate increase due to reduced interpolation operations, but the result of limiting sampling in the forward vector to one per slice is effective nearest-neighbor interpolation. A "stepped" appearance is characteristic of this method, where curved surfaces in the volume have ridges where the surface "jumps" between volume slices. Another drawback of shear-warp rendering is increased memory requirements compared to traditional raytracing. The speed of ray sampling is increased by using run-length encoding, where voxels are addressed in the order encountered by a ray cast from front-to-back through the volume. Run-length encoding necessitates three sets of volume data to be encoded, because depending on the perspective angle of the camera, one of the three primary viewing axes must be selected for aligning the volume. A tripled memory requirement would impact the ability to implement the algorithm on a large number of GPUs. While many current mid-range GPUs contain more than 3GB of video memory, an example 512^3 dataset may occupy 1GB, increasing to 3GB and beyond for shear-warp encoding.

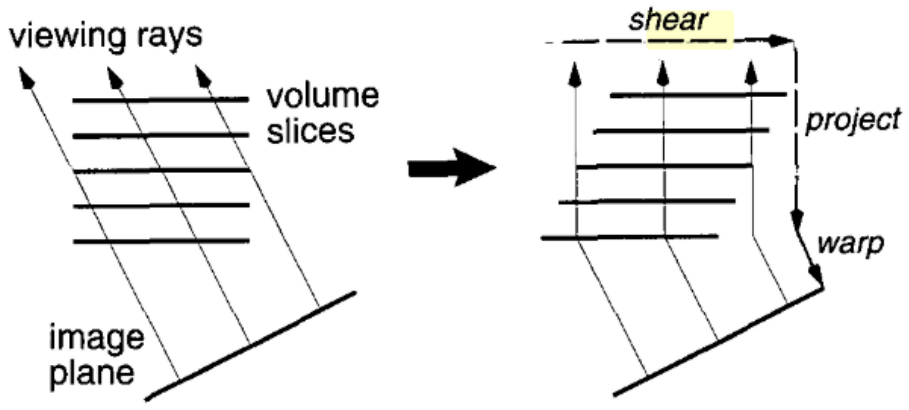

Figure 5: Diagram of shear warp procedure, courtesy of Lacroute [20]

Fortunately, GPUs contain hardware dedicated to highly parallel procedures used for vertex and fragment shading operations. GPUs also use hardware to accelerate texture interpolation, including both 2D and 3D textures. A common GPU raytracing approach takes advantage of OpenGL's linear 3D texture filtering to reduce overhead while sampling the volume, smoothing the resulting image and improving raytracing performance without requiring multiple reencoded datasets. Performance is also improved over CPU raytracing through distribution of the ray sampling across the hundreds or thousands of processors on the GPU die. This form of GPU raytracing is used to achieve interactive framerates in medical volume rendering applications such as Osirix, RadiAnt, 3D Slicer [40], and VIPRE [11, 38].

Virtual Reality

As technological advancements in real-time graphics and medical imaging have proceeded, the desire for a more immersive way of viewing and interacting with medical images has emerged. Virtual reality has become a topic of great interest of late, with an increase in consumer-oriented VR devices and a push for greater VR compatibility in video games, mobile applications, and streaming videos. Rendering 3D environments in VR presents multiple

challenges, with requirements ranging from high framerate video output, to high resolution, and high frequency motion tracking. Recent developments in mobile phone technology have allowed these requirements to be met at a lower cost than in previous years, but the underlying techniques have been in practice for some time.

History of VR Viewing Methods

Figure 6: Acer 3D display and Nvidia 3D shuttering glasses

Perception of depth in a 3D image is achieved by sending images rendered from different perspectives to each eye, mimicking differences in convergence at varying depths. For this, there are two conventional approaches. One method involves encoding multiple images into a single viewing target, such as a computer monitor, while another requires the eyes to focus on separate targets, such as an HMD.

Various displays require the viewer to wear specialized glasses and function by displaying left and right eye images simultaneously, or alternately in quick succession [9].The first scheme requires the images to be polarized differently for each eye and interlaced. Interlaced displays are typically comprised of rows of pixels covered by filters that are polarized perpendicularly to each other. Glasses are worn by the viewer with alternately polarized lenses

so that each eye only sees half of the interlaced pixel rows. This method effectively halves the vertical resolution of the stereoscopic image. For the second scheme, the display alternates between showing full frames of the left and right eye images. Without glasses, a viewer would see a blurry image containing both left and right views. Instead, the user wears glasses that "shutter" each eye in synchronization with the display. When the display shows the left eye's view, the viewer's right eye lens becomes opaque, switching with the right eye's view and the left eye lens on alternating frames. Stereoscopic 3D displays were compared with monoscopic displays in a study [41] testing the effect of stereopsis cues on users viewing medical volumes. Users scored significantly higher on a relative position estimation task when using the stereoscopic display than when using the monoscopic display.

Stationary 3D displays like the example in Figure 6 attempt to convince the brain that the viewer is looking at an actual 3D object, but the effect is lost when the user moves their head or looks at the display from an off-axis position due to distortion. Some VR applications compensate for an off-axis viewing angle using head tracking, sending head positions to the application to update the camera position in real-time. This increases immersion and lets the viewer move their head without distorting the image, but since the display remains stationary, the viewer's range of movement is limited. Furthermore, the viewer's field of view is limited by the size of the display and viewing distance.

Pictured in Figure 7, CAVE Automatic Virtual Environments (CAVEs) improve upon single 3D display applications by surrounding the viewer with multiple large (wall-sized) 3D displays, usually between four and six [11, 42, 43]. Head tracking is also used to render a view for each screen that follows the viewer's movement. One advantage of a CAVE over a single 3D

display is that nearly the entire field of view is occupied by a real-time stereoscopic image. The viewer can move around within the CAVE and look in any direction. CAVEs typically use projectors, and the large surface area of the projected surface makes it possible to render the virtual environment at a high resolution. However, the increased resolution significantly increases the graphical power needed to render a real-time interactive environment. This is especially apparent when rendering volumes using raytracing, as an increase in pixels occupied by the volume directly increases the number of rays that must be traversed, sampled, and interpolated.

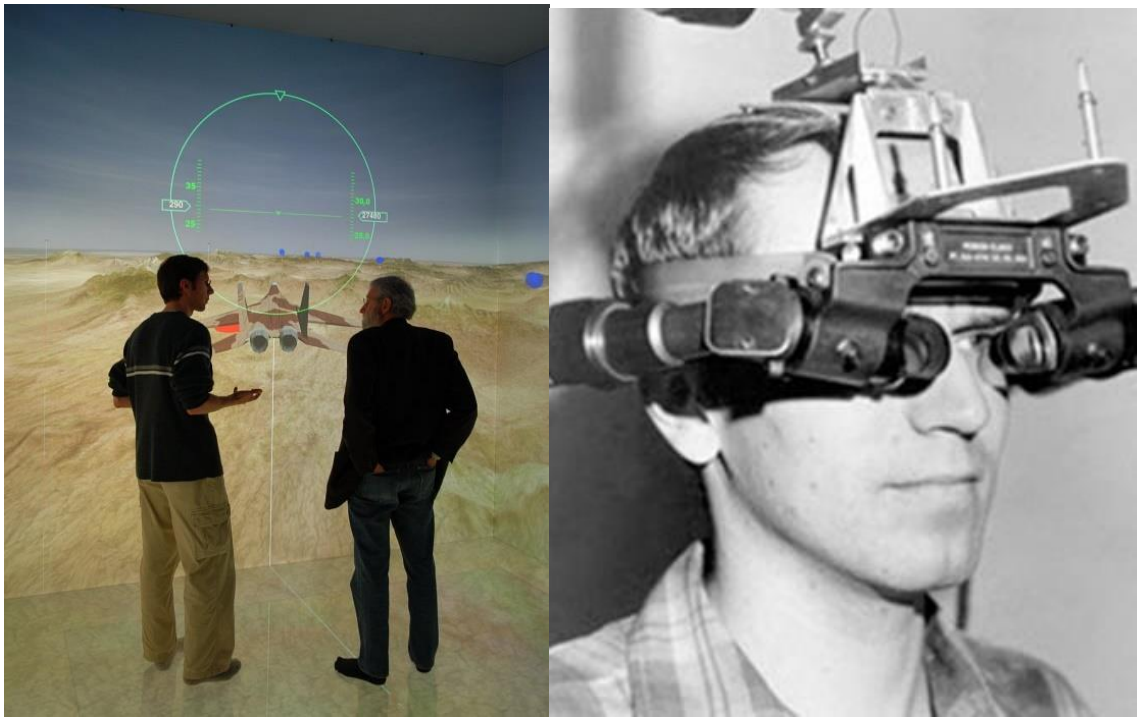

Figure 7: Iowa State University's C6 CAVE (left), Ivan Sutherland's Sword of Damocles HMD (right)

In contrast, HMDs require the viewer's eyes to focus on separate displays, or separate regions of the same display, by being worn on the viewer's head. The displays are often mounted

within centimeters of the eye, but lenses are placed before the display that allow the eye to focus much farther away. Early HMDs employed varying levels of tracking. Some, like Ivan Sutherland's Sword of Damocles [44] also seen in Figure 7, mechanically tracked a user's head position and orientation to render a primitive scene according to their gaze. Other headsets aimed primarily at viewing traditional media, such as the Sony HMZ-T1, had no built-in tracking. More recently, headsets like the early Oculus developer kits and the Samsung Gear VR [45, 1] emerged that track the orientation of the user's head, but not its position. If no external tracking method is used, the user is expected to remain stationary, rotating their head to look around. Figure 8 shows the variety of commodity headsets currently available. Each pictured headset along with their controllers is compatible with the medical volume viewer described in this thesis.

Current HMDs

Figure 8: From top-left proceeding clockwise, Oculus Rift, HTC Vive, Lenovo Explorer, HP headset, Del Visor, Samsung Odyssey, Acer headset (center)

Consumer versions of the Oculus Rift and HTC Vive were first released in 2016. The Oculus Rift uses a camera-based positional tracking method, where infrared LEDs are arranged on the HMD and controllers and recorded by two or more cameras. The HTC Vive headset and controllers contain sensors which pick up signals from rotating light emitters mounted around the tracked area. Device position is determined by measuring the time of sensor light acquisition and comparing to the world position and angle of the emitter. On both headsets, this external tracking data is combined with internal accelerometers and gyroscopes mounted on the headset to increase accuracy and fill in movement for times where external tracking is briefly lost.

The Oculus Rift and HTC Vive HMDs officially support hand tracked motion controllers designed by their respective developers. In addition, both HMDs support the OpenVR SDK. OpenVR is most commonly implemented in Valve's SteamVR, which is compatible with major game engines including Unity3D and Unreal Development Kit.

A collection of HMDs created for Microsoft's Windows MR (mixed reality) platform was released by several major manufacturers including Dell, Asus, Samsung, Acer, and HP. Microsoft provides an SDK for development, but Windows MR headsets are also compatible with OpenVR. Windows MR headsets vary in hardware appearance and display resolution, but have several specifications in common. All Windows MR headsets use inside-out positional tracking achieved through SLAM (simultaneous localization and mapping). Additionally, the headsets include controllers resembling a combination of those developed for the Rift and Vive.

The implementation described in this thesis was developed using the Samsung Odyssey HMD. It was chosen over the other HMDs for its high-resolution display, but the VR volume renderer is compatible with any headset supporting OpenVR.

VR Controllers

Traditional mouse/keyboard-based control schemes are ubiquitous in general computing, and they are commonly used for viewing 3D volume rendered data. They require values used for rendering, such as clipping plane position and window densities, to be manually typed into fields one-by-one. Real-time control of multiple parameters is difficult, given that there are only 2 available axes of movement while using the mouse. Use of a gamepad, such as an Xbox 360 [20] controller, increases the number of input axes and positions controls in a way that allows multiple axes to be manipulated with a single hand. The gamepad's dual joysticks both move along 2 axes, and an additional axis of movement is added by each trigger button. Commonly, one joystick is mapped to translation, and the other mapped to rotation. The trigger buttons may act as modifiers for joystick movement or controls for a separate function. The gamepad increases the level of control afforded to the user's hands but lacks position and orientation tracking. The Nintendo WiiMote [43, 21] has also been proposed as a VR controller because of its tracking provided by infrared light sensors, gyroscopes, and accelerometers.

A study comparing low cost input devices found that though the mouse and keyboard were measured to be the most efficient while moving virtual objects, users preferred the Xbox 360 Gamepad over both the mouse and keyboard and the WiiMote [22]. It was speculated that this is because of the users' familiarity with traditional gamepads, and the absence of advanced controls on the WiiMote such as analog joysticks and triggers. These results suggest that a

controller that combines the physical control ability of a traditional gamepad with the motion tracking offered by the WiiMote and others could prove effective for VR applications. Another paper outlines the use of HMD medical volume viewing to diagnose patients [46]. A non-tracked handheld controller was used to adjust tissue density windowing and zoom and rotation of the volume. This configuration was effective because it moved useful controls from the mouse and keyboard to a single handheld device that the user can operate without seeing their hands. Without hand tracking, however, movement of the volume or clipping planes cannot be achieved with the user's natural hand movement. For this, a handheld controller that is fully motion tracked with analog controls would be desired for use with VR headsets.

The advent of VR HMDs introduced a need for tracking a user's hand movements in 3D space and accepting button input simultaneously. Operation of a traditional gamepad, even if tracked in 3D, requires both hands to be positioned close together. To design a control method more fitting for VR, several 3rd party manufacturers proposed controllers designed for use in VR HMDs, such as the Sixense [47] STEM and Razer Hydra [48, 14]. As HMDs became more mainstream, manufacturers began to offer their own control solutions designed to work in a similar manner. The HTC Vive includes two "wand" style controllers, and Oculus offers the "touch" hand controller. In [23] HTC Vive wands were compared to hand-only tracking methods like the Leap Motion for users performing movement, selection, and grabbing tasks on virtual objects. Users performed significantly better on selection tasks when using the HTC Vive wands, and all hand-tracking input methods showed an improvement over traditional mouse/keyboard and gamepad methods. All current VR-focused controllers are constructed similarly to a traditional gamepad, but split in half, with each half held in either the left or right hand. Like a

traditional gamepad, they include buttons, triggers, joysticks, and touchpads. However, they also allow for independent hand movement and motion tracking in 3D space.

Figure 9: Clockwise from top: HTC Vive Wand, Samsung HMD Odyssey controller, Oculus touch

The controllers offered with the Oculus Rift, HTC Vive, and Windows MR headsets are similar in appearance and tracking capabilities but differ in button configuration, as shown in Figure 9. All three controllers are tracked in 3D in a similar manner to the headsets. While they all include primary and secondary trigger buttons, the Vive controller positions its secondary trigger button in the user's palm, requiring a "gripping" motion for activation. The face of the Vive controller includes a large touchpad that functions as a button, while the Rift controller instead has a joystick and two buttons. Windows MR controllers include both a touchpad and a joystick on each controller. The Vive wands are identical and can be held in either hand, where the others are asymmetric and can only be held in one hand.

Testing the HMD volume renderer in multiple headsets presents an opportunity to develop a control scheme that retains the same functionality despite controller differences. The differences between HMD controllers raise issues when designing controls meant to be used on multiple devices. One proposed video game design heuristic suggests improving user experience by ensuring intuitive controller mappings [49], meaning that similar hand movements should be used for the same volume rendering controls regardless of how controller inputs appear through software. Rather than assigning volume rendering controls to inputs as identified in OpenVR, they should be translated to each controller depending on the physical position of each input in the user's hand.

Interacting with Medical Volumes in VR

While viewing a medical volume, users may desire control of functions similar to those used when viewing 2D scans, such as view manipulation, tissue density adjustment, selection of color transfer functions, and clipping. Translation of the volume relative to the 3D world is often not necessary in existing non-VR applications because any viewing angle can be assumed by moving the camera. Keeping the volume stationary and located at the origin of the 3D world also simplifies finding intersections between clipping planes and determining whether the camera is located inside the volume. If the world position of a 3D point is known, it is known to be inside the volume if the vector describing the point's position are all greater than zero and smaller than the volume's dimensions. With the use of an HMD, however, an egocentric point of view is assumed. Only the user's head movement can be used to position the camera. Attaining some viewing angles would require unnatural movements if the volume remains locked in place, but enabling translation and rotation of the volume allows the user to achieve any view with minimal

head movement. Since the volume is no longer locked in place at the world origin, the extra step of transforming a point of interest into the volume's transform is needed to find collisions and intersections with other 3D objects.

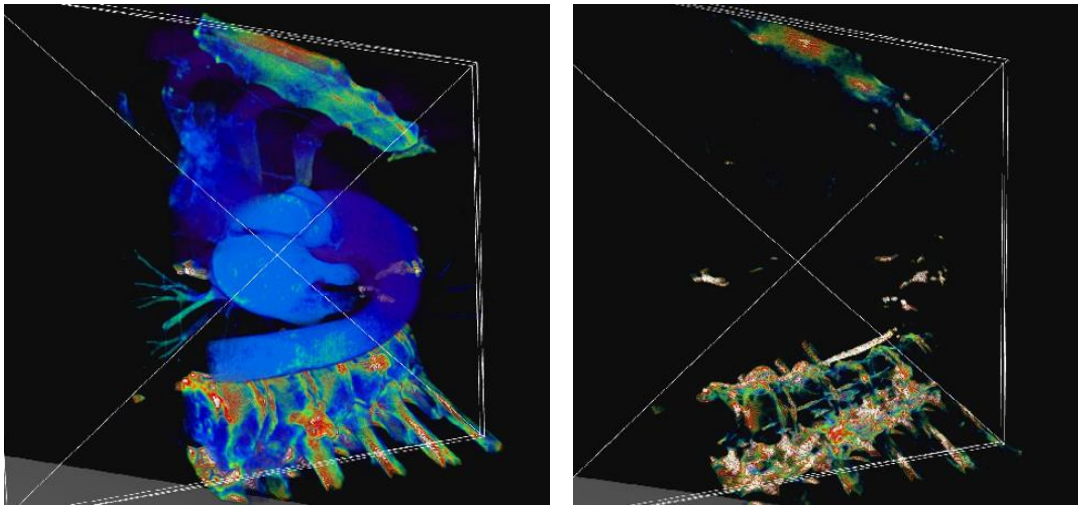

Figure 10: Adjusting clipping planes in VR (left), adjusting density window values (right)

Clipping is a technique used to "slice" through a volume, hiding a region and exposing previously hidden structures in the unclipped region. Clipping planes are one way to describe the boundary between clipped and unclipped regions. Placement of a clipping plane can be aligned to the transverse, coronal, or sagittal axes (respectively, the X, Y, and Z axes of a body scan) to mimic traversal through a set of slices in 2D, or aligned in any orientation to create arbitrary slices.

Another useful function often employed by volume rendering software is tissue density windowing, or adjustment of the minimum and maximum intensity thresholds [15, 16, 4] [11, 33, 34]. Decreasing the maximum intensity of a volume will remove the densest parts of the scan, such as bones and dense tissues, while increasing the minimum intensity filters out softer tissues and blood vessels. Like clipping, windowing can also expose structures in the volume previously

occluded by other tissues of a different density. Figure 10 shows both clipping plane adjustment and windowing functions.

Tissue density values are translated into screen pixel color values through opacity and color transfer functions. Color transfer functions are used to make tissue types more distinct, as the eye is not as sensitive to grayscale values as it is to differences in color [50]. Colorization has also been shown to aid in computer segmentation of tissue types in [7]. Switching between color transfer functions may be desired when attempting to make different tissue types more visually distinct. Examples of different color schemes are pictured in

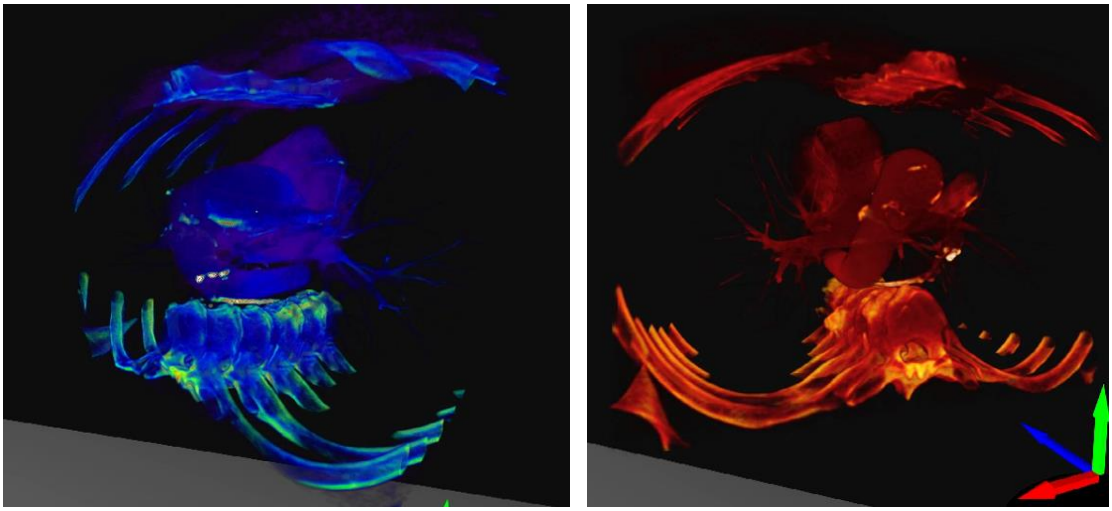

Figure 11.

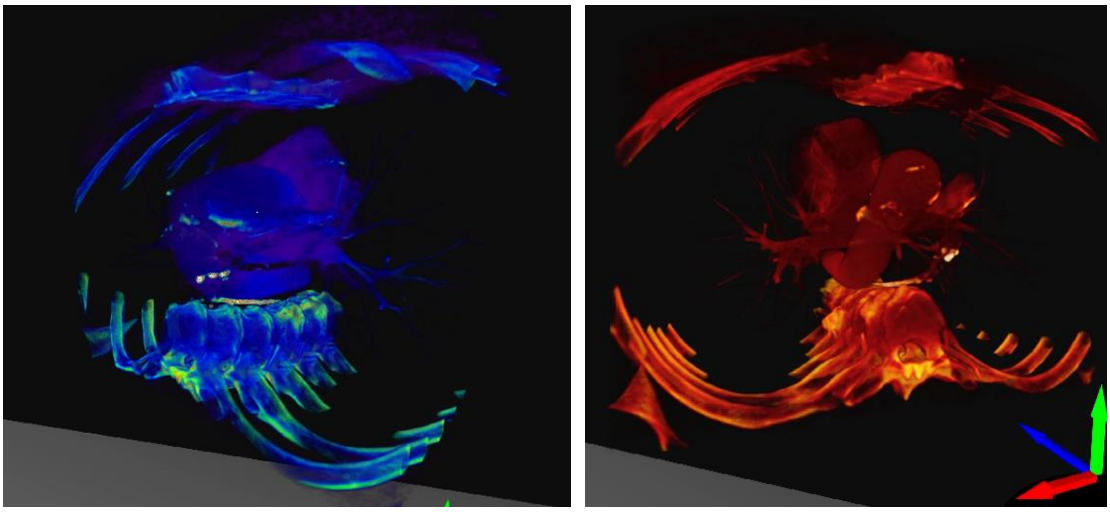

Figure 11: Comparing color transfer functions: NIH (right), "Muscle and Bone" (left)

Support for the HTC Vive was enabled for the 3D medical image viewer MeVisLab in [51]. The VR implementation was not HMD-specific, as any OpenVR headset would work, but controls were not developed for use across multiple devices. Performance testing showed that framerates of 90fps were attainable using desktop hardware, but the medical volumes were displayed using a less intensive surface rendering method rather than volume rendering. ParaView Glance is another 3D viewer that was recently updated with HMD volume rendering compatibility [52]. As with the project described in the previous paper, only the HTC Vive is explicitly supported. Additionally, advanced volume manipulation functions like clipping planes and density windowing are not implemented using VR controllers. There appears to be a gap in medical volume rendering applications that support current HMDs and controllers which the implementation described in this thesis fills.

HMD Volume Rendering Concerns

Moving from rendering to a traditional display to a VR HMD fundamentally changes how the user perceives and interacts with the virtual environment. In VR, the user's entire field

of view is constructed in 3D, and the user's physical movement must result in a view that matches what the user would expect to see. Motion sickness is experienced by many VR users even with current hardware. In order to minimize its effects in this application, movement and volume interaction controls were designed with consideration for phenomena that has been shown to disorient users. In addition, the application's performance target, meaning ideal framerates achieved while running on test hardware, was determined using previous research.

"Vection," or the illusion of self motion, contributes to feelings of motion sickness in people viewing media containing large fields of motion [19, 18]. This effect is magnified by the increased immersion offered by VR devices. Motion sickness can be induced physically or visually. In the former case, the user's actual motion is not reflected in the virtual environment, and in the latter, movement in the virtual environment causes a stationary user to feel as though they should be moving. The general solution to this problem is to increase accuracy of motion tracking and decrease the latency between a user's movement and recalculation of the display. High frequency tracking cameras and location algorithms used in modern HMDs have largely solved the problem of inaccurate tracking contributing to motion sickness.

Even with perfectly accurate tracking, however, users are still subject to visually induced motion sickness under certain conditions within the virtual environment. Any motion applied to the virtual user's position separately from head tracking will suggest to the user that their body is moving. Unless the user's body experiences physical movement that matches with virtual movement, they will feel a conflict between their visual and physical perception of motion. Low framerates are a common culprit of feelings of disorientation in VR in part because longer frame-times increase latency between input and perception. It has been observed that motion sickness

sharply increases as latency increases past 40 ms [14], which is equivalent to framerates dropping below 25 fps. Oculus and Valve have approached this problem with the introduction of positional interpolation functions built in to the Oculus and OpenVR SDKs, respectively called "asynchronous timewarp" and "reprojection". However, if the render framerate is low enough, the user will still notice choppiness and will have difficulty making adjustments to the volume in real-time. One heuristic that has been proposed for use in the design of games and interactive software is to provide consistent responses to user actions [49]. It is impossible to provide consistent feedback to the user if the framerate is changing dramatically. Further contributing to this problem, as the user moves closer to the volume it occupies more screen space, resulting in a larger number of rays cast through the volume. Since the performance impact of volume rendering increases with a greater number of rays to compute, framerates begin to decrease when the GPU is pushed to full load.

These problems highlight the necessity of rendering volumes at a high framerate and the avoidance of unnatural movement schemes. To achieve a high enough framerate for the VR medical volume viewer, the number of ray sampling iterations were capped differently depending on the user's position relative to the volume. When the user's eyes are inside the volume, the maximum number of iterations is halved. This reduces the computational load, resulting in a framerate increase. Because samples are taken less frequently along the length of the ray, resolution in the view direction is decreased, however. Additionally, the "trackball" camera control originally used in VIPRE would force the user to move around the volume along the outside of a sphere. In VR, this movement disorients the user. In this application camera control was limited to user head movement, allowing the user to remain stationary while moving

the volume with their hands, and still encouraging the user to physically move around the volume.

CHAPTER 3: SOFTWARE AND HARDWARE COMPONENTS OF THE MEDICAL VOLUME VIEWER

OpenSceneGraph

OpenSceneGraph (OSG) is a 3D rendering engine developed for platform-agnostic real-time applications [53] OSG uses OpenGL for rendering, providing compatibility with all major operating systems and architectures. Its functions abstract some of the lower level OpenGL tasks, like buffering vertex array objects and creating graphics contexts, to allow the programmer fine control of 3D object properties at a large scale. Furthermore, OSG organizes 3D objects in a graph hierarchy which simplifies the process of applying properties to multiple objects. For example, geometry nodes can be both parents and children of other geometry nodes, and materials applied to a parent can also apply to child objects. The graph structure also simplifies searching for and updating objects during the program's main loop, as the graph can be traversed recursively, each parent calling the update functions of its child nodes. In addition to handling geometry and 3D environment hierarchy, OSG also supports a plugin for DICOM Toolkit (DCMTK) [28] which enables loading of DICOM files.

VIPRE

VIPRE is a library containing low-level volume rendering functionality meant to be incorporated into applications targeting a wide variety of devices. Example volume rendering applications have been demonstrated working on desktop, mobile, and CAVE platforms [11, 38]. The VIPRE library facilitates handling of medical volumes in software through objects that build a 3D texture from loaded DICOM slices and assign it to a geometry cuboid structure. The library incorporates OSG and DCMTK for file loading, texture creation, shader handling, camera manipulation, and other similar functions.

OpenVR

All user interaction is handled through the VR HMD and its controllers. Images rendered using OSG and VIPRE must be communicated to the user, and input must be interpreted by the main application through software. Each HMD is paired with manufacturer-provided driver software that acts as a connection from the operating system to the device, but middleware is needed to link the volume rendering application with the HMD's driver. OpenVR serves in this regard, and is compatible with all major HMDs [54]. SteamVR, Valve's commercial implementation of OpenVR, serves additionally as the main software driver for the HTC Vive, with no additional middleware needed. Like OpenVR, SteamVR is also compatible with the Oculus Rift, Samsung Odyssey, and other major headsets. Communication through OpenVR is achieved through the OpenVR SDK, which presents an interface for retrieving headset and controller states regardless of the device's manufacturer. To create an OSG viewer compatible with OpenVR, the software package osgopenvr [55] was used. This package provides OSG objects like cameras and transforms to represent OpenVR devices, as well as callback functions used for retrieving input and updating the display.

Development Hardware

The VR medical image viewer described in this thesis was developed on a computer running Windows 10 with an Nvidia GTX 1070 GPU, an Intel Core i5-6600 CPU @ 3.3 GHz and 16GB DDR4 RAM. The Oculus Rift, HTC Vive, and Samsung Odyssey HMDs and controllers were used for development and testing. SteamVR was used for its OpenVR driver and runtime application.

CHAPTER 4: INTERACTING WITH GPU RAYTRACED MEDICAL VOLUMES USING COMMODITY VIRTUAL REALITY HMDS

Foreword

The combination of medical imaging and virtual reality is not a new problem. However, a practical application of current consumer-facing HMDs with a control scheme that is both capable of manipulating the volume and consistent across devices has yet to be demonstrated. The following paper describes an application developed to solve this problem and is the focus of this thesis. This paper was submitted to the PLOS ONE journal. It was titled "Interacting with GPU Raytraced Medical Volumes using Commodity Virtual Reality HMDs." The paper has been formatted according to the thesis guidelines, but its contents have not been altered.

Abstract

Radiologists and surgeons rely on medical imaging for tasks including patient diagnosis and surgical planning. Body scans are typically viewed in 2D, but in recent years a rising number of medical professionals are choosing to use 3D rendered scans. Though computationally intensive, increases in computer graphics power have allowed for real time rendering of 3D volumetric datasets built from body scans. As with 2D scans, volume rendered scans are often viewed using traditional 2D computer displays. However, the additional depth information provided by volume rendering creates a desire for a more immersive and interactive experience for which virtual reality (VR) head-mounted displays (HMDs) are a promising choice. When compared to 2D displays, research has shown HMDs to aid users in the classification of bone fractures and tumors. Moreover, motion tracked handheld controllers, included with most HMDs, enable functions desired by physicians including translation, rotation, clipping plane placement, and density windowing. Though several VR implementations of medical imaging

software have been tested with varying degrees of user interaction via hand controllers, none have presented a control scheme that remains consistent across multiple HMDs. The increasing variety of HMDs available to consumers has widened software support for VR in medical imaging, but inconsistencies between controllers of different HMDs have made developing a cross-platform control scheme cumbersome. This paper presents an adaptation of an existing volume rendering engine to current HMDs along with a universal control scheme, resulting in a method of viewing and manipulating volume rendered medical images in VR across several different HMDs.

Introduction

Surgeons and radiologists are trained to examine 2D body scans on a screen, stepping through the body slice by slice from fixed axial viewing angles. The viewer can quickly traverse through a set of 2D scan data, giving 2D scans a high degree of interactivity. The amount of information seen at one time, however, is limited to two dimensions. Through volume rendering, multiple slices can be arranged into a 3D structure and rendered from any point of view, increasing the amount of detail seen at one time. With the ability to render body scans in 3D, a more capable method of viewing and interacting with medical images would be of value to medical professionals and patients. Figure 1 shows the difference between a 2D body scan render and a 3D rendered scan created from similar data.

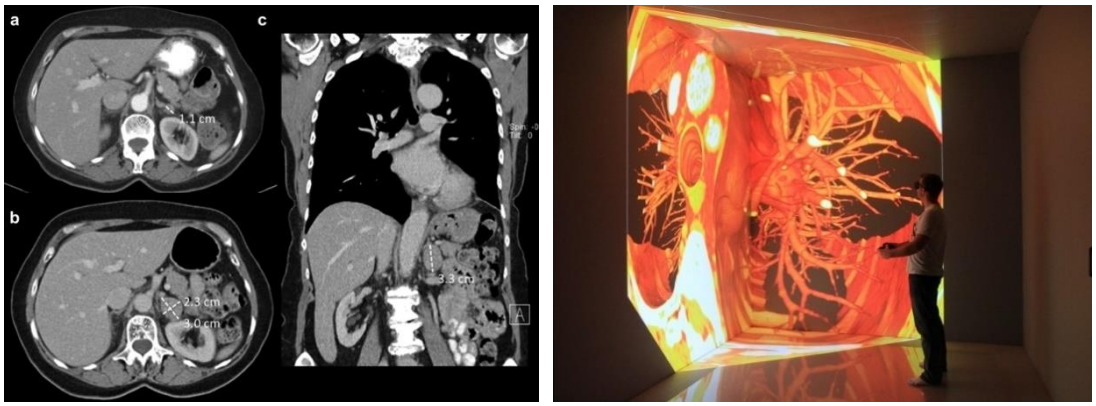

Figure 1: Comparison between 2D rendering of CT scan (left) to volume rendered 3D CT scan (right)

Virtual Reality (VR) Head Mounted Displays (HMDs) introduce stereoscopic viewing and head/hand tracking and have been shown to further aid users in identifying tumors and fractures [1]. Due to recent advances in display and motion tracking technology, Current HMDs such as the Oculus Rift [2], HTC Vive [3], and Windows Mixed Reality HMDs, such as the Samsung Odyssey [4], offer high quality VR viewing at costs much lower than previous headsets. HMD-specific SDKs are of little use for software development if compatibility across multiple headsets is desired, but common drivers and cross-platform SDKs have been developed [5] [6]. Still, as shown in Figure 2 and Table 1, inconsistencies exist between the controllers designed for these HMDs. Because of these factors, incorporating modern HMDs with medical imaging presents two difficulties: 1) rendering volumetric medical data in VR, and 2) creating a VR application that both renders and handles input across multiple headsets and controllers.

Figure 2: From left to right, Oculus touch, HTC Vive wands, and Windows MR motion controllers

Table 1: Available inputs on current HMD controllers

Input Type	*Oculus Rift*	*HTC Vive*	*Samsung Odyssey*
Touchpad	No	Yes	Yes
Joystick	Yes	No	Yes
Triggers (per controller)	Primary and secondary	Primary and grip button	Primary and Secondary
Auxiliary Buttons	Yes	Yes	No
Face Buttons	Yes	No	No

VR demands highly responsive rendering and interactivity, as low framerates and motion tracking latency increase the viewer's chance of experiencing motion sickness [7]. For this reason, HMD manufacturers prioritize high refresh rates and high frequency tracking systems with low latencies. Current HMDs display images at up to 90 frames per second (fps), and VR developers often aim for their software to reach that framerate at all times. However, volume rendering presents challenges when high framerates are needed due to the increase in data size and computations compared to 2D rendering.

Whether viewing is done using a traditional display or a VR HMD, rendering volumes involves projecting the desired view of an object containing 3D data onto a 2D image. To accomplish this, several techniques have been developed. Image splatting projects each point of

the volumetric data set onto the image plane, using a 3D kernel to represent each point's contribution to the image. Splatting can produce high framerates but produces artifacts when kernels overlap [8, 9]. Iso-surface rendering is another technique where the surface of a medical volume is represented by geometric primitives [10]. This gives the volume a smooth and well defined look, but differences in tissue density and internal structures cannot be revealed through transparency as only the surface is rendered. Texture slicing is a technique where rasterized polygonal slices of the volume are created in the form of 2D textured quadrilaterals [11, 12]. These quadrilaterals can be stacked in an alignment that is either perpendicular to the camera, or aligned with the axis closest to the view direction. The use of axis-aligned slices allows for a wide range of camera movement before the slices must be recomputed, but it produces artifacts as the angle between the camera view and the slices increase due to texture filtering. Camera-aligned planes can be rendered with less artifacts but at the expense of more frequent texture computations if the camera is continuously moving. These techniques render volumes using the rasterization capabilities of GPUs. However, as CPU and GPU power have increased, other more accurate techniques such as raytracing were developed.

Raytracing produces images by simulating light rays traveling through a volume. It involves sampling parts of the volume traversed by rays shot from screen pixels in the view direction, and is more computationally intensive than rasterization-based methods like texture slicing. Previously, raytracing has been performed using the CPU. To speed up CPU rendering, methods like shear warp were proposed [11] [13]. Shear warp reduces unnecessary interpolation between values in the medical volume on one axis by aligning the view rays along one of the axes and setting sampling intervals equal to the thickness of a slice. This prevents sampling

points from falling in between volume values. Also, the axis-aligned sampling procedure allows for the use of optimized run-length encoded (RLE) datasets in place of a 3D texture, speeding up the process of sampling along each ray. Forcing the alignment of rays along an axis requires an extra "shearing" adjustment to each slice along with a "warp" performed on the final image to produce the correct camera view.

The appearance of multithreaded CPUs has helped increase performance for these rendering techniques, but a more dramatic increase in performance was necessary to achieve framerates suitable for VR. Fortunately, the programmable shader pipeline implemented in OpenGL [14] and supported by modern GPUs renders pixels in parallel, lending itself well to raytracing algorithms [15, 16]. Using shaders, the ray sampling procedure can be divided among the hundreds or thousands of cores present in modern GPUs. As a result, recent GPU volume renderers have allowed for real-time rendering in full detail.

Interaction is another important component for VR medical imaging software. For an application that supports multiple HMDs, a control scheme that is consistent among different devices is desired. This means that, for a control scheme using hand controllers, similar hand movements and finger presses should be used for the same tasks regardless of the HMD. However, while current HMDs include handheld motion-tracked controllers, button layouts between them are different. Common drivers such as Open Source Virtual Reality (OSVR) and OpenVR have been developed with compatibility for multiple current HMDs, but the input values reported for each control (i.e. button, trigger, joystick, or touchpad) differ between headsets. Some controllers do not include buttons that exist on others, and there is a question of whether button configurations on future HMDs will follow existing conventions. Table 1 shows

differences between three representative HMD controllers. To account for these differences, a mapping of buttons, triggers, and joysticks to controls used for volume placement, clipping plane adjustment, and tissue thresholding was created for each controller.

Background

The VR medical image viewer described in this paper incorporates previously established medical imaging, volume rendering, and VR imaging and interaction concepts. This review will examine previous works that informed the development process.

Volume Rendering of Medical Images

Both 2D and 3D viewing of medical images begins with loading sets of data. For storage and transfer of multiple body scan types, a common file format was needed. Thus, the Digital Imaging and Communications in Medicine (DICOM) format was established, digitally storing medical images along with patient information [17]. The image data stored in DICOM files can opened by many photo viewers including Adobe PhotoShop and GIMP, but specialized software is needed for advanced functions like contrast adjustment, measurement of features, and multi-planar image reconstruction. RadiAnt [18] and Osirix [19] are examples of DICOM viewers that support 2D medical image viewing and reading patient data. For these applications as well as others, DICOM file handling is enabled in software through use of DICOM Toolkit (DCMTK) [20].

Of more interest to the development of this project, however, is the ability to render 3D medical images. In addition to 2D viewing, GPU raytracing has been used to deliver 3D volume rendered DICOM images at interactive framerates in RadiAnt, Osirix, and other applications. Another widely used example of volume rendering software packages is 3D Slicer, which allows

users to make complex three dimensional slices to segment parts of medical volumes [21]. These three applications are built using the Visualization Toolkit (VTK), a rendering engine designed for scientific data visualization [22]. ParaView Glance [23], also built using VTK, includes an HMD viewer that can load DICOM files. It supports the HTC Vive for viewing, but there is no interaction via hand controllers. VIPRE, a volume rendering software kit built using the OpenSceneGraph (OSG) rendering engine, is also capable of delivering interactive framerates using GPU raytracing. VIPRE has been demonstrated rendering medical volumes in both desktop and CAVE configurations at interactive framerates. High framerates were maintained even when rendering stereoscopically, suggesting that VIPRE could also be suitable for HMD applications [15] [13].

Viewing Medical Images in VR

The performance abilities of modern volume rendering software packages combined with modern GPUs has made VR viewing and interaction with volume rendered medical images possible, and some practical applications of this technology have reinforced the idea. The use of a 3D environment for surgical training and viewing medical images was explored by Reinschluessel et al. in an interactive virtual operating room [24]. An HMD was used for viewing 2D body scans with a Leap Motion sensor used to detect navigational hand gestures. Hand and foot movements were mapped to controls surgeons normally operated by verbally instructing an assistant. No significant differences in usability between the two methods were measured. Venson et al. measured a user's ability to identify bone fractures when viewing 3D volume renders in an HMD [1]. Real-time control of tissue density windowing was enabled with the use of a traditional gamepad. Testing showed that external fractures were correctly identified

at high rates, but internal fractures were identified less often. As explained, this result was expected with no way to segment or clip the volume to expose internal structures. Similarly, several medical imaging applications of VR were described by Douglas in [25], where 3D VR viewing was shown to be more effective than 2D viewing in a study tracking user classification of breast cancer microcalcifications. It was also noted that an HMD with a joystick control method enhanced the user's abilities and improved accuracy in movement. Furthermore, the idea that virtual reality could also enhance communication between radiologists and surgeons was presented, stating that the 3D location of features could be better communicated in a VR environment. An integration of the HTC Vive with existing medical imaging software was tested for performance in [26] [27]. Framerates of 90 fps were achieved in the HMD, which is sufficient for VR [28]. Basic interaction with the host program using the Vive controllers and viewing with multiple HMDs was achieved, but the user had no direct interaction with the volume (translation, rotation, or clipping). In another study, the stereopsis cues given when viewing medical volumes in an HMD was tested for its effect on users performing position judgement tasks [29]. It was shown that stereoscopic viewing led to higher performance.

VR Hand Controls and Interaction

Previous uses of hand tracking for 3D controls demonstrates its potential advantages over traditional keyboards or gamepads. Gallo et al. proposed a touchless medical image viewer with input controlled only by hand gestures [30].

In [31], the HTC Vive wands were compared to hand-only tracking while performing motor tasks on virtual objects close to the user's face. It was found that subjects using Vive wands were significantly less error-prone while performing selection-based tasks. For other tasks, like

grabbing, the Vive wands were similar in effectiveness to the hands. This research suggests that the use of VR-focused control methods that track the hands may benefit medical imaging applications.

Through review of prior works, examples of projects implementing multiple facets of medical imaging, VR, and interaction concepts were found. It was shown that framerates high enough for VR could be achieved when rendering volumes using raytracing, that HMDs can enhance a user's spatial understanding compared to a traditional display, and that tracked hand controllers improve interaction compared to a keyboard or gamepad. However, no single example or project fulfilled all areas of interest. Namely, there is a gap in implementations of VR medical image viewers that accept hand controller input for common volume rendering tasks, and also support multiple types of HMD controllers. The goal of the development process described in this paper was to build off existing knowledge to create methods for real-time viewing and interacting of 3D medical volumes in a VR HMD. These methods should also allow viewers using a variety of HMDs to interact with volumes in a consistent manner. To demonstrate the methods, an interactive VR application was developed. Approaching this task first involved selecting software components that could be used to create a VR medical volume viewer and establish communication with VR hardware.

Methodology

Volume Rendering Components

Established volume rendering software like 3D Slicer, RadiAnt, and Osirix render volumes at high framerates. However, they are not all optimal for the purpose of this project. While Osirix uses the platform-independent VTK rendering engine, the software is only

compatible with Mac OS, an operating system that is unsupported by current HMDs. 3D Slicer, which also uses VTK, is platform independent. 3D Slicer features elaborate tissue segmentation and medical image viewing, but it is aimed more at big data visualization than interactive experiences, and VTK's scene graph implementation is primitive compared to alternatives. VIPRE is an API that provides a shader-based raytracing volume renderer and objects for building and manipulating medical volumes. It was developed to fulfill a need for a volume renderer that could be deployed across a large range of devices, from large clusters to mobile tablets [15] [13]. Where 3D Slicer uses VTK, VIPRE uses the OSG rendering engine. OSG's geometry creation, texture loading, and matrix transform functions were used to create the 3D environment. OSG uses OpenGL for rendering, meaning that while complex graphics operations can be abstracted and performed with simple commands, custom shaders written using GLSL can be loaded and low-level OpenGL commands are available if desired [32]. OSG also supports the use of a plugin for DCMTK, a software library through which DICOM medical image files are loaded. VIPRE was chosen for this VR renderer due to its previous use in stereoscopic volume renderers and its inclusion of a scene graph-based rendering engine suited for interactive applications.

Rendering the VR scene according to the movement of the user and interpreting controller input also requires a way of communicating with the HMD through software. For this, SDKs created specifically for the Oculus Rift and Windows Mixed Reality HMDs were options. The HTC Vive SDK is not restricted to any HMD, however, it is an implementation of OpenVR. OpenVR, compatible with all current HMDs, provides a way to read hardware information and tracking data and send rendered images back to the HMD. It sits between the HMD's driver and

the software currently rendering to the HMD. The OpenVR SDK was desired for its compatibility with all major VR devices and its documented uses in similar projects [27]. Osgopenvr [33], a collection of HMD setup code, provides an interface between OpenVR and OSG.

Development Hardware

The computer used for development was equipped with an Intel Core-i5 6600 processor, 16GB RAM, and an Nvidia GeForce GTX 1070. The HTC Vive, Oculus Rift, and Samsung Odyssey HMDs were used for development and testing. The HTC Vive and Oculus Rift both have total resolutions of 2160 x 1200, while the Samsung Odyssey's total resolution is 2880 x 1600. Per-eye resolution is 1080 x 1200 and 1440 x 1600, respectively. The VR volume renderer described in this paper is compatible with Microsoft Windows 10.

Base Volume Rendering Functionality

In the starting position, the volume is placed in front of the user. To allow the user to move around the volume naturally, controller input is used to move the volume rather than the user's virtual body. User movement is achieved by physically walking, crouching, and leaning around the tracked space. Since the user experiences no movement apart from that of their own body, motion sickness is minimized.

Initially, a user specified DICOM image set is loaded and the slices are assembled into a 3D texture. The 3D volume consists of a cuboid-shaped geometry enclosing the 3D texture. If a problem is detected with the DICOM file during loading or if no file is specified, an error message is sent as output and the program terminates. After successfully loading the image, the OpenVR system is initialized. This process connects the currently running program to the

OpenVR driver, receives information about the headset, and creates objects in software representing the state of the user's HMD and controllers. A buffer texture is created as a render target for the HMD, and software cameras are created to represent the user's eyes.

With an image file loaded and information gathered from the headset, the VR scene is built. A simple virtual environment was created in which the volume and the user were placed, along with a stationary floor that adds a locational reference for the user's movement. A transform was added to the volume so it can be manipulated separately from the cameras. While the user's head and hand positions are reported though OpenVR relative to the world, the volume's inverse transform must be applied to world transforms to find positions relative to the volume. A set of arrows indicating the XYZ axes is placed above the right hand, following the volume's rotation to indicate heading. Next to the arrows, a 3D text object is placed containing the clipping plane position. Figure 3 shows an illustration of the virtual environment and a screenshot from the running program.

During every iteration of the main loop, controller position and input states are updated. Input used for moving the volume, controlling the clipping planes, and moving the virtual controllers is applied to the scene and sent to the shader through uniform variables. Stereoscopic 3D requires two views to be rendered each frame, one per eye. This means that the rendering procedure alternates between the eyes, and that shader variables must be updated twice per frame. During each eye camera's pre-draw stage, a boolean value representing whether the eye is currently inside the volume is sent to the shader. The vertex shader receives variables necessary for applying user input to the volume, including the location of the current eye relative to the

volume and the location and orientation of the currently selected clipping plane. This information is then sent to the fragment shader, where the volume is rendered.

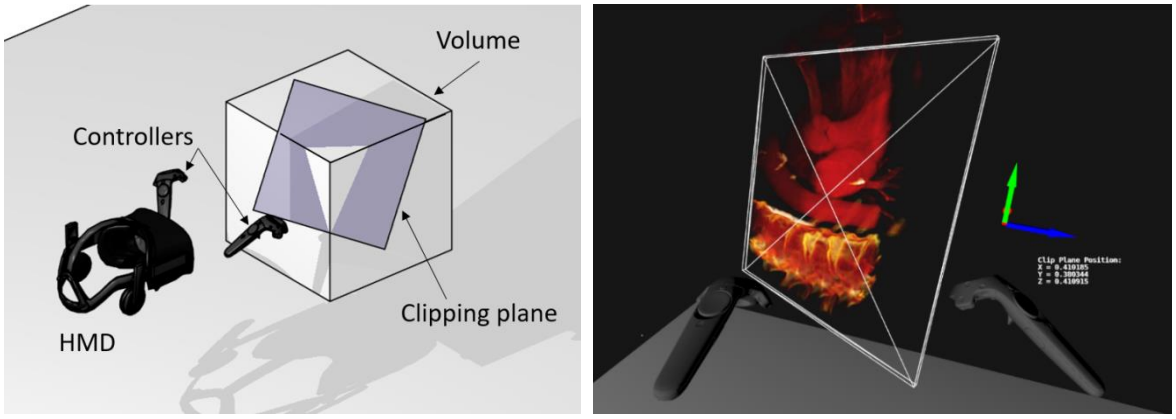

Figure 3: Diagram of the virtual environment (left) and runtime screenshot (right).

Adding OpenVR viewer to VIPRE

The osgopenvr software package provides an interface from OSG to the OpenVR SDK. VIPRE's window creation and camera setup code was replaced with OpenVR-compatible functions enabling stereoscopic rendering. Previously, VIPRE instantiated a single camera that rendered an image to a window. With Osgopenvr, a master camera is created and placed at the user's head, with two slave Render-to-Texture (RTT) cameras located at the position of each eye. Each RTT camera renders to its respective half of the HMD buffer texture. Since both RTT cameras are slaves to the master camera, transformations applied to the master camera also apply appropriately to each eye. Thus, virtual body movement can be achieved by translating only the master camera. Additionally, OpenVR supports movement through "teleporting." This function moves the user's virtual body by manipulating the world's origin, meaning that user movement can be achieved in software built with OpenVR without additional code.

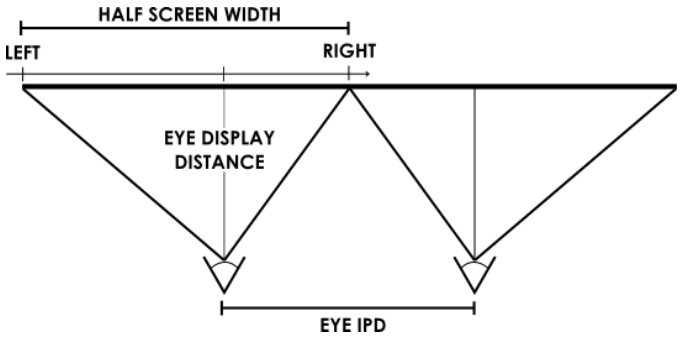

Figure 4: Illustration of eye position relative to image plane

Raycasting modifications

Stereoscopic rendering to an OpenVR headset requires the composition of a single image containing each eye's view. A buffer texture is created to store each frame, comprised of the left and right eye camera views arranged in Side-by-Side (SBS) format. As shown in Figure 4, each frame acts as an image plane while the eye is the camera origin point. To ensure compatibility with HMDs, modifications to VIPRE's raycasting procedure, specifically the portion determining ray direction and origin, were necessary. The ray-based intersection computation procedure previously used in VIPRE places the start point of each ray by calculating the position of the current fragment on the image plane as shown in equation 1.

$$vec3\ startpt\ =\ vec3(gl\_fragCoord.x, gl\_fragCoord.y, 0)\ *\ view\_matrix \qquad (1)$$

This method assumes that the camera projection matrices are symmetric, an assumption that is correct for a majority of 2D rendering applications. However, if the eye position is off-center relative to the image plane on one or both axes, the view frustum becomes oblique, meaning the angles of the bounds on the affected axes are unequal. Current HMDs require the use of off-axis view frusta mainly to accommodate differences in Inter-Pupillary Distance (IPD)

between users. As user IPD is adjusted, the origin point of each eye's camera changes, but the HMD screens typically remain in place. For any IPD adjustment that places the eyes off-center, the left and right bounds of the view frusta will have different angles. Unique projection matrices are designed according to the construction of each headset, and are retrieved in software by querying the OpenVR HMD device object. To find the ray direction during raycasting, when rendering for an HMD, the eye camera's position must be used as the start point of the ray for each fragment rather than the fragment's position on the image plane. Additionally, in the VR application, transformations are applied to both the camera and the volume. Thus, the view matrix is replaced with the inverse of the model-view matrix to determine camera position relative to the volume as shown in equation 2.

$$vec3\ start\_point\ =\ vec3(0,0,0,1)\ *\ modelViewMatrixInverse \qquad (2)$$

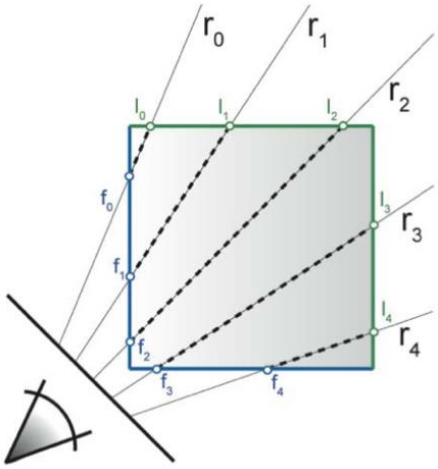

Figure 5: Ray generation when the eye is located outside of the volume

Because the eye position remains constant for all fragments in each frame, the start point value is only calculated once per eye, during the vertex shader phase. In the fragment shader, the

ray's direction is determined by creating a line that passes through the eye camera's origin and the position on the outside of the volume to which that fragment points. The end point of the ray is given automatically by the vertex shader, and does not change with an oblique frustum.

Finding eye position relative to volume

The start and end points of each ray must be calculated differently depending on the position of the eye relative to the volume. When the eye is outside the volume, rays are cast starting from the cuboid face pointing into the volume, terminating once they reach the backface behind the volume. When the eye is inside of the volume, the start point of the ray is the camera position and the ray terminates at the first collision with the backface of the cuboid. Example rays are shown in Figure 5. The ray r_0 is cast through the volume, and start point f_0 and end point i_0 are calculated.

Detecting when the head is inside the volume is performed by finding the eye position relative to the volume's transform. This process is done twice, using each eye's respective camera. Multiplying the inverse of the volume's transformation matrix by the eye camera's inverse view matrix gives the position of the eye in volume space. The volume space eye position is then compared to the dimensions of the volume. If any component of the eye position is outside of the dimensions of the volume, it is known that the eye is outside of the volume. The eye status is sent as a boolean value to the shader before each camera's draw, ensuring that it is completed before the draw stage of each eye.

GPU clipping planes

Clipping is used to "slice through" the volume, revealing internal bone and tissue structures normally occluded by surrounding material. A plane is used to divide the clipped and

non-clipped regions, specified by an origin point and a normal vector. VIPRE's rendering stage was modified to test points sampled from the volume for their positions relative to the clipping plane. In the sampling portion, rays cast from the eye pointing to the volume are used to traverse through the volume. During each iteration, the point increments along the line and samples the intensity of the 3D volume texture at the current point. The sampled value is converted to a density and then color value through a density lookup table and a color transfer function. The computed color value for each step along the ray is accumulated into a final fragment color. During each step, however, the color computation procedure can be skipped if the current point is located on the clipped side of the clipping plane. The side of the clipping plane on which each point is located is found by taking the dot product of the plane normal with a vector pointing from the plane origin to the point in question. If the current point is found to be located in the clipped region, it is skipped, and its value contributes nothing to the final fragment color.

There are two clipping plane types employed in the VR volume viewer, a free moving clipping plane that follows the hand's position exactly, and a clipping plane that snaps on the X and Y rotation axes in 15 degree increments. Controller assignment for adjusting both planes is shown in Figure 6. The user presses one of the triggers on the right controller; the top trigger to select the free clipping plane, and the bottom trigger to select the snapped clipping plane. In the initial placement mode, the clipping plane follows the position of the user's hand, but in the secondary placement mode, the clipping plane remains in place relative to the hand when positioning to aid the user in making smaller adjustments. After placing a clipping plane, the user may also wish to move the position of the plane in fine increments without involuntary hand movements affecting the adjustment. To do this, the user places a clipping plane normally, and

translates it on the horizontal plane it by scrolling with the thumb on the right-hand touchpad. For controllers without a touchpad, such as the Oculus Touch, pressing a modifier button on the right controller toggles the right joystick between volume rotation and clipping plane movement. The 3D text element above the right hand is then updated to show the current position of the clipping plane in volume coordinates.

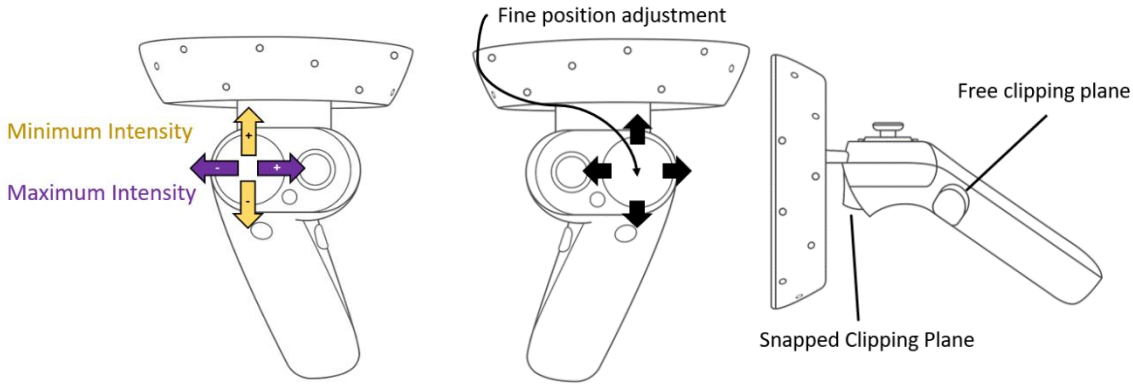

Figure 6: Illustration of density windowing and clipping plane adjustment

Tissue Density Windowing

Users viewing medical volumes may wish to adjust the rendering window, which changes the upper and lower density values that are displayed. Raising the lower threshold will cause low density tissues to disappear, and lowering the upper threshold removes high density tissues and bones. This can reveal structures in the volume previously obscured by materials of different densities. By default, the upper and lower thresholds are set to the minimum and maximum intensities recorded in the DICOM set, so that the entire volume is rendered. Also shown in Figure 6 is the controller assignment for this windowing. To raise and lower the upper threshold, the user touches the left touchpad on the horizontal axis, with the left side of the touchpad lowering the value, and the right side increasing the value. The same action is used to

change the lower threshold, but the vertical axis is used for adjustment. Touching the lower half of the touchpad decreases the lower threshold, and the upper half increases it.

Designing a cross-platform control scheme

As seen in Figure 2 and Table 1, the controllers currently available for the Oculus Rift, HTC Vive, and Windows MR headsets are similar in appearance and tracking capabilities but differ in button, joystick, and touchpad configuration. Testing the HMD volume renderer in multiple headsets presented an opportunity to develop a control scheme that retained the same functionality despite controller differences.

All three controllers are tracked in 3D, sending position and orientation to software through OpenVR. While all controllers include primary and secondary trigger buttons, the Vive controller positions its secondary trigger button in the user's palm, requiring a "gripping" motion for activation. The face of the Vive controller includes a large touchpad that functions as a button, but the Rift controller instead has a joystick and two buttons. The Samsung Odyssey controller include both a touchpad and a joystick on each controller.

To increase the likelihood of compatibility with current and future OpenVR-compliant controllers, the most commonly used functions for volume viewing were assigned to the features most consistent among current controllers. Translation is achieved either by moving the left joystick, or by holding down the primary trigger on the left controller and moving the left controller, similar to a "rope pulling" gesture. Rotation is assigned to the horizontal right joystick axis. This control scheme follows conventional "twin-stick" gamepad configurations where translation is assigned to the left hand and rotation assigned to the right hand.

HMD controllers report values to OpenVR by assigning joysticks, touchpads, and triggers to a list of numbered input axes. Some differences in how each controller reports inputs to OpenVR are organized in Table 2. For example, OpenVR controller axis 0 corresponds to both the Windows MR touchpad and the Oculus Rift joystick, and joystick control using a Windows MR controller instead requires reading OpenVR axis 2. During the program's initialization, the name, manufacturer, and other device-specific information is read from the HMD. Hardcoded input axis values for each of the current HMDs and controllers were stored in the program. To ensure that controls are interpreted properly across all headsets, the input axis numbers for each controller are determined by matching the information read from the HMD to the hardcoded values. The mapping from controller input to OpenVR is shown in Figure 7 and Table 2. This method simplifies interpreting input from multiple controllers. To add a new OpenVR-compatible controller, only the joystick, touchpad, and trigger axes must be specified.

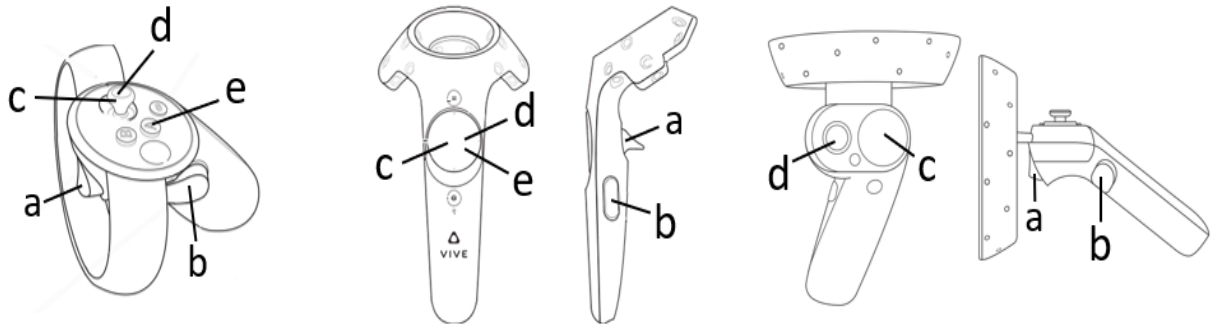

Figure 7: Illustration of controller mapping

Table 2: Input types with their corresponding OpenVR designations

Input Type	*Oculus Rift*	*HTC Vive*	*Odyssey*
a. Primary trigger (free clip plane, volume translation)	Axis 1	Axis 2	Axis 1
b. Secondary trigger (snapped clip plane,)	Grip trigger	Grip Button	Grip Button
c. Touchpad (clip plane adjustment, density windowing)	N/A	Axis 0	Axis 0
d. Joystick (translation, rotation)	Axis 0	N/A	Axis 2
e. Touchpad/Joystick toggle modifier	"A" button	Right touchpad	N/A

Evaluation of feasibility and performance

The feasibility of the rendering and interaction methods described in this paper was assessed by implementation and testing in a VR medical volume viewing application. Testing the application involved measuring performance, verifying that the application worked properly, and making sure the controls were consistent on all three HMDs.

While using the application with different HMDs and controllers several areas of improvement were found. The Oculus Rift and HTC Vive place their world origin at different heights than the Samsung Odyssey, resulting in an unpredictable starting position. This was fixed with a one-time adjustment, so a keyboard button was assigned to reset the viewing position. Also, a control modification was made to facilitate positioning of the free clipping plane. Initially, the clipping plane followed the absolute position and orientation of the right hand. After the user finished placing the clipping plane and moved their hand, an attempt to further adjust the position of the clipping plane immediately forced the plane to the current position and orientation of the hand. This was jarring when a user wished to make small movements, so a change was made to how clipping plane input was handled.

Initial placement of the clipping plane was locked to the user's hand, but on subsequent adjustments the clipping plane placement was handled in a different way. As the clipping plane trigger was depressed, the position and rotation of the clipping plane relative to the hand was recorded and maintained through the duration of the placement gesture, until the user released the trigger. This means that the clipping plane moves as a child of the hand rather than assuming the hand's exact position and rotation, similar to the grabbing motion used for translating the volume. This scheme allows a user to make repeated gradual hand movements to adjust the clipping plane's location and orientation through the volume.

Table 3: Performance of the VR medical volume viewer application

Distance	**Far from volume (2m-10m)**	**Near volume (0-1m m)**	**Inside Volume (original method)**	**Inside Volume (with raycasting modification)**
Observed FPS	55-90fps (max)	45-60 fps	15-35fps	25-35fps

In pursuit of interactive framerates, performance requirements were set at a minimum of 25 fps. This was because it has been observed that motion sickness in users increases significantly as frame times reach values higher than 40 ms [28]. DICOM image sets of different sizes were loaded as volumes in the virtual environment. Framerates measured while viewing the volumes using the Samsung Odyssey were recorded and are shown in Table 3.

It was observed that framerates dropped below the minimum target when the user's head was located inside the volume. In this situation, the volume occupies the user's full field of view, and a ray must be shot through the volume for every pixel on the display. To increase framerates when rendering from inside the volume, a modification to the ray sampling stage of the

raytracing fragment shader was made. The number of rays cast through the volume cannot be reduced without effectively decreasing screen resolution. Instead, sampling calculations were reduced by decreasing the maximum number of samples taken along each ray and increasing the distance between samples. When the user is located outside the volume, sampling iterations are capped at 1024, but this was decreased to 512 while inside the volume. Effectively, the resolution of the volume render in the view axis is decreased by up to one half. However, the perceived difference in quality between maximum iterations is minor in the tested datasets, likely because many of the rays cast through these volumes pass through fewer than 512 data points. This modification increased minimum framerates experienced while rendering from inside the volume, and its effects on performance were then observed.

When the volume was fully in view (i.e. the eye position was far away), framerates between 55 and 90 fps were achieved for all data sets. When approaching the volume framerates began to decrease sooner for the larger Cardiac-CT and MANIX scans. As the user moves closer, minimum framerates drop to 30 fps. Entering the volume caused framerates to drop to their lowest value. Before limiting ray sampling iterations for rendering from within the volume, framerates ranged from 45 fps to below 20 fps, but after the modification the application reached a minimum of 34 fps when the user's head entered inside the volume. Framerates measured while inside the volume are highly dependent on the viewing angle and eye position, as approaching the backface of the volume results in shorter rays and thus fewer sampling iterations. To test the lower limits of framerates while inside the volume, measurements were taken just after the eye entered inside the volume.

Decreasing the number of maximum sample iterations to 512 was effective in increasing minimum framerates while inside the volume. Increasing maximum sample iterations past 1024 did not show a significant effect on the tested datasets due to their size, but continuing to decrease the maximum sample iterations below 512 increased the framerates achieved on all datasets. While limiting the number of maximum sample iterations to low values causes a difference in appearance, lower maximum values could be used to achieve acceptable framerates when rendering larger volumes. To further observe its effects with lower sample iteration values, a larger dataset was tested and compared to a previously tested dataset. The results are shown in Table 4. The Thorax dataset did not achieve minimum framerates of 25 fps or higher until maximum iterations were decreased to 256. At 128 iterations the minimum reached 37 fps, which significantly improved interaction. Appearance of the volume changes, as shown in Figure 8, due to opacity values for each sample being unchanged. Color and opacity could be corrected by recalculating values based on the lowered maximum iterations values. However, this still leaves the problem of an effective decrease in depth resolution due to the increase in space between samples. Further testing could determine the level of accuracy observed when rendering with low maximum iterations, and whether these values would be acceptable for medical volume viewing.

Table 4: Minimum framerates observed under multiple maximum iterations values

				Minimum Framerates (fps)			
				Maximum Iterations			
DATASET	**Slice Resolution**	**No. Slices**	**Slice Thickness (mm)**	**128**	**256**	**512**	**1024**
CARDIAC-CT	512x512	356	1	45	36	36	23
THORAX	512x512	593	1	37	26	22	20

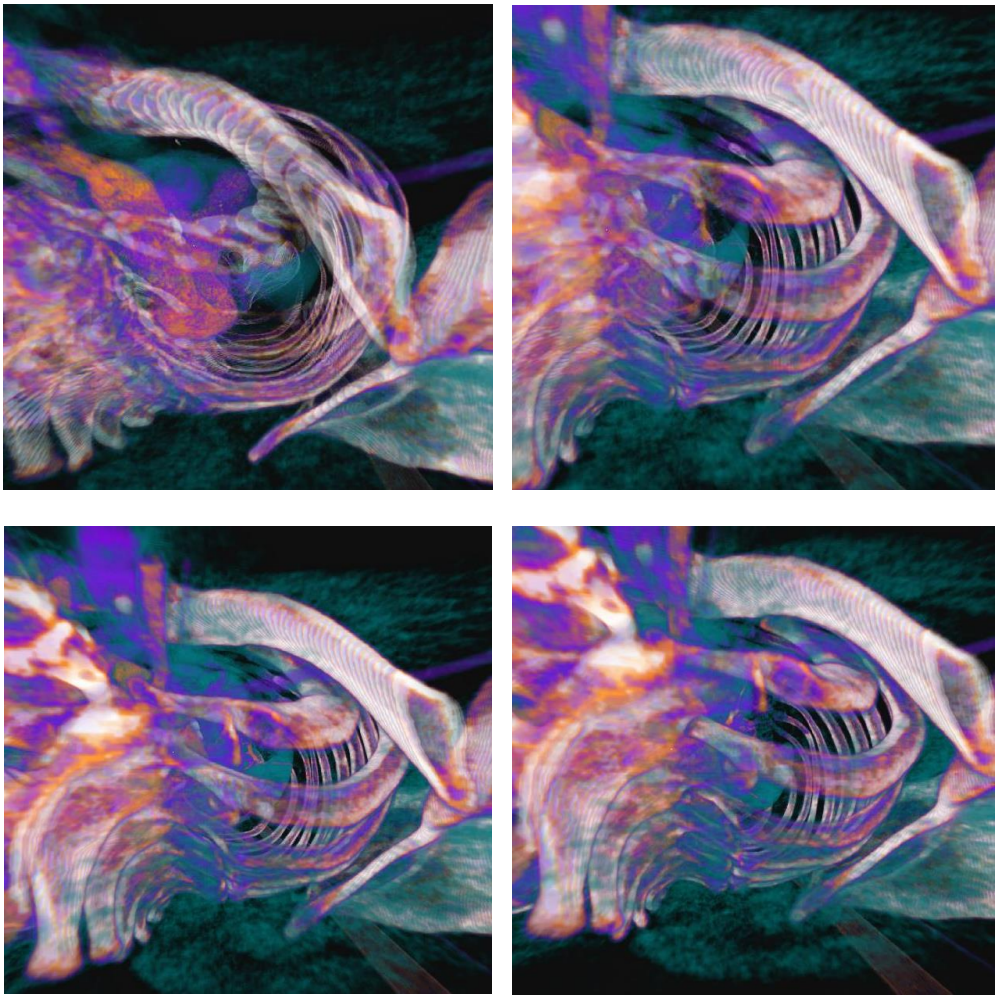

Figure 8: Thorax dataset observed rendered at different maximum iterations values. Clockwise from top left: 128, 256, 512, and 1024 iterations

Framerates were the same between the Rift and Vive, but higher framerates were achieved when observing the volume up close using these headsets when compared to the Odyssey. The similarity in performance between the Rift and Vive was expected, as all three headsets are limited to 90 fps. The difference in performance noted when using the Samsung Odyssey is due to the headset's higher resolution screens, which require more rays to be calculated than when using the Rift or Vive. Up-close rendering performance was observed to be

bound by GPU power, as CPU utilization remains unchanged as framerates drop while the user enters the volume. Running at a minimum of 34 fps, the application is usable, but a framerate of 60 fps or greater would greatly improve interaction with the volume and decrease the user's chance of experiencing motion sickness [19]. Replacing the Nvidia GTX 1070 with newer and faster GPU would show an immediate benefit to performance, but the raytracing shader also contains some inefficiencies that could further improve framerates across all hardware if addressed.

Conclusions and Future Work

For this project, methods were developed for viewing and interacting with volume rendered medical images using multiple VR HMDs. Differences in controller layouts were handled through a control scheme that assigns software inputs based on the construction of each controller. This was done to ensure that similar volume rendering functions are operated by the same fingers regardless of the HMD and controllers the viewer is using. Functions used when viewing medical volumes like clipping and windowing were enabled through use of GPU shaders. Other volume manipulation functions like translation and rotation were implemented through creation of a 3D environment. Motion sickness was minimized by uncoupling camera movement from the volume, instead allowing the user to physically move around the volume. A prototype application was built using the VIPRE volume renderer, and VR viewing was enabled using OpenVR. Interactive framerates between 25 fps and 90 fps were achieved through GPU raytracing. Performance improvements to the rendering method that could be realized in future work were described as well. This research demonstrates that GPU volume raytracing can

produce satisfactory framerates for VR, and that support for volume viewing and interaction among current consumer-facing HMDs and controllers is possible.

In addition to more powerful hardware, aspects of the volume rendering method could be modified to improve performance. Currently, the ray sampling procedure traverses all the way through the volume, even when part of the volume is clipped. As an improvement to this method, the start and end points of the ray could be found according to the position of clipping planes. This would decrease the number of iterations required to compute each pixel proportionately to the amount of the volume that is clipped out, resulting in potentially higher framerates. One way to implement this would be to modify the geometry of the structure enclosing the cuboid after clipping planes are placed. If a clipping plane is placed that divides the cuboid in half, this would modify the geometry to move the boundary of the clipped region to the position and orientation of the clipping plane. Another improvement could involve dynamically adjusting the maximum number of sampling iterations through the volume to limit calculations more when the framerate drops. This could allow for maintaining a desired framerate, possibly 90 fps, while rendering detail increases and decreases according to computational load.

References

[1] J. E. Venson, J. Berni, C. S. Maia, A. M. Da Silva, M. D'Ornelas and A. Maciel, "Medical imaging VR: Can immersive 3D aid in diagnosis?," *22nd ACM Conference on Virtual Reality Software and Technology, VRST 2016,* Vols. 02-04-Nove, pp. 349-350, 2016.

[2] "Oculus Rift," [Online]. Available: https://www.oculus.com/rift/#oui-csl-rift-games=robo-recall.

[3] "VIVE™," [Online]. Available: https://www.vive.com/us/.

[4] "HMD Odyssey," [Online]. Available: https://www.samsung.com/us/computing/hmd/windows-mixed-reality/xe800zaa-hc1us-xe800zaa-hc1us/.

[5] "OSVR Organization · GitHub," [Online]. Available: https://github.com/osvr.

[6] "GitHub - ValveSoftware/openvr: OpenVR SDK," [Online]. Available: https://github.com/ValveSoftware/openvr.

[7] L. J. Hettinger and G. E. Riccio, "Visually Induced Motion Sickness in Virtual Environments," *Presence: Teleoperators and Virtual Environments*, vol. 1, no. 3, pp. 306-310, 1992.

[8] K. Mueller, T. Moller and R. Crawlis, "Splatting without the blur," in *Proceedings Visualization '99 (Cat. No.99CB37067)*.

[9] M. Botsch and L. Kobbelt, "High-quality point-based rendering on modern GPUs," in *11th Pacific Conference onComputer Graphics and Applications, 2003. Proceedings.*.

[10] W. E. Lorensen, H. E. Cline, W. E. Lorensen and H. E. Cline, "Marching cubes: A high resolution 3D surface construction algorithm," in *Proceedings of the 14th annual conference on Computer graphics and interactive techniques - SIGGRAPH '87*, New York, New York, USA, 1987.

[11] Lacroute and P. Gilbert, *Fast volume rendering using a shear-warp factorization of the viewing transformation*, Stanford University, 1996.

[12] C. Rezk-Salama, K. Engel, M. Bauer, G. Greiner and T. Ertl, "Interactive volume on standard PC graphics hardware using multi-textures and multi-stage rasterization," in *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS workshop on Graphics hardware - HWWS '00*, New York, New York, USA, 2000.

[13] C. Noon, J. Holub and E. Winer, "Real-time volume rendering of digital medical images on an iOS device," *IS&T/SPIE Electronic Imaging. International Society for Optics and Photonics*, no. March 2013, 2013.

[14] "OpenGL - The Industry Standard for High Performance Graphics," [Online]. Available: https://www.opengl.org/.

[15] C. Noon, E. Foo and E. Winer, "Interactive GPU volume raycasting in a clustered graphics environment," *Medical Imaging 2012: Image-Guided Procedures, Robotic Interventions, and Modeling*, vol. 8316, no. February 2012, 2012.

[16] M. Hadwiger, P. Ljung, C. R. Salama and T. Ropinski, "Advanced illumination techniques for GPU volume raycasting," in *ACM SIGGRAPH ASIA 2008 courses on - SIGGRAPH Asia '08*, New York, New York, USA, 2008.

[17] W. D. Bidgood and S. C. Horii, "Introduction to the ACR-NEMA DICOM standard.," *Radiographics : a review publication of the Radiological Society of North America, Inc*, vol. 12, no. 2, pp. 345-55, 1 3 1992.

[18] "RadiAnt DICOM Viewer," [Online]. Available: https://www.radiantviewer.com/.

[19] "OsiriX DICOM Viewer | The world famous medical imaging viewer," [Online]. Available: https://www.osirix-viewer.com/.

[20] "dicom.offis.de - DICOM Software made by OFFIS - DCMTK - DICOM Toolkit," [Online]. Available: https://dicom.offis.de/dcmtk.php.en.

[21] "3D Slicer," [Online]. Available: https://www.slicer.org/.

[22] "VTK - The Visualization Toolkit," [Online]. Available: https://www.vtk.org/.

[23] "ParaView Glance," [Online]. Available: https://kitware.github.io/paraview-glance/index.html.

[24] A. V. Reinschluessel, L. Raimondo, L. Reisig, M. Ruedel, D. Thieme, T. Vahl, G. Zachmann, R. Malaka, J. Teuber, M. Herrlich, J. Bissel, M. van Eikeren, J. Ganser, F. Koeller, F. Kollasch and T. Mildner, "Virtual Reality for User-Centered Design and Evaluation of Touch-free Interaction Techniques for Navigating Medical Images in the Operating Room," in *Proceedings of the 2017 CHI Conference Extended Abstracts on Human Factors in Computing Systems - CHI EA '17*, New York, New York, USA, 2017.

[25] D. B. Douglas, J. M. Boone, E. Petricoin, L. Liotta and E. Wilson, "Augmented Reality Imaging System: 3D Viewing of a Breast Cancer.," *Journal of nature and science*, vol. 2, no. 9, pp. 1-6, 2016.

[26] P. Boechat, A. Hann, X. Li, J. Egger, M. Gall, X. Chen and D. Schmalstieg, "HTC Vive MeVisLab integration via OpenVR for medical applications," no. Fig 1, pp. 1-14, 2017.

[27] J. Egger, M. Gall, J. Wallner, P. Boechat, A. Hann, X. Li, X. Chen and D. Schmalstieg, "HTC Vive MeVisLab integration via OpenVR for medical applications," *PLOS ONE*, vol. 12, no. 3, p. e0173972, 21 3 2017.

[28] J. Jerald and Jason, The VR book : human-centered design for virtual reality, Association for Computing Machinery and Morgan & Claypool, 2015, p. 599.

[29] M. Martinez Escobar, B. Junke, J. Holub, K. Hisley, D. Eliot and E. Winer, "Evaluation of monoscopic and stereoscopic displays for visual–spatial tasks in medical contexts," *Computers in Biology and Medicine*, vol. 61, pp. 138-143, 1 6 2015.

[30] L. Gallo, A. P. Placitelli and M. Ciampi, "Controller-free exploration of medical image data: Experiencing the Kinect," in *Proceedings - IEEE Symposium on Computer-Based Medical Systems*, 2011.

[31] L. Figueiredo, E. Rodrigues, J. Teixeira and V. Techrieb, "A comparative evaluation of direct hand and wand interactions on consumer devices," *Computers & Graphics*, vol. 77, pp. 108-121, 1 12 2018.

[32] "OpenSceneGraph," [Online]. Available: http://www.openscenegraph.org/.

[33] Chris Denham, "GitHub - ChrisDenham/osgopenvrviewer: An OpenSceneGraph/OSG viewer for VR devices compatible with OpenVR / SteamVR," [Online]. Available: https://github.com/ChrisDenham/osgopenvrviewer.

CHAPTER 5: SUMMARY AND FUTURE WORK

VIPRE, a volume rendering API, was adapted for use in VR. This entailed creating a virtual environment containing the volume and the user. An OSG viewer compatible with OpenVR-supporting HMDs was created using the osgopenvr software package. VIPRE volume rendering functions like density windowing, color transfer function selection, and clipping were assigned to VR controller inputs. Translation and rotation of the volume was enabled through the use of added transforms controlled by the user. The ray calculation stage of VIPRE's raytracing procedure was also modified so that volumes are correctly rendered in HMDs. This involved using the eye position as the ray's starting point rather than the screen position, and the same value was used for every fragment each frame.

Controller differences across the three headsets mentioned were addressed by creating a control scheme that maps similar finger movements to the same medical viewing functions. This mapping is determined at the start of the program, when the HMD type is queried. Functions that are assigned to the joystick and touchpad inputs, like density windowing and volume translation, can be accessed using controllers that only include one of the two inputs by pressing a modifier button. Newly released controllers can be added in the future by specifying the OpenVR axes for each button, trigger, touchpad, or joystick. This could be handled with an input configuration file that received updated information as new hardware becomes available.

Performance needs were addressed by limiting the number of sampling iterations while raytracing when necessary. It was found that GPU load increased when the user's eyes were located inside of the medical volume, so a cap on sampling iterations was applied under this circumstance. For the tested dataset using the previously specified hardware, framerates of 90fps

were achieved when far away from the volume, reaching a minimum of 25fps when inside the volume. Performance could be further improved by modifying the geometry surrounding the volume while clipping. If the clipped regions are removed, then rays cast through the clipped regions of the volume are partially or entirely diminished, and ray sampling iterations across empty parts of the volume are skipped.

VR headsets like the Oculus Rift, HTC Vive, and Samsung Odyssey are at the forefront of consumer-facing VR technology. Their high quality and relatively low price makes them attractive for use in recreational, educational, and medical settings. The VR medical volume viewer described in this thesis was developed to be compatible with these HMDs and provides real-time viewing and interaction through GPU raytracing.

REFERENCES

[1] D. Douglas, C. Wilke, J. Gibson, J. Boone and M. Wintermark, "Augmented Reality: Advances in Diagnostic Imaging," *Multimodal Technologies and Interaction,* vol. 1, no. 4, p. 29, 2017.

[2] K. Doi, "Current status and future potential of computer-aided diagnosis in medical imaging," *The British Journal of Radiology,* vol. 78, no. suppl_1, pp. s3-s19, 28 1 2005.

[3] B. C. Wood, S. R. Sher, B. J. Mitchell, A. K. Oh, G. F. Rogers and M. J. Boyajian, "Conjoined Twin Separation," *Journal of Craniofacial Surgery,* vol. 28, no. 1, pp. 4-10, 1 2017.

[4] F. Alfano, F. P. Garcia, J. E. O. Fisac, M. H. Conde, O. B. Zamora, F. A. Calvo, S. Lizarraga, A. Santos, J. Pascau and M. J. L. Carbayo, "Tumor localization using prone to supine surface based registration for breast cancer surgical planning," in *2018 IEEE 15th International Symposium on Biomedical Imaging (ISBI 2018)*, 2018.

[5] D. Selle, B. Preim, A. Schenk and H.-O. Peitgen, "Analysis of vasculature for liver surgical planning," *IEEE Transactions on Medical Imaging,* vol. 21, no. 11, pp. 1344-1357, 11 2002.

[6] M. W. Vannier, J. L. Marsh, J. O. Warren, M. W. Vannier, J. L. Marsh and J. O. Warren, "Three dimensional computer graphics for craniofacial surgical planning and evaluation," in *Proceedings of the 10th annual conference on Computer graphics and interactive techniques - SIGGRAPH '83*, New York, New York, USA, 1983.

[7] M. Martinez-Escobar, J. Leng Foo and E. Winer, "Colorization of CT images to improve tissue contrast for tumor segmentation," *Computers in Biology and Medicine,* vol. 42, no. 12, pp. 1170-1178, 1 12 2012.

[8] C. L. Ventola, "Medical Applications for 3D Printing: Current and Projected Uses.," *P & T : a peer-reviewed journal for formulary management,* vol. 39, no. 10, pp. 704-11, 10 2014.

[9] H. Fuchs, M. Levoy and S. M. Pizer, *Interactive Visualization of 3D Medical Data*, 1989.

[10] Lacroute and P. Gilbert, *Fast volume rendering using a shear-warp factorization of the viewing transformation,* Stanford University, 1996.

[11] C. Noon, E. Foo and E. Winer, "Interactive GPU volume raycasting in a clustered graphics environment," *Medical Imaging 2012: Image-Guided Procedures, Robotic Interventions, and Modeling,* vol. 8316, no. February 2012, 2012.

[12] B. A. Urban, L. E. Ratner and E. K. Fishman, "Three-dimensional Volume-rendered CT Angiography of the Renal Arteries and Veins: Normal Anatomy, Variants, and Clinical Applications," *RadioGraphics,* vol. 21, no. 2, pp. 373-386, 1 3 2001.

[13] D. B. Douglas, J. M. Boone, E. Petricoin, L. Liotta and E. Wilson, "Augmented Reality Imaging System: 3D Viewing of a Breast Cancer.," *Journal of nature and science,* vol. 2,

no. 9, pp. 1-6, 2016.

[14] J. Jerald and Jason, The VR book : human-centered design for virtual reality, Association for Computing Machinery and Morgan & Claypool, 2015, p. 599.

[15] "Oculus Rift," [Online]. Available: https://www.oculus.com/rift/#oui-csl-rift-games=robo-recall.

[16] "VIVE™," [Online]. Available: https://www.vive.com/us/.

[17] "HMD Odyssey," [Online]. Available: https://www.samsung.com/us/computing/hmd/windows-mixed-reality/xe800zaa-hc1us-xe800zaa-hc1us/.

[18] J. J. LaViola and J. J., "A discussion of cybersickness in virtual environments," *ACM SIGCHI Bulletin*, vol. 32, no. 1, pp. 47-56, 1 1 2000.

[19] L. J. Hettinger and G. E. Riccio, "Visually Induced Motion Sickness in Virtual Environments," *Presence: Teleoperators and Virtual Environments*, vol. 1, no. 3, pp. 306-310, 1992.

[20] "Xbox | Official Site," [Online]. Available: https://www.xbox.com/en-US/.

[21] C. A. Wingrave, B. Williamson, P. D. Varcholik, J. Rose, A. Miller, E. Charbonneau, J. Bott and J. J. LaViola, "The Wiimote and Beyond: Spatially Convenient Devices for 3D User Interfaces," *IEEE Computer Graphics and Applications*, vol. 30, no. 2, pp. 71-85, 3 2010.

[22] C. Ardito, P. Buono, M. F. Costabile, R. Lanzilotti and A. L. Simeone, "Comparing low cost input devices for interacting with 3D Virtual Environments," *Proceedings - 2009 2nd Conference on Human System Interactions, HSI '09*, pp. 292-297, 2009.

[23] L. Figueiredo, E. Rodrigues, J. Teixeira and V. Techrieb, "A comparative evaluation of direct hand and wand interactions on consumer devices," *Computers & Graphics*, vol. 77, pp. 108-121, 1 12 2018.

[24] T. Whitted, "An improved illumination model for shaded display," *Communications of the ACM*, vol. 23, no. 6, pp. 343-349, 1980.

[25] B. T. Phong and B. Tuong, "Illumination for computer generated pictures," *Communications of the ACM*, vol. 18, no. 6, pp. 311-317, 1 6 1975.

[26] "OpenGL - The Industry Standard for High Performance Graphics," [Online]. Available: https://www.opengl.org/.

[27] T. J. Purcell, I. Buck, W. R. Mark, P. Hanrahan, T. J. Purcell, I. Buck, W. R. Mark and P. Hanrahan, "Ray tracing on programmable graphics hardware," in *Proceedings of the 29th annual conference on Computer graphics and interactive techniques - SIGGRAPH '02*, New York, New York, USA, 2002.

[28] "dicom.offis.de - DICOM Software made by OFFIS - DCMTK - DICOM Toolkit," [Online]. Available: https://dicom.offis.de/dcmtk.php.en.

[29] W. D. Bidgood and S. C. Horii, "Introduction to the ACR-NEMA DICOM standard.," *Radiographics : a review publication of the Radiological Society of North America, Inc,* vol. 12, no. 2, pp. 345-55, 1 3 1992.

[30] "OsiriX DICOM Viewer | The world famous medical imaging viewer," [Online]. Available: https://www.osirix-viewer.com/.

[31] "RadiAnt DICOM Viewer," [Online]. Available: https://www.radiantviewer.com/.

[32] A. Rosset, L. Spadola and O. Ratib, "OsiriX: An open-source software for navigating in multidimensional DICOM images," *Journal of Digital Imaging,* 2004.

[33] L. Ebert, P. Flach, M. Thali and S. Ross, "Out of touch – A plugin for controlling OsiriX with gestures using the leap controller," *Journal of Forensic Radiology and Imaging,* vol. 2, no. 3, pp. 126-128, 1 7 2014.

[34] L. Gallo, A. P. Placitelli and M. Ciampi, "Controller-free exploration of medical image data: Experiencing the Kinect," in *Proceedings - IEEE Symposium on Computer-Based Medical Systems,* 2011.

[35] A. V. Reinschluessel, L. Raimondo, L. Reisig, M. Ruedel, D. Thieme, T. Vahl, G. Zachmann, R. Malaka, J. Teuber, M. Herrlich, J. Bissel, M. van Eikeren, J. Ganser, F. Koeller, F. Kollasch and T. Mildner, "Virtual Reality for User-Centered Design and Evaluation of Touch-free Interaction Techniques for Navigating Medical Images in the Operating Room," in *Proceedings of the 2017 CHI Conference Extended Abstracts on Human Factors in Computing Systems - CHI EA '17*, New York, New York, USA, 2017.

[36] O. Sommer, A. Dietz, R. Westermann and T. Ertl, "An interactive visualization and navigation tool for medical volume data," *Computers & Graphics,* vol. 23, no. 2, pp. 233-244, 1 4 1999.

[37] M. Hadwiger, P. Ljung, C. R. Salama and T. Ropinski, "Advanced illumination techniques for GPU volume raycasting," in *ACM SIGGRAPH ASIA 2008 courses on - SIGGRAPH Asia '08*, New York, New York, USA, 2008.

[38] C. Noon, J. Holub and E. Winer, "Real-time volume rendering of digital medical images on an iOS device," *IS&T/SPIE Electronic Imaging. International Society for Optics and Photonics,* no. March 2013, 2013.

[39] K. Mueller, T. Moller and R. Crawlis, "Splatting without the blur," in *Proceedings Visualization '99 (Cat. No.99CB37067).*

[40] "3D Slicer," [Online]. Available: https://www.slicer.org/.

[41] M. Martinez Escobar, B. Junke, J. Holub, K. Hisley, D. Eliot and E. Winer, "Evaluation of monoscopic and stereoscopic displays for visual–spatial tasks in medical contexts," *Computers in Biology and Medicine,* vol. 61, pp. 138-143, 1 6 2015.

[42] C. Cruz-neira, D. J. Sandin and T. A. Defanti, "Surround-Screen Projection-Based Virtual Reality: The Design and Implementation of the CAVE".

[43] J. J. LaViola, "Bringing VR and Spatial 3D Interaction to the Masses through Video Games," *IEEE Computer Graphics and Applications*, vol. 28, no. 6, pp. 83-85, 2008.

[44] I. E. Sutherland, "The ultimate display," *Multimedia: From Wagner to virtual reality*, pp. 506-508, 1965.

[45] D. B Douglas, C. A Wilke, D. Gibson, E. F Petricoin and L. Liotta, "Virtual reality and augmented reality: Advances in surgery," *Biology, Engineering and Medicine*, vol. 3, no. 1, pp. 1-8, 2017.

[46] J. E. Venson, J. Berni, C. S. Maia, A. M. Da Silva, M. D'Ornelas and A. Maciel, "Medical imaging VR: Can immersive 3D aid in diagnosis?," *22nd ACM Conference on Virtual Reality Software and Technology, VRST 2016*, Vols. 02-04-Nove, pp. 349-350, 2016.

[47] "Hardware — Sixense — next-level VR/AR motion tracking," [Online]. Available: https://www.sixense.com/platform/hardware/.

[48] "Razer United States | For Gamers. By Gamers.," [Online]. Available: https://www.razer.com/.

[49] D. Pinelle and N. Wong, "Heuristic evaluation for games," in *Proceeding of the twenty-sixth annual CHI conference on Human factors in computing systems - CHI '08*, New York, New York, USA, 2008.

[50] L. Zhou and C. D. Hansen, "A Survey of Colormaps in Visualization.," *IEEE transactions on visualization and computer graphics*, vol. 22, no. 8, pp. 2051-69, 2016.

[51] J. Egger, M. Gall, J. Wallner, P. Boechat, A. Hann, X. Li, X. Chen and D. Schmalstieg, "HTC Vive MeVisLab integration via OpenVR for medical applications," *PLOS ONE*, vol. 12, no. 3, p. e0173972, 21 3 2017.

[52] "ParaView Glance," [Online]. Available: https://kitware.github.io/paraview-glance/index.html.

[53] "OpenSceneGraph," [Online]. Available: http://www.openscenegraph.org/.

[54] "GitHub - ValveSoftware/openvr: OpenVR SDK," [Online]. Available: https://github.com/ValveSoftware/openvr.

[55] Chris Denham, "GitHub - ChrisDenham/osgopenvrviewer: An OpenSceneGraph/OSG viewer for VR devices compatible with OpenVR / SteamVR," [Online]. Available: https://github.com/ChrisDenham/osgopenvrviewer.

[56] T. Whitted, Turner, Whitted and Turner, "An improved illumination model for shaded display," in *Proceedings of the 6th annual conference on Computer graphics and interactive techniques - SIGGRAPH '79*, New York, New York, USA, 1979.

[57] K. E. Stull, M. L. Tise, Z. Ali and D. R. Fowler, "Accuracy and reliability of measurements obtained from computed tomography 3D volume rendered images," *Forensic Science International*, vol. 238, pp. 133-140, 1 5 2014.

[58] C. Rezk-Salama, K. Engel, M. Bauer, G. Greiner and T. Ertl, "Interactive volume on standard PC graphics hardware using multi-textures and multi-stage rasterization," in *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS workshop on Graphics hardware - HWWS '00*, New York, New York, USA, 2000.

[59] D. Nörenberg, W. H. Sommer, W. Thasler, J. D'Haese, M. Rentsch, T. Kolben, A. Schreyer, C. Rist, M. Reiser and M. Armbruster, "Structured Reporting of Rectal Magnetic Resonance Imaging in Suspected Primary Rectal Cancer," *Investigative Radiology*, vol. 52, no. 4, pp. 232-239, 4 2017.

[60] W. E. Lorensen, H. E. Cline, W. E. Lorensen and H. E. Cline, "Marching cubes: A high resolution 3D surface construction algorithm," in *Proceedings of the 14th annual conference on Computer graphics and interactive techniques - SIGGRAPH '87*, New York, New York, USA, 1987.

[61] S. IEEE Computer Society. Technical Committee--Computer Graphics., M. SIGGRAPH. and T. Ertl, Visualization 2000 : proceedings : October 8-13, 2000, Salt Lake City, Utah, ACM Press, 2000, p. 602.

[62] D. Ebert, H. (. Hagen, H. E. Rushmeier, C. IEEE Computer Society. Technical Committee--Computer Graphics. and P.-P. SIGGRAPH., Visualization '98 : proceedings, October 18-23, 1998, Research Triangle Park, North Carolina, ACM Press, 1998, p. 576.

[63] D. S. Ebert, P. (. Brunet, I. Navazo, S. N. Spencer, SIGGRAPH., European Association for Computer Graphics. and Association for Computing Machinery., Proceedings of the symposium on data visualisation 2002 : Barcelona, Spain, May 27-29, 2002, Eurographics Association, 2002, p. 259.

[64] M. Botsch and L. Kobbelt, "High-quality point-based rendering on modern GPUs," in *11th Pacific Conference onComputer Graphics and Applications, 2003. Proceedings.*.

[65] T. Aila and S. Laine, "Understanding the efficiency of ray traversal on GPUs," *Proceedings of the 1st ACM conference on High Performance Graphics - HPG '09*, p. 145, 2009.

[66] "VTK - The Visualization Toolkit," [Online]. Available: https://www.vtk.org/.

[67] "OSVR Organization · GitHub," [Online]. Available: https://github.com/osvr.

[68] A. Lasso, H. H. Nam, P. V. Dinh, C. Pinter, J.-C. Fillion-Robin, S. Pieper, S. Jhaveri, J.-B. Vimort, K. Martin, M. Asselin, F. X. McGowan, R. Kikinis, G. Fichtinger and M. A. Jolley, "Interaction with Volume-Rendered Three-Dimensional Echocardiographic Images in Virtual Reality," *Journal of the American Society of Echocardiography*, vol. 31, no. 10, pp. 1158-1160, 1 10 2018.

[69] D. F. Keefe and T. Isenberg, "Reimagining the Scientific Visualization Interaction Paradigm," *Computer,* vol. 46, no. 5, pp. 51-57, 5 2013.

[70] J. A. J. Jo Soo Yi, Youn ah Kang, John T. Stasko, "Towards a Deeper Understanding of the Role of Interaction in Information Visualization," vol. 13, no. 6, pp. 1-13, 2017.

[71] D. Datcu, S. Lukosch and F. Brazier, "On the Usability and Effectiveness of Different Interaction Types in Augmented Reality," *International Journal of Human-Computer Interaction,* vol. 31, no. 3, pp. 193-209, 2015.

[72] K.-S. Choi, X. He, V. C.-L. Chiang and Z. Deng, "A virtual reality based simulator for learning nasogastric tube placement," *Computers in Biology and Medicine,* vol. 57, pp. 103-115, 1 2 2015.

[73] Y. S. Boger, R. A. Pavlik and R. M. Taylor, "OSVR: An open-source virtual reality platform for both industry and academia," *2015 IEEE Virtual Reality Conference, VR 2015 - Proceedings,* pp. 383-384, 2015.

[74] P. Boechat, A. Hann, X. Li, J. Egger, M. Gall, X. Chen and D. Schmalstieg, "HTC Vive MeVisLab integration via OpenVR for medical applications," no. Fig 1, pp. 1-14, 2017.