Iowa State University
Digital Repository

Graduate Theses and Dissertations | Iowa State University Capstones, Theses and Dissertations

2016

Dynamic Volume Rendering of Functional Medical Data on Dissimilar Hardware Platforms

Joseph Scott Holub
Iowa State University

Follow this and additional works at: https://lib.dr.iastate.edu/etd

Part of the Computer Engineering Commons

Recommended Citation
Holub, Joseph Scott, "Dynamic Volume Rendering of Functional Medical Data on Dissimilar Hardware Platforms" (2016). *Graduate Theses and Dissertations*. 15932.
https://lib.dr.iastate.edu/etd/15932

This Dissertation is brought to you for free and open access by the Iowa State University Capstones, Theses and Dissertations at Iowa State University Digital Repository. It has been accepted for inclusion in Graduate Theses and Dissertations by an authorized administrator of Iowa State University Digital Repository. For more information, please contact digirep@iastate.edu.

المنارة للاستشارات

www.manaraa.com

Dynamic volume rendering of functional medical data on dissimilar hardware platforms

By

Joseph Scott Holub

A dissertation submitted to the graduate faculty
in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

Co-Majors: Human Computer Interaction and Computer Engineering

Program of Study Committee:
Eliot Winer, Major Professor
Stephen Gilbert
Phillip Jones
James Oliver
Rafael Radkowski

Iowa State University

Ames, Iowa

2016

Copyright © Joseph Scott Holub, 2016. All rights reserved.

المنارة للاستشارات

www.manaraa.com

Table of Contents

ABSTRACT vii

CHAPTER 1: INTRODUCTION 1

1.1 Medical Imaging 1

1.2 CT and MRI Technology 2

1.3 Volumetric Data 4

1.4 Overview of Volume Rendering 6

1.4.1 Iso-surface Surface Rendering 7

1.4.2 Image Splatting 7

1.4.3 Shear Warp 8

1.4.4 Texture Slicing 9

1.4.5 Raycasting 10

1.4.6 Implementing Different Volume Rendering Techniques 11

1.5 Functional Medical Imaging 12

1.6 Functional MRI 14

1.7 Motivation 15

1.8 Dissertation Organization 17

CHAPTER 2: VOLUME RENDERING 18

2.1 Computer Graphics and OpenGL 18

2.2 Raycasting 21

2.3 Volume Pipeline 22

2.3.1 Segmentation 24

2.3.2 Gradient Computation 25

2.3.3 Resampling 27

2.3.4 Classification 30

المنارة للاستشارات

www.manaraa.com

2.3.5 Coloring 30
2.3.6 Shading 32
2.3.7 Compositing....33
2.4 Advanced Raycasting Techniques....34
2.4.1 Lighting and Shadows 34
2.4.2 Clipping....37
CHAPTER 3: 4D VOLUME RENDERING 39
3.1 Rendering Speed Optimizations 39
3.1.1 Octrees 40
3.1.2 KD-Trees 42
3.1.3 Time Space Partitioning Trees (TSP)....43
3.2 Data Compression Techniques....43
3.2.1 Run Length Encoding (RLE)....44
3.2.2 Discrete Cosine Transform (DCT) 44
3.2.3 Wavelets....44
3.2.4 Index and Data Maps 45
3.3 Multi-Volume Rendering....45
3.3.1 Multi-volume Texture Slicing....47
3.3.2 Depth Peeling 49
3.3.3 Multi-volume Raycasting....50
3.3.4 Scene Graph Methods....52
3.3.5 Other Multi-volume Methods 54

المنارة للاستشارات
www.manaraa.com

3.4 Methods to Improve Understanding of 4D Data 54
3.4.1 Region of Interest (ROI) 55
3.4.2 Compressing 4D Volume Data for 3D Visualization 56
3.4.3 4D Animations 58
3.5 Current Volume Rendering Tools..59
3.5.1 Static Data Volume Renderers 60
3.5.2 Dynamic Data Volume Renderers 62
3.6 Research Issues 64
CHAPTER 4: VISUALIZING FMRI DATA USING VOLUME RENDERING IN VIRTUAL REALITY 67
4.1 Abstract 67
4.2 Introduction 68
4.3 Background 70
4.3.1 Volume Rendering..71
4.3.2 4D Volume Rendering Tools 73
4.3.3 4D Volume Rendering in VR 75
4.4 Methodology..76
4.4.1 Third-party APIs...78
4.4.2 OpenSceneGraph...79
4.4.3 VR Juggler...79
4.4.4 NIfTI File Reader 80
4.4.5 VIPRE-fMRI Framework 80
4.4.6 CAVE™ Implementation Details 81
4.4.7 Desktop Implementation 82

المنارة للاستشارات
www.manaraa.com

4.5 Results 83

4.6 Discussion 86

CHAPTER 5: ENABLING REAL-TIME VISUALIZATION OF FUNCTIONAL MAGNETIC RESONANCE IMAGING ON AN IOS DEVICE 88

5.1 Abstract 88

5.2 Introduction 89

5.2.1 Magnetic Resonance Imaging 90

5.2.2 3D Volume Rendering 90

5.2.3 Volume Raycasting Pipeline 91

5.2.4 4D Volume Raycasting 97

5.2.5 Mobile Raycasting 99

5.3 Materials and Methods 100

5.3.1 Raycasting with the iOS Metal Shading Language 101

5.4 Results 108

5.5 Discussion 112

5.6 Conclusions 115

CHAPTER 6: 4D FMRI VOLUME RAYCASTING ON DISSIMILAR PLATFORMS 116

6.1 Abstract 116

6.2 Introduction 117

6.2.1 Magnetic Resonance Imaging 118

6.2.2 3D Volume Rendering 119

6.2.3 Volume Raycasting Pipeline 120

6.2.4 4D Volume Raycasting 125

6.2.5 Functional Imaging Tools 126

المنارة للاستشارات

www.manaraa.com

6.3 Materials and Methods 128

6.3.1 Desktop and Virtual Reality Prototypes 129

6.3.2 Mobile Prototype 131

6.3.3 4D Volume Raycasting 132

6.4 Results 135

6.5 Discussion 139

6.6 Conclusions 141

CHAPTER 7: SUMMARY AND FUTURE WORK 143

7.1 Summary 143

7.2 Future Work 145

ACKNOWLEDGEMENTS 148

REFERENCES 149

المنارة للاستشارات

www.manaraa.com

ABSTRACT

In the last 30 years, medical imaging has become one of the most used diagnostic tools in the medical profession. Computed Tomography (CT) and Magnetic Resonance Imaging (MRI) technologies have become widely adopted because of their ability to capture the human body in a non-invasive manner. A volumetric dataset is a series of orthogonal 2D slices captured at a regular interval, typically along the axis of the body from the head to the feet. Volume rendering is a computer graphics technique that allows volumetric data to be visualized and manipulated as a single 3D object. Iso-surface rendering, image splatting, shear warp, texture slicing, and raycasting are volume rendering methods, each with associated advantages and disadvantages. Raycasting is widely regarded as the highest quality renderer of these methods.

Originally, CT and MRI hardware was limited to providing a single 3D scan of the human body. The technology has improved to allow a set of scans capable of capturing anatomical movements like a beating heart. The capturing of anatomical data over time is referred to as functional imaging.

Functional MRI (fMRI) is used to capture changes in the human body over time. While fMRI's can be used to capture any anatomical data over time, one of the more common uses of fMRI is to capture brain activity. The fMRI scanning process is typically broken up into a time consuming high resolution anatomical scan and a series of quick low resolution scans capturing activity. The low resolution activity data is mapped onto the high resolution anatomical data to show changes over time.

Academic research has advanced volume rendering and specifically fMRI volume rendering. Unfortunately, academic research is typically a one-off solution to a singular medical case or set of data, causing any advances to be problem specific as opposed to a general capability. Additionally, academic volume renderers are often designed to work

المنارة للاستشارات
www.manaraa.com

on a specific device and operating system under controlled conditions. This prevents volume rendering from being used across the ever expanding number of different computing devices, such as desktops, laptops, immersive virtual reality systems, and mobile computers like phones or tablets.

This research will investigate the feasibility of creating a generic software capability to perform real-time 4D volume rendering, via raycasting, on desktop, mobile, and immersive virtual reality platforms. Implementing a GPU-based 4D volume raycasting method for mobile devices will harness the power of the increasing number of mobile computational devices being used by medical professionals. Developing support for immersive virtual reality can enhance medical professionals' interpretation of 3D physiology with the additional depth information provided by stereoscopic 3D. The results of this research will help expand the use of 4D volume rendering beyond the traditional desktop computer in the medical field.

Developing the same 4D volume rendering capabilities across dissimilar platforms has many challenges. Each platform relies on their own coding languages, libraries, and hardware support. There are tradeoffs between using languages and libraries native to each platform and using a generic cross-platform system, such as a game engine. Native libraries will generally be more efficient during application run-time, but they require different coding implementations for each platform. The decision was made to use platform native languages and libraries in this research, whenever practical, in an attempt to achieve the best possible frame rates.

4D volume raycasting provides unique challenges independent of the platform. Specifically, fMRI data loading, volume animation, and multiple volume rendering. Additionally, real-time raycasting has never been successfully performed on a mobile device. Previous research relied on less computationally expensive methods, such as

المنارة للاستشارات
www.manaraa.com

orthogonal texture slicing, to achieve real-time frame rates. These challenges will be addressed as the contributions of this research.

The first contribution was exploring the feasibility of generic functional data input across desktop, mobile, and immersive virtual reality. To visualize 4D fMRI data it was necessary to build in the capability to read Neuroimaging Informatics Technology Initiative (NIfTI) files. The NIfTI format was designed to overcome limitations of 3D file formats like DICOM and store functional imagery with a single high-resolution anatomical scan and a set of low-resolution anatomical scans. Allowing input of the NIfTI binary data required creating custom C++ routines, as no object oriented APIs freely available for use existed. The NIfTI input code was built using C++ and the C++ Standard Library to be both light weight and cross-platform.

Multi-volume rendering is another challenge of fMRI data visualization and a contribution of this work. fMRI data is typically broken into a single high-resolution anatomical volume and a series of low-resolution volumes that capture anatomical changes. Visualizing two volumes at the same time is known as multi-volume visualization. Therefore, the ability to correctly align and scale the volumes relative to each other was necessary. It was also necessary to develop a compositing method to combine data from both volumes into a single cohesive representation.

Three prototype applications were built for the different platforms to test the feasibility of 4D volume raycasting. One each for desktop, mobile, and virtual reality. Although the backend implementations were required to be different between the three platforms, the raycasting functionality and features were identical. Therefore, the same fMRI dataset resulted in the same 3D visualization independent of the platform itself. Each platform uses the same NIfTI data loader and provides support for dataset coloring and windowing (tissue density manipulation). The fMRI data can be viewed changing

المنارة للاستشارات
www.manaraa.com

over time by either animation through the time steps, like a movie, or using an interface slider to “scrub” through the different time steps of the data.

The prototype applications data load times and frame rates were tested to determine if they achieved the real-time interaction goal. Real-time interaction was defined by achieving 10 frames per second (fps) or better, based on the work of Miller [1]. The desktop version was evaluated on a 2013 MacBook Pro running OS X 10.12 with a 2.6 GHz Intel Core i7 processor, 16 GB of RAM, and a NVIDIA GeForce GT 750M graphics card. The immersive application was tested in the C6 CAVE™, a 96 graphics node computer cluster comprised of NVIDIA Quadro 6000 graphics cards running Red Hat Enterprise Linux. The mobile application was evaluated on a 2016 9.7” iPad Pro running iOS 9.3.4. The iPad had a 64-bit Apple A9X dual core processor with 2 GB of built in memory.

Two different fMRI brain activity datasets with different voxel resolutions were used as test datasets. Datasets were tested using both the 3D structural data, the 4D functional data, and a combination of the two. Frame rates for the desktop implementation were consistently above 10 fps, indicating that real-time 4D volume raycasting is possible on desktop hardware. The mobile and virtual reality platforms were able to perform real-time 3D volume raycasting consistently. This is a marked improvement for 3D mobile volume raycasting that was previously only able to achieve under one frame per second [2]. Both VR and mobile platforms were able to raycast the 4D only data at real-time frame rates, but did not consistently meet 10 fps when rendering both the 3D structural and 4D functional data simultaneously. However, 7 frames per second was the lowest frame rate recorded, indicating that hardware advances will allow consistent real-time raycasting of 4D fMRI data in the near future.

المنارة للاستشارات
www.manaraa.com

CHAPTER 1: INTRODUCTION

From the earliest cave paintings to today's boardroom bar charts, visualization strategies have been used to tell a story. Computers provide the ability to perform complex computations, process large amounts of data, and increasingly visualizations are computer generated. Medical data, such as patient charts, medical history, lab test results, and medical imaging, is one of the fastest growing sources of data with 150 exabytes (i.e. 150 billion gigabytes) in 2011 and growing rapidly [3]. The most significant contributor to this increase is medical imaging data, such as digital X-rays, Computed Tomography (CT), ultrasounds, and Magnetic Resonance Imaging (MRI).

1.1 Medical Imaging

Medical imaging began in 1895 when a German physicist by the name of William Conrad Rontgen took an X-ray of his wife's hand [4]. The idea of seeing inside of the human body without surgery was a paradigm shift that has since spawned the creation of similar imaging techniques such as Magnetic Resonance Imaging (MRI), Computed Tomography (CT), Positron Emission Tomography (PET), and Ultrasound to name a few of the more commonly used methods.

Computer Tomography is a medical imaging technology that uses ionizing radiation from an x-ray tube to generate cross-sectional slices of the patient's body [5]. The rotating x-ray tube generates a series of 2D slice images along a single axis of rotation at consecutive intervals. Medical imaging is the most common application of CT scans, but there are other industrial applications as well such as nondestructive evaluation of mechanical structures.

Magnetic Resonance Imaging is a completely different technology used to accomplish the same task. Instead of x-rays, the MRI uses strong magnetic fields and

المنارة للاستشارات
www.manaraa.com

radio waves to create the 2D slice images of the patient [6]. The radio waves are used to resonate magnetically charged nuclei like Hydrogen and the resulting resonance is used to create the images seen in the 2D slices. MRIs have been found to be superior to CT scans when viewing soft tissue and CT scans are superior when viewing bone and muscle. This is the reason MRI technology is typically used to diagnose issues in the brain, cartilage and tendons in joints, and other organs composed of softer tissues. A more in-depth description of the underlying MRI technology will be covered in Section 1.6.

Today's CT and MRI scanners are capable of generating 2D slices of the order of 512 x 512, 1024 x 1024 pixels, and sometimes higher. The higher the number of pixels, the more detailed information can be obtained. While 1024 x 1024 resolution is not considered high relative to the current TVs, computer monitors, and cell phones, it is high for medical imaging because CT and MRI scanners are limited by physics. CT and MRI scanners rely on sending a field into the human body and measuring changes in that field, making medical imaging limited by both the strength of the field produced and the sensitivity of the instruments used to measure the field. The distance between the slices is also shrinking with the newest scanners able to achieve 1mm distance between slices, allowing smaller anatomical items to be scanned with more precision.

1.2 CT and MRI Technology

Computer Tomography is a medical imaging technology that uses ionizing radiation from an x-ray tube to generate cross-sectional slices of the patient's body [5]. An x-ray generating tube is placed to one side of the patient and x-rays are emitted in the direction of the patient with an x-ray sensor on the opposite side of the tube. This process generates a single planer image. The tube and sensor are then rotated slightly

المنارة للاستشارات
www.manaraa.com

around an axis relative to the patient and take another planer image. This process is continued until a complete scan of the patient is performed. Medical imaging is the most common application of CT scans, but there are other industrial applications such as nondestructive evaluation of mechanical structures.

Magnetic Resonance Imaging (MRI) is a technology built upon the use of magnetic fields and radio frequency pulses. MRIs work by placing the scanned object, often a person, into a powerful unidirectional magnetic field. The purpose of the magnetic field is to align nuclei in the object, typically hydrogen. Nuclei with an odd number of protons or neutrons exhibit the quantum-mechanical phenomenon known as Nuclear Magnetic Resonance (NMR), or a spin causing a weak magnetic field. A specific radio frequency (RF), known as the Larmor frequency, is determined by the magnetic field strength and is able to resonate the atoms and knock them out of alignment [7]. When this RF signal is removed, the atoms oscillate back into alignment. The time taken for the atoms to oscillate back into alignment is measured to determine properties like tissue density. This process would provide one sample for the entire object. To obtain samples at 3D points in the space of the object, gradient magnetic fields are used to change the magnetic field strength along the object and thus change the Larmor frequency required to resonate the atoms along the object [6,7].

Felix Bloch and Edward Purcell first discovered MRI technology in 1946. It was not until the early 1970s that is was used for imaging purposes. Its use has since exploded with the highest number of users being in the medical industry [6].

المنارة للاستشارات

www.manaraa.com

1.3 Volumetric Data

While CT and MRI scanners can obtain slices from any angle or direction, they are typically obtained by scanning along the axis of the body, from head to feet or vice versa, as seen in Figure 1. "Scanning", as discussed in the preceding paragraph, is the process of sending a field into the human body and measuring the changes to the field to determine physical characteristics within the body, generally tissue densities compared to air or water. For CT scanning, the field is an x-ray and for MRI the field is a magnetic field. These 2D slices can then be combined to create a single 3D block of data representing the entire scan of the patient, as shown in Figure 1, known as volumetric data. The data itself can be stored in various ways depending on the application itself. Medical data often stores volume data as a set of files, where each file contains a single 2D image slice as well as patient specific data (e.g., name, age, date of scan, etc.).

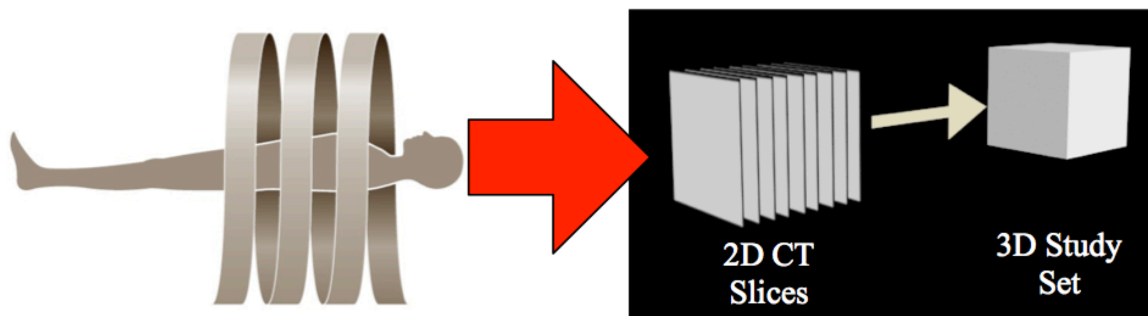


Figure 1: CT scanner showing the scanned body path to a 3D volume.

There are many different file types associated with volumetric data depending on the industry and their use, such as Digital Imaging and Communications in Medicine (DICOM), Neuroimaging Informatics Technology Initiative (NIfTI), Analyze/SPM, Medical Imaging NetCDF (MINC), and Analysis of Functional NeuroImages (AFNI). For example, DICOM is a very common file type used in the medical industry to store volumetric data

المنارة للاستشارات
www.manaraa.com

of body parts representing a single time step. However, when looking to store volumetric data associated with neuroimaging, the neuroimaging community created the NIfTI format to accommodate time-based scanning. NIfTI was designed to store both high resolution single time-step anatomical scans of the brain as well as lower resolution functional scans of brain activity over time. While the original purpose of the NIfTI standard was aimed at storing brain scans, the NIfTI format is generalizable to any fMRI data. This research will focus on using the NIfTI standard because of its widespread adoption in the fMRI area.

Storing modeling and simulation data in volumetric form is critical in medical imaging but is also in other industries as well. The meteorological industry has used volumetric data to visualize cloud formation and rain patterns. There has also been research done on using volumetric data for complex modeling of trends and anomalies in different phenomena such as ocean turbulence, precipitation, hurricanes, and acid rain [8]. For example, a 3D visualization of hurricane wind speeds can be seen in Figure 2. Geologists have been using CT scans to better visualize geological information like porosity, pressure, and temperature in three dimensions from core samples for more than a decade [9]. At a smaller scale, micro-biologists have been using volumetric data to visualize microscopic organisms in high-resolution without disturbing them [10].

المنارة للاستشارات
www.manaraa.com

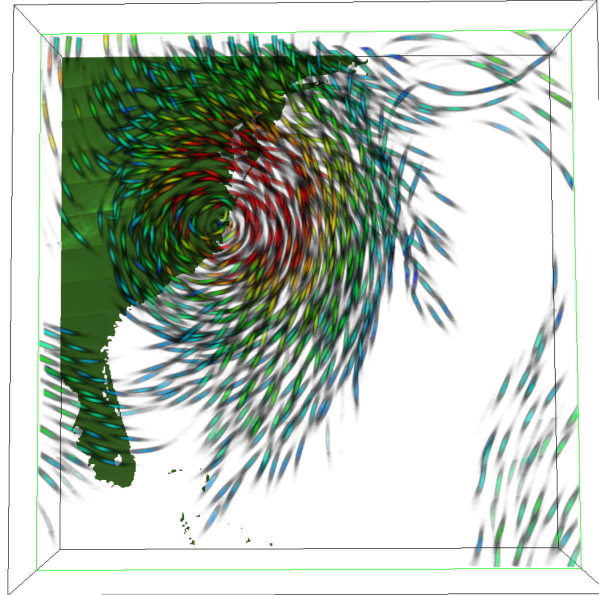


Figure 2: Hurricane wind speeds visualized in 3D [8]**.**

1.4 Overview of Volume Rendering

Volume rendering is the process of taking volumetric data and visualizing it. Unlike surface rendering that represents an object as a series of vertices making up an outer shell to show the object's shape, volume rendering sees an object as a three-dimensional lattice of vertices similar to a Rubik's Cube. A volumetric dataset does not contain any defined surfaces or edges. Therefore, surface rendering techniques are inadequate to use for visualizing volumetric data. Surface rendering does not contain values inside the object shell, whereas volume rendering contains values at vertices throughout the mass of data being represented.

New rendering techniques were required to take advantage of the full three-dimensions of data provided in volumetric representations. The main classes of volume rendering techniques are Iso-surface Surface Rendering, Image Splatting, Shear Warp, Texture Slicing, and Raycasting.

المنارة للاستشارات

www.manaraa.com

1.4.1 Iso-surface Surface Rendering

Iso-surface surface rendering was created to reduce the complexity of volume rendering by representing the volumetric data as a surface comprised of geometric primitives [11]. The Marching Cubes algorithm is the most popular rendering method for extracting a surface from the volumetric data [12]. The major advantage to iso-surface surface rendering is computational efficiency. However, there are several disadvantages including the accuracy of the representation. The geometric primitives can only approximate the surface of the volume, therefore small details can be lost when approximated. The other major drawback is the loss of internal data because the volume is represented as a surface. Figure 3 shows an iso-surface surface rendering of a human skull.

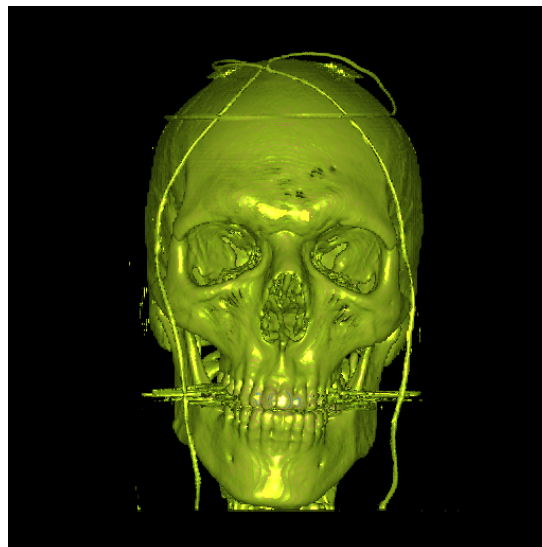


Figure 3: Iso-surface rendering (http://www.aravind.ca/images/ivis_gallery/isoColour.png)

1.4.2 Image Splatting

Image splatting is a technique using overlapping basis functions to represent the volume. The most commonly used basis functions are Gaussian kernels [13]. The basis

المنارة للاستشارات

www.manaraa.com

functions are projected to the screen as a superposition of pre-integrated 3D kernels, referred to as 2D footprints. One of the advantages of image splatting is that only the volume points are stored, ignoring the empty space around the volume. The disadvantage to image splatting are color bleeding, aliasing, and blurring due to blending the splats [14–16]. Figure 4 shows an example of a human head rendered using Image Splatting [17].

Figure 4: Image splatting [17]

1.4.3 Shear Warp

Shear warp volume rendering [18,19] determines the face of the volume that is most parallel to the viewing plane and then casts rays through the volume starting at the base plane. The resulting image is then projected onto the image plane through a 3D transformation and a 2D image resampling operation. The advantage of shear warp rendering is the computational efficiency achieved from only sampling each voxel once. The disadvantage is the image quality due to the sampling technique used. The image also becomes more distorted as the base plane goes from perfectly parallel to the image

المنارة للاستشارات

www.manaraa.com

plane to closer to perpendicular due to the 3D projection transform. Figure 5 shows an example of shear warp rendering when used on a chest scan of a human.

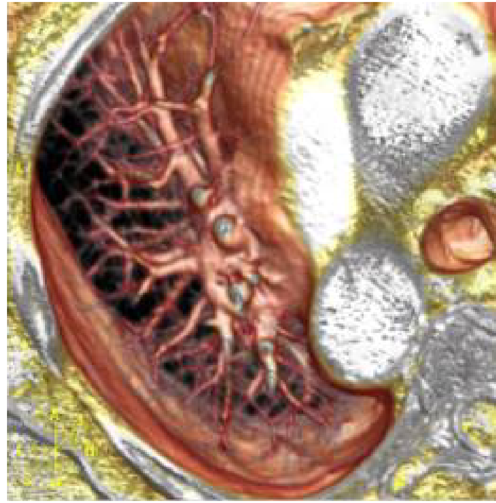


Figure 5: Shear warp [16]

1.4.4 Texture Slicing

Texture slicing [20,21] is a technique that slices the vo

parallel to the image plane. These parallel planes are then c

back-to-front compositing to achieve the final image. The ad

are a higher quality image than the previous techniques with

efficiency. One disadvantage is the requirement to recomputed the parallel planes every time the view matrix is updated. Figure 6 shows an example of texture slicing when used to visualize a human head scan.

المنارة للاستشارات

www.manaraa.com

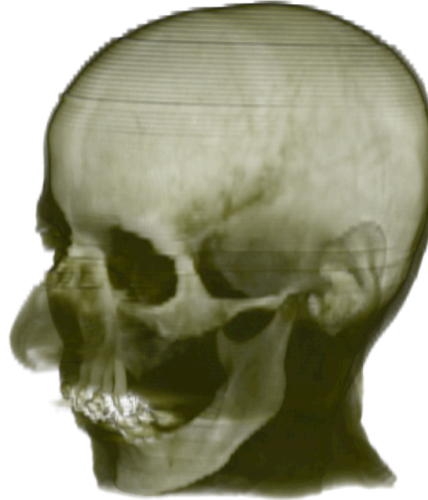


Figure 6: Texture slicing [17]

1.4.5 Raycasting

Raycasting [22,23] is a technique that involves casting rays from each pixel of the image plane in the view direction through the volume. The rays take multiple samples of the volume and composite them together to achieve a final pixel value. Raycasting is widely accepted as the best quality volume rendering technique of those discussed here. Figure 7 shows an example of raycasting using a human chest CT scan. Raycasting will be discussed in more detail in Chapter 2.

المنارة للاستشارات

www.manaraa.com

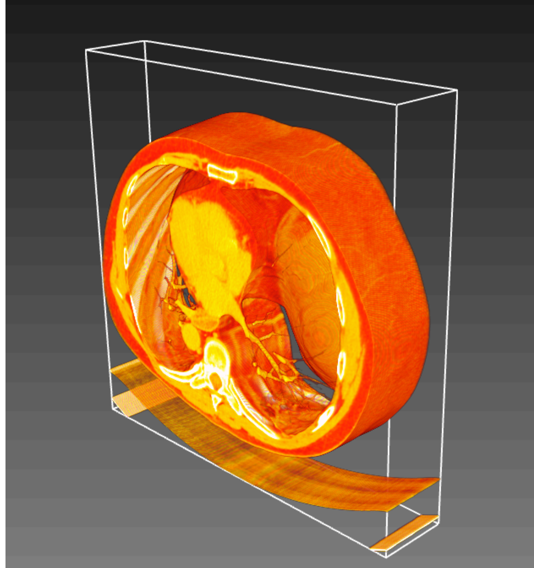


Figure 7: Raycasting

1.4.6 Implementing Different Volume Rendering Techniques

Each of the discussed volume rendering techniques allows for three-dimensional information to be rendered each frame to varying visual accuracy and computational cost. Iso-surface surface rendering and image splatting are the least computationally expensive, but the image quality is substantially below the other techniques. Texture slicing and raycasting are both computationally expensive but provide a higher quality image.

All of these volume rendering techniques are more computationally expensive than surface rendering. However, there have been many advances in software optimization and hardware acceleration that has allowed volume rendering to achieve real-time speeds [11]. Many of these improvements are from advances in graphics hardware and a programmable graphics pipeline [24].

The medical industry has been the leader in volume rendering research due to widespread use of CT and MRI scans for diagnoses and treatment planning. However,

المنارة للاستشارات

www.manaraa.com

volume rendering can be used in many other industries. Childhood biology education is one example where volume rendering can be used to teach children about the internal anatomies of animals, as shown in Figure 8 [25]. Volume rendering allows the children to see inside the animals without requiring dissection. In addition, because the data is digital, they can learn about many more types of animals at the push of a button. It is possible in the same way to use volume rendering to see inside objects that are too valuable to be dissected. Archeologists can use volume rendering to nondestructively observe the inside of artifacts such as mummies, as seen in Figure 9 [26]. Another example of volume rendering in the medical field, but for a different application, is surgical training. It is now possible to take a scan of an actual patient and practice the surgery multiple times before ever stepping into the operating room [27].

Figure 8: Voumetric visualization of a biology frog (http://www.cs.utah.edu/~jmk/images/frog2.jpg)

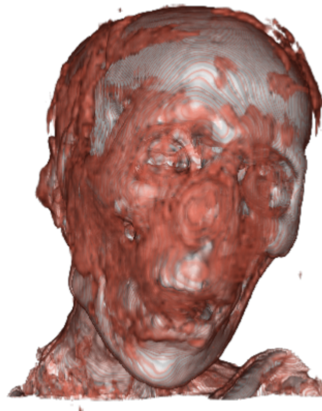


Figure 9: Iso-surface rendering of a mummy (http://www.cs.utah.edu/~pnathani/courses/cs5630/Project4/DirectVolumeRendering1.png)

1.5 Functional Medical Imaging

Unlike traditional medical imaging, which focus on viewing inside the body to identify anatomical features statically, functional medical imaging looks at these features over time. For example, this results in a doctor viewing not just a heart, but how that heart beats. Functional imaging is becoming more prevalent for various diagnosis and

المنارة للاستشارات

www.manaraa.com

treatment planning. Currently, it is heavily used in identifying brain activity. This fundamental change to what is being viewed changes the technology requirements. Functional scanning technology must be able to look inside a patient, but it must be able to do so much quicker than standard methods. Functional imaging technology must be able to perform a complete scan, yet capture the necessary motions at the characteristic scales. If a heart pulses at 80 beats per minute, the scan must capture it accurately. CT, MRI, and ultrasound technologies are often used for functional imaging.

Functional CT scanners are used to capture anatomical movements like a beating heart or the movement of an arm. They have a faster scan time relative to an MRI machine but CT scans do not provide sufficient contrast when looking at soft tissues like the brain. Imaging contrast is critical because it indicates how discernable different anatomical parts are from one another. The higher the contrast, the easier it is to discern differences and make necessary observations. For brain scans, MRI machines are used because they provide a higher contrast for soft tissue, but require a longer relative scan time.

Data storage becomes a concern when moving from 3D to 4D medical imaging. The data storage requirement multiplies the original 3D size by the number of time steps scanned. For a simple example, suppose a 3D scan of the human heart may require 300 MB of storage, but a 4D scan of that heart over a one-minute period with a scan every two seconds would result in 9 GB or 30 times the size. This dramatic increase in data creates problems with appropriate file format types, computer hard drive space, and transferring over networks.

المنارة للاستشارات

www.manaraa.com

1.6 Functional MRI

MRI imaging technology has been around for the last 50 years but functional MRI (fMRI) technology has only been available for the last 20. Functional MRIs are often used synonymously with brain activity scans, even though that is not their only use [28]. The impact of fMRI on researchers has been dramatic. In 1992, there were zero publications with the word fMRI in the title, abstract, or a keyword. In 2005, there were close to 2500 research publications meeting those criteria, showing explosive growth in the fMRI research being performed [29]. The impact on the world has been just as significant, with impacts in the areas of science, clinical practice, cognitive neuroscience, mental illness, and society [30].

The reason for the large growth in functional imaging is a change in the scanning methodology. In static imaging, the time taken to scan an object is not really an issue as long as the person can be held relatively still. The goal with static imaging is to capture the highest resolution slice scans with the smallest slice spacing and longer scanning times accomplishes this high resolution scanning. This provides medical personnel the most coverage area and the ability to identify very small anatomical features or tumors.

Functional imaging would ideally provide the same high resolution scans with small slice spacing multiple times a second. In practice, this is not always possible because of MRI technology is limited by how slowly it takes atoms to relax back into alignment after knocking them out of alignment. This alignment time can be shortened by using stronger magnets but it will never be able to be completely overcome. Medical imaging research has created a work around that combines one high resolution scan (slow) of the anatomy of the brain with a series of lower resolution scans (fast) to measure brain activity. The brain activity data can then be combined with the high resolution anatomical data to achieve a view of what parts of the brain are activated.

www.manaraa.com

This difference between static and functional imaging created a need for new ways to store the data. The DICOM format was designed for storing static data like X-rays and CT scans, but it is not easily extended to 4D time-varying data. Instead, new data formats, such as NIfTI [31] and Analyze [32], were envisioned to work with the functional imaging requirement of a structural and a functional scan in the same data. The Neuroimaging Informatics Technology Initiative (NIfTI) data format is one of the most popular and will be the data format used in this research.

1.7 Motivation

The benefits of medical imaging technologies are extensive with the most prominent being the ability to diagnose patients without using invasive methods. Using functional imaging technologies, doctors and researchers are learning new things about how the body works. Surprisingly, this is just beginning to scratch the surface of what can be learned using these technologies. As scanning technologies improve, the amount that can be learned about the human body will only be limited by our ability to process and understand the data.

Creating visualization tools for interpreting large amounts of medical data is a significant challenge. Many researchers have spent their careers building tools and methods for visualizing medical imaging data. The advancements have been significant, but have largely remained in the realm of research and have not expanded beyond. These advancements often are specific to one project or application and not general enough to be used for other purposes. The applications described earlier in this section are visualization tools, but they are limited to their specific use case, data format, and hardware platform. Software developers that are interested in creating commercial tools for medical professionals, must wade through the mass of literature on volume rendering

المنارة للاستشارات
www.manaraa.com

and determine if any specific implementation could be generalizable enough for their use on their hardware platform. Volume rendering techniques that may work on a desktop computer with an advanced graphics card for 3D data, probably won't work on a virtual reality or mobile device with 4D data.

Another issue is that the use of medical imaging technology has been reserved for high-end desktop computers and laptops. Mobile smartphone adoption has increased dramatically in the US and worldwide [33]. Virtual reality may be primed to make the same sort of meteoric rise in adoption with commercial companies jumping into the virtual reality market, such as Facebook buying Oculus Rift in 2014 for $2 billion [34]. This dramatic adoption of mobile computers, smartphones and tablets, and the new commercialization of virtual reality technologies has opened the door to more platforms than previously available. There is a need to drive medical imaging technology toward supporting these types of devices, because it is becoming more common for medical professionals to have access to mobile computing platforms, than traditional desktop computers. Training and simulation are also a significant focus for medical professionals and the use of virtual reality for training will only increase in the future.

It is therefore necessary to determine whether it is feasible to build 4D volume raycasting tools across these different hardware and software platforms. It is especially important to determine whether real-time interaction can be achieved. This is critical in volume rendering where user interaction with the volume provides an additional sense of depth. This research will explore the feasibility of developing 4D volume raycasting capabilities on desktop, mobile, and immersive virtual reality platforms.

www.manaraa.com

1.8 Dissertation Organization

The remainder of this dissertation will cover the relevant background on 3D volume rendering in Chapter 2. Chapter 3 will present the current research contributing to 4D volume rendering and the research issues addressed by this work. Chapters 4 will discuss work on building a generic NIfTI data loader. Chapter 5 will describe implementing 4D volume raycasting on a mobile device. Chapter 6 will present the work done in determining the feasibility of developing a generic 4D volume raycasting functionality across dissimilar hardware and software platforms. Chapter 7 will summarize the findings of this research and discuss possible future directions.

المنارة للاستشارات

www.manaraa.com

CHAPTER 2: VOLUME RENDERING

There has been a substantial amount of research done in the area of volume rendering. This chapter will cover the basics of graphics programming, volume rendering, and the specific research being done in 4D volume rendering. This section will evaluate the different methods based on computational efficiency and visual quality.

2.1 Computer Graphics and OpenGL

Volume rendering is a computationally expensive method for rendering data that relies heavily on the understanding of computer graphics and rendering pipelines. Many of the rendering pipelines available today are similar in theory, but different in implementation. This section will focus on describing the OpenGL rendering pipeline and how this pipeline takes a 3D scene and converts it into a 2D image to display on screen.

Graphics scenes, like volumetric data, are comprised of multiple objects with their own characteristics such as geometry, transformations, textures, lighting, bump maps, and shading. These characteristics describe how the object should look on the screen. Developers define all of the objects and their characteristics to set up a scene before it is handed to a renderer to convert everything into a single graphics frame to display on the screen. In computer graphics applications, this rendering takes place continuously until the application is terminated, this is known as the rendering loop. A great analogy is film based movies, where each frame in the film is a single static image, but by showing those images very quickly to the audience, there is motion. In computer graphics, the graphics card is generating those static frames on the fly and showing them to the viewer fast enough to create motion.

The OpenGL pipeline, shown in Figure 10, outlines the path that scene objects take to go from being a 3D scene to a 2D frame. In this flow chart of the rendering

المنارة للاستشارات
www.manaraa.com

pipeline, object data begins as a Display List. From there, data takes two different paths depending on whether it is vertex data (vertices, lines, polygons) or pixel data (pixels and images).

The vertex data (vertices, lines, polygons) goes through the vertex path in the rendering pipeline where at the minimum, the vertex data is transformed from 3D scene space to 2D screen space. The vertex path is labeled the Geometry Path in Figure 10. Advanced operations, such as computing texture coordinates, lighting, and material properties, are also performed here. After the scene is converted into 2D screen space, clipping and culling are performed to generate the final primitives.

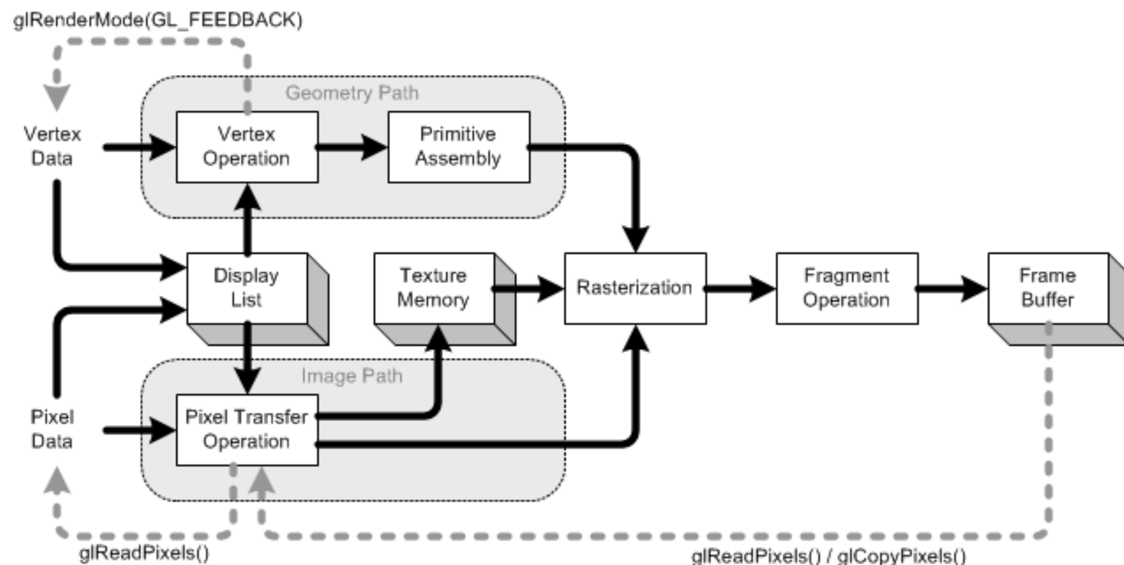


Figure 10: OpenGL Rendering pipeline (http://www.songho.ca/opengl/gl_pipeline.html)

The pixel data (pixels and textures) also starts as a Display List, but continues down a different path than the vertex data. The pixel path is labeled the Image Path in Figure 10. The pixel data is first unpacked from the current format into the proper number of components. The data is then scaled, biased, and processed by a pixel map. The pixel data is then stored in texture memory or sent to the rasterization step.

www.manaraa.com

Originally, both the vertex and pixel pipelines were fixed functionality. There was no way to change how the GPU handled data. This has since changed from a fixed pipeline to a programmable pipeline, where developers can write their own ways to handle vertex and pixel data known as shaders. This change to a programmable pipeline has revolutionized the way computer graphics, specifically volume rendering, is done.

Both the vertex and pixel data come together in the rasterization step. The rasterizer takes both the vertex and pixel data and converts them into fragments. The fragments represent individual pixels in the framebuffer and are assigned both a color and a depth. Before being assigned to the framebuffer, the fragments go through a series of fragment operations, which can contain any or all of the following, texturing, fog, scissor test, alpha test, stencil test, depth-buffer test, blending, and dithering. Only after going through these fragment operations, is the fragment stored in the framebuffer and finally displayed to the viewer in the final image.

One example of the programmable graphics pipeline at work is bump mapping. Figure 11 shows how bump mapping works with a smooth sphere represented in the left and the bump map texture in the middle. The pixel pipeline can be programmed using a fragment shader to modify the sphere's normal based on the bump map, so when Phong shading is used, there is the appearance of a bumpy texture. This technique is very effective because it reduces the number of polygons required for rendering while providing more realism. This is just one example of programming the rendering pipeline. More information on the OpenGL rendering pipeline can be found in the OpenGL Programming Guide [35].

المنارة للاستشارات
www.manaraa.com

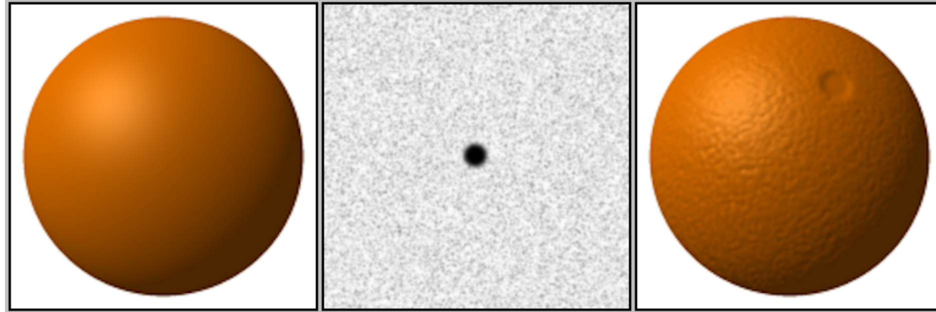


Figure 11: Bump mapping using a sphere and a texture to make an orange.

2.2 Raycasting

The various forms of volume rendering have been described in Chapter 1.4 with raycasting being one of the most popular. When compared to other volume rendering techniques, raycasting is considered by many to offer the highest visual quality. Raycasting is referred to as a direct volume rendering technique [36]. It involves casting rays from each pixel in the framebuffer through the volume in the view direction. All points along the rays' path that intersect with the volume are sampled and composited to generate the final pixel color. This process will be outlined in the resampling and compositing sections of this paper. Advances in computational and graphics processing speeds, along with research to optimize raycasting including early ray termination and empty space skipping, allow for real-time use of it on many hardware platforms.

When volume rendering on the central processing unit (CPU), it is common to divide the screen into sections to try and parallelize the process. Each section is given to one of the CPU cores to render. The sections rendered by each core are then combined together to get the final image to show [37–39]. This provides a faster render than a single threaded application, but with most commodity computers topping out at eight cores, the maximum speed up is eight times.

المنارة للاستشارات

www.manaraa.com

Raycasting is a parallel method by nature with all rays executing independently of each other. This makes raycasting an ideal method for use on massive parallel architectures like graphics processing units (GPUs). There has been a significant movement in parallel computing to use commodity GPUs containing upwards of 3,000 computational cores [40]. Using a GPU as a parallel computation device was made possible by making the vertex and fragment shaders programmable. Research has shown significant improvements in computational capability over modern CPUs [41,42]. To take advantage of this, multiple software packages and languages have been created for programming the GPU [43–45]. This research will utilize GLSL for GPU programming [45].

The main drawback to using a GPU for volume raycasting is the amount of texture memory on the graphics card. It can be a challenge to fit all the 3D voxel data as well as gradients and transfer functions into a GPU's memory. This limitation is slowly being overcome with some of the newest high-end GPUs containing as much as 12 gigabytes of texture memory, such as the NVIDIA Quadro K6000. While the K6000 is a commodity card, it will take a few years before the typical workstation has this type of power. This mentions nothing of the challenge of mobile devices where the texture memory limitation is significantly less than 12 gigabytes.

Raycasting was selected as the volume rendering method for this research. Therefore, the remainder of this paper will presume the use raycasting when discussing volume rendering methods.

2.3 Volume Pipeline

Volume rendering begins with acquiring a volumetric dataset. As described in the introduction, there are many ways to acquire volume data as well as many formats.

المنارة للاستشارات
www.manaraa.com

However, volumetric data is all comprised of a set of samples, known as voxels, in the three dimensions (i.e. *x, y,* and *z*). Each point contains a measured value, *v*. The measured value of the voxel can vary widely depending on the application. For example, the measured value could be a property of the data, such as density, pressure, or temperature.

In medical imaging, the voxel values typically represent tissue densities and are a one-dimensional value. In fMRI brain scans, the voxel value is a one-dimensional value representing the blood oxygenation level-dependent (BOLD) signal change [28]. This is used because neural activity has been linked with local changes in brain oxygen content [46]. The change of oxygen levels over time also requires the addition of a time component, *t*, to the volume data sample, resulting in a (*x,y,z,t,v*) for each sample.

Volumetric data samples are not required to be in any specific order or orientation but it typically follows an evenly spaced rectilinear grid. Medical imaging follows this ordered approach to volumetric data. Voxel data is typically obtained by scanning along the axis of the body, from head to feet or vice versa, and perpendicular to this axis as seen in Figure 1. These 2D slices can then be combined to create a single 3D block of data representing the entire scan of the patient, as was previously shown in Figure 1.

The volumetric data is then run through the volume-rendering pipeline. The actual pipeline itself might differ depending on the application or the type of data being used, but the basic pipeline is the same for all volume renderers. Figure 12 shows the basic volume-rendering pipeline. Not all steps of this pipeline are used in all cases, nor are they always performed in this specific order.

المنارة للاستشارات
www.manaraa.com

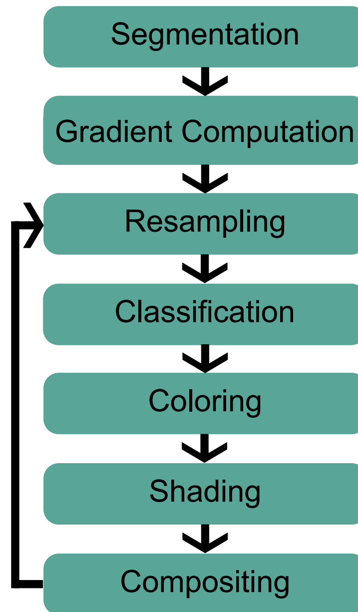


Figure 12: Generic volume-rendering pipeline

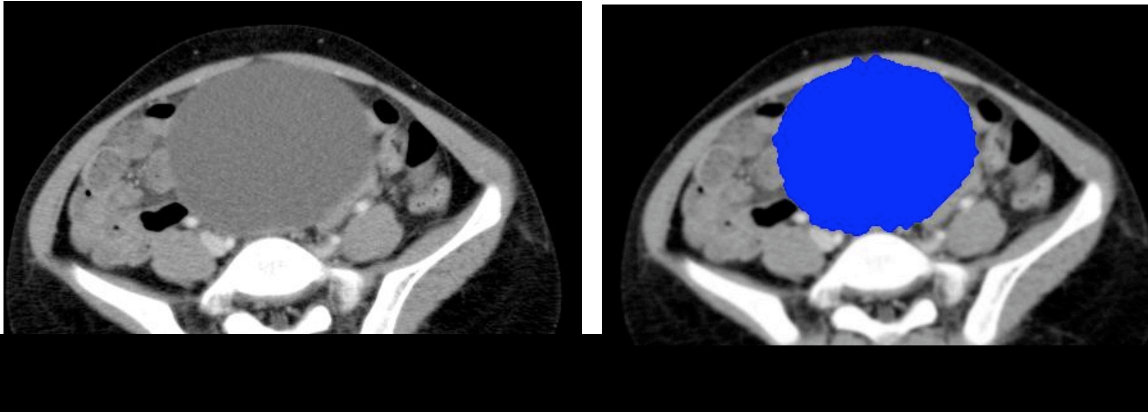
2.3.1 Segmentation

Segmentation is an optional step in the volume-rendering pipeline that partitions a single volume into multiple sub-volumes. The segmentation step is almost always performed just once before rendering. This is used extensively in the medical field when attempting to identify tumors or other masses within a patient. Education also uses this technique extensively when teaching anatomy, where it is necessary to show a single organ instead of the entire body. Visualizing segmented data is typically done in one of three ways, visualizing the segmented data separate from the original volume, render the segmented volume as a surface within the original volume, or tag and store each voxel contained within the original volume. These three methods can then be used in the rendering process to change how the segmented volume is visualized to give it greater contrast to the rest of the volume. This is typically done by rendering the segmented volume in a different color or as a higher opacity than the rest of the volume. More details on segmentation can be found in these references [41,47,48]. Figure 13 shows

المنارة للاستشارات

www.manaraa.com

an example of tumor segmentation using fuzzy logic to classify tumor versus healthy tissue [49].



shading techniques. The gradient is computed as the amount of variation between the voxel and its neighboring voxels and is represented as a three-dimensional vector. Gradients can be computed using many different methods with the tradeoff being, the more complex and accurate the gradient method, the more computation required. This is typically not a problem as the gradient is only calculated once and stored for use later in rendering.

Three of the more commonly used gradient methods are the Central Different Gradient Estimator, the Intermediate Different Operator, and the Sobel Operator [16,50,51]. The main difference between these different gradient methods are the

المنارة للاستشارات
www.manaraa.com

number of neighboring voxels they sample. The Central and Intermediate methods both use six neighboring voxels to compute the gradient, making these methods computationally fast and relatively easy to implement. Figure 14 shows the central difference gradient estimator for the green voxel in the middle. The surrounding six blue voxels are used to calculate the gradient vector. Calculating the resulting vector is done using the equations in (1) where the intensity value of each neighboring voxel along each plane is averaged. This results in a single three-dimensional vector that can be used for shading.

$$G_x = \frac{S(i+1,j,k) - S(i-1,j,k)}{\Delta x}$$

$$G_y = \frac{S(i,j+1,k) - S(i,j-1,k)}{\Delta y} \quad (1)$$

$$G_z = \frac{S(i,j,k+1) - S(i,j,k-1)}{\Delta z}$$

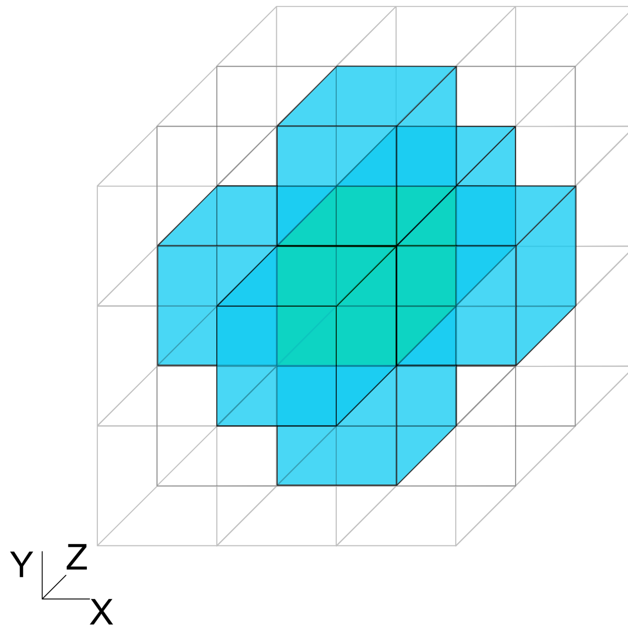


Figure 14: Central difference gradient calculation for the green central voxel. Blue voxels represent the sampled voxels.

المنارة للاستشارات
www.manaraa.com

The computational efficiency of these methods, make them ideal for use in implementations requiring the gradient to be recomputed each frame instead of once at the beginning. The Sobel method is a more accurate gradient calculation than the other two methods because it analyzes 26 neighboring voxels. This method is best applied to implementations where the gradient is only calculated once before rendering because of the relatively large computational expense.

2.3.3 Resampling

Resampling is typically the first step in a continuous render loop. The goal of resampling is to be able to sample the volume data at different positions in three-dimensional space to be used later in the render. The sampling techniques used are one of the primary differences between volume rendering techniques. Texture slicing samples the 3D space along planes that slice through the volume. Raycasting, on the other hand, samples at evenly space intervals along a ray. Figure 15 shows a 2D raycasting example of resampling where the person's eye is on the left looking at the computer screen represented by the black line, as rays are cast into the gray volume. The bounds of the volume data must first be determined by sending imaginary rays, r_i, from each pixel of the framebuffer through the scene, in the viewing direction. Each ray is looking for the front of the volume, the blue boundary, represented by the first intersection with the volume, f_i, and looking for the back of the volume, the green boundary, l_i.

المنارة للاستشارات

www.manaraa.com

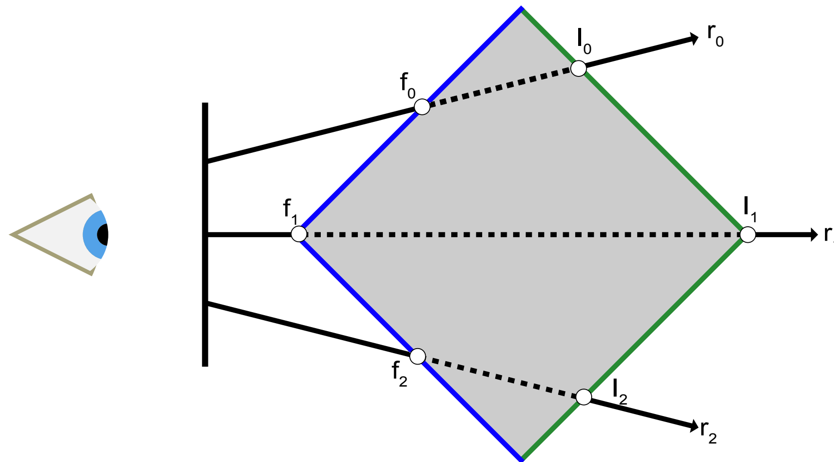


Figure 15: Diagram of raycasting in 2D.

The rays that enter the volume take samples at specified intervals, shown by the dashed lines in Figure 15, along the ray until it exits the volume. The sampling interval can be changed to accommodate different implementation goals. The tradeoff is smaller sampling intervals are more computationally expensive but produce higher quality images. In general, the sampling frequency should be at minimum, half the size of the shortest distance between voxels to meet the Nyquist rate for sampling. The Nyquist rate is the lower bound for alias-free signal sampling and achieved by sampling the signal at twice the rate of the signal [52] and provides the optimal level of sampling. Rays that do not intersect with the volume will render the background image color.

A voxel is commonly represented as a cube in medical imaging where each vertex represents a value. During the step of casting rays and taking samples at specific intervals, it is rare that a ray can sample a vertex directly. More commonly, the sampling interval will require sampling a point within the voxel. Therefore, interpolation methods are needed to accurately determine an approximate value for the sample lying within a voxel. There is a tradeoff between quality and computational efficiency when selecting an interpolation method. The interpolation methods capable of generating the highest

المنارة للاستشارات

www.manaraa.com

quality renders are not real-time because of the significant computational time required to perform the calculations. Nearest Neighbor and Trilinear Interpolation are two of the least computationally expensive methods that are most commonly used with real-time rendering applications. Tricubic and B-spline interpolation are slower but higher quality interpolation methods, most commonly used when accuracy is critical [27,53].

Most real-time implementations of volume rendering use Trilinear Interpolation due to its reasonable visual quality and minimal computational requirements. Commercial graphics processing units all provide hardware-accelerated Trilinear Interpolation, making them very fast. The key to Trilinear Interpolation is the assumption that there is a linear relationship between neighboring voxels. Figure 16 demonstrates the process of linear interpolation for sample point C within a voxel. First, values along the fours axes (i.e. C_{011} to C_{111}, C_{001} to C_{101}, C_{000} to C_{100}, and C_{010} to C_{110}) parallel to the x-axis are used to interpolate values for C_{00}, C_{01}, C_{10}, and C_{11}. Next, those values are interpolated along the z-axis, resulting in C_0 and C_1. Finally, C_0 and C_1 are interpolated along the y-axis to produce the final value of sample C.

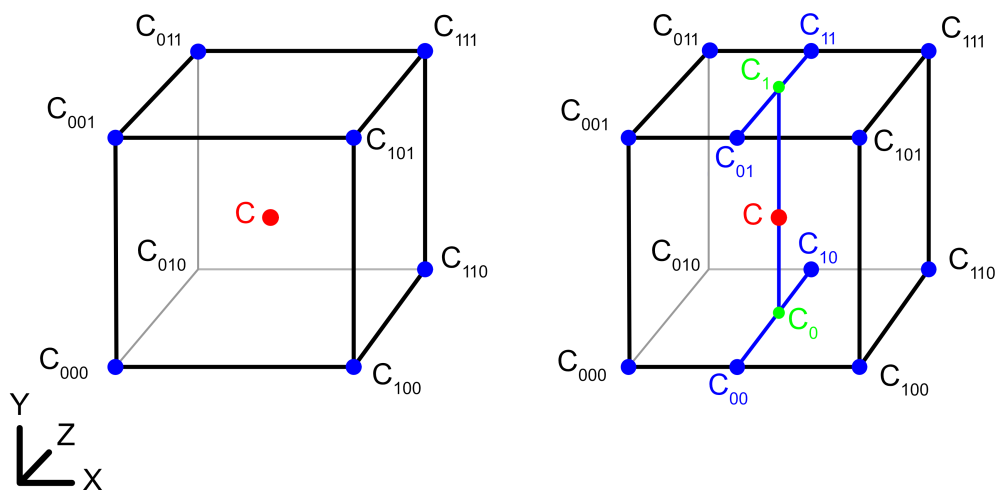


Figure 16: Trilinear Interpolation (http://en.wikipedia.org/wiki/Trilinear_interpolation).

www.manaraa.com

2.3.4 Classification

Classification is the process of determining the subset of interpolated points to make up the final image. This is one of the most powerful tools in volume rendering from a visualization standpoint as it determines what parts of the volume are shown. This is done by mapping the voxel intensity values to opacity values between zero and one. Setting a voxel's opacity to zero prevents it from being viewed in the final image, as it is translucent. Conversely, setting the voxel's opacity to one ensures all of the voxel's data will be shown. A range between zero and one indicates how much of that voxel's data should be included in the final image. By including a percentage of the voxel's opacity in the final image is how volume rendering achieves transparency to see the interior of a volume dataset.

The mapping between voxel intensity and opacity is known as an opacity transfer function [50,54]. Creating an opacity transfer function can be very complex depending on the type of data being viewed. Generating a histogram of the voxel data to visualize the high frequency intensities is typically the first data analysis used to create an opacity transfer function. Opacity transfer functions can be designed to allow specific features of a volume dataset to be easily examined.

2.3.5 Coloring

Coloring is another step in the rendering process that provides immense power in understanding data. The purpose of coloring is not always to provide photo realism, but to provide sufficient contrast to identify desired features. Figure 17 shows two different coloring schemes for the same data with the Muscle and Bone coloring scheme on the left and the NIH scheme on the right. The challenge with coloring is typical volume data assigns a single value (intensity in medical imaging) to each voxel and not a three-

المنارة للاستشارات
www.manaraa.com

dimensional color. Therefore, coloring methods must intelligently convert a single value into three different values in a way that makes the desired structures more visible. Typically, to accomplish this, a different color transfer functions is used for each color channel. So, for a red-green-blue (RGB) color scheme, then three transfer functions would be used one each for red, green, and blue. These three colors are then combined to obtain the final color value for each individual voxel.

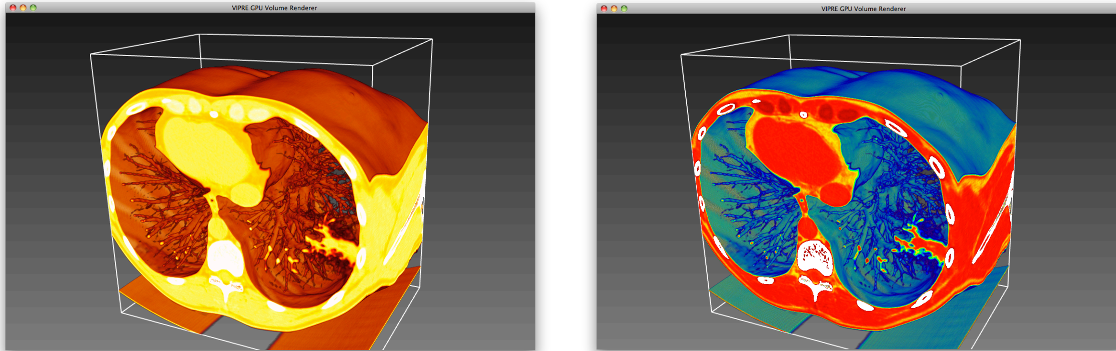


Figure 17: Two different coloring schemes of the same data.

The color transfer functions can be created manually or automatically. Creating color transfer functions manually allows for precise control and the possibility of a better result but can require significant time. Automatically generating transfer functions is less time consuming but all generators are not equally good across different volume rendering uses and data histograms. It is also possible to use a hybrid approach to select transfer functions, where the search for a transfer function is treated as an optimization problem and addressed with stochastic search techniques. The user is kept in the loop by selecting a set of images closest to their desired outcome during each step of the search. Allowing the algorithm to progress toward an optimal set of color transfer functions for the user's application [55].

المنارة للاستشارات

www.manaraa.com

2.3.6 Shading

Shading is the simulation of reflections of light and shadows on a surface. The importance of shading cannot be understated as it provides a better understanding of surface contours and depth in a computer graphics scene. In volume rendering, the shading model is applied per voxel after coloring. Figure 18 shows the importance of shading on a CT of a human skull. The left image shows the skull with standard lighting and no shading algorithm. The right image shows the same skull using a multidirectional occlusion shading model designed for more realistic renders [56]. The addition of shading provides added realism and a sense of depth to the skull that allows better understanding of the structure.

Figure 18: Shading on a human skull CT scan. Left shows no shading. Right image shows the same skull with shading [56]**.**

The first step in shading is the calculation of the gradient of the sampled voxel. Different interpolation methods can be used to calculate the gradient with the most common method being Trilinear Interpolation. The gradient is then combined with the light vector, view direction, and voxel color to compute the final color.

The most popular shading algorithms in volume rendering are the Phong Reflection model [57] and Gouraud Shading model [58] for their computational efficiency. Both methods use ambient light, diffuse reflection, and specular reflection with the light vector, gradient vector, and viewing direction.

المنارة للاستشارات

www.manaraa.com

2.3.7 Compositing

The last stage of the volume-rendering pipeline is to take all the voxel data sampled by an individual ray and composite the information into a single color for that pixel. This compositing process is typically done either in a back-to-front or a front-to-back order for each ray. Front-to-back compositing is more commonly used because of the performance benefits and is the method used in this research.

The front-to-back compositing method in Equation 2 shows the calculation for the final intensity value, $I(x,y)$, for each ray. The final intensity value is a sum of the sample point intensities, I_i, multiplied by all the transparencies (1-α_j) encountered previously along the ray. Put another way, each voxel sample can be thought of as a pane of colored glass that has some opacity, α. If the first pane of glass is completely opaque, the following panes of glass cannot be seen. If the first pane of glass is 75% opaque, the second pane of glass can be seen but its color will only be 25% visible at best. Compositing works the same way, the higher the opacity of a sample, the less the following sample's intensity can contribute to the final intensity value.

$$I(x,y) = \sum_{i=0}^{n} I_i \prod_{j=0}^{i-1} (1 - \alpha_j) \qquad (2)$$

Each voxel sample's intensity value is a combination of the color, C_i, from the color transfer functions and the opacity, $\alpha_{i,}$ from the opacity transfer function. Equation 3 shows how these two values are multiplied together to computer the voxel sample's color. The higher the opacity the more intense the resulting color contribution is.

$$I_i = C_i \times \alpha_i \qquad (3)$$

Front-to-back compositing is continuously evaluating the current voxel sample's intensity and blending it with previous samples. This constant evaluation is what allows

المنارة للاستشارات
www.manaraa.com

front-to-back compositing to achieve the performance benefits mentioned earlier. When the cumulative opacity reaches 1.0, there is no need to continue compositing along the ray because the contributions of all subsequent samples would be zero according to Equation 2. This allows the front-to-back compositing algorithm to perform early ray termination and speed up the render. This is one performance optimization that can be easily implemented and can have significant impact on rendering speeds. More information on volume rendering compositing can be found in [59–62].

2.4 Advanced Raycasting Techniques

Many advances have been made in volume rendering beyond the basic rendering techniques mentioned above. Some involve improving visuals and some involve speed optimizations. The speed optimizations will be discussed in Chapter 3 because they are very important to visualizing the large amounts of data involved in functional imaging. This section will focus on the advanced volume rendering techniques of lighting and clipping.

2.4.1 Lighting and Shadows

One of the most important visual strategies in any rendering type is the use of lighting and shadows due to its contribution in spatial comprehension through monocular depth cues [63,64]. With the goal of volume rendering being to interpret volumetric data, realistic lighting and shadows are crucial for viewers to understand special relationships.

A simplified direction illumination model is often used in volume rendering instead of a global illumination model because of the computational complexity inherent in global illumination models. Phong Illumination [57] is a simplified illumination model and is the most widely used due to its computationally efficiency, as stated previously in the Shading section. This simplified illumination model does not consider the impacts of

المنارة للاستشارات

www.manaraa.com

other objects in the scene on lighting, but instead the light coming directly from its source. Phong illumination requires inputs of voxel position, voxel gradient, voxel color, and the light source's position. Diffuse, specular, and ambient lighting are then applied to the voxel to obtain the final pixel color [65].

Illumination helps improve the visual quality and give some understanding of depth but it is shadows that really provide improve depth understanding. In computer graphics, shadows have been extensively studied in raytracing algorithms for improved visual realism [66–68]. However, not a lot of research has been done on shadows in volume raycasting implementations [65]. Raytracing is capable of accounting for light interactions from many objects within the scene, often requiring significantly more computational overhead. This allows raytracing to simulate a wide variety of visual effects, such as refraction of light through glass and reflections off surfaces. The additional computational overhead required achieving these high-fidelity effects make raytracing shadow models unattractive for interactive volume rendering applications.

The computational overhead associated with raytracing methods for shadowing required a new set of methods specifically designed for raycasting. The first volume raycasting specific method for shadow mapping was in 1978 [69]. Shadow mapping adds an additional render pass, which determines which voxels are closest to the light source. Then in the main rendering pass, each voxel is tested to determine whether a shadow should be applied to it. Shadow mapping efficiently calculates shadows on a per-fragment basis, but is not capable of semi-transparent shadows. It can also provide soft shadows by using percentage closer filtering [70].

Opacity shadow maps [71] and deep shadow maps [72,73] are two techniques that built upon shadow mapping to achieve semi-transparent shadows. Of these methods, deep shadow maps are more often used for high quality rendering but at higher

المنارة للاستشارات
www.manaraa.com

computational cost. This technique uses a stack of textures to store both the depth and opacity information for various layers of the shadow map. This process can generate visual artifacts around very thin or complex areas, which can only be eliminated through the generation of additional shadow layers at additional computational cost.

Ambient Occlusion is a different illumination technique that calculates the visibility of a light from a voxel. For computational efficiency, Vicinity Shading [74] pre-computes the occlusion for each voxel in the scene and stores them in a 3D texture for reference during the rendering loop. The computational efficiency of Vicinity Shading can be further improved by combining ambient occlusion volumes into a composite occlusion volume [75].

Local Ambient Occlusion is slightly different from traditional Ambient Occlusion as it is based on casting rays in multiple directions from each voxel for a specified radius. The level of occlusion is based on how many non-transparent voxels each ray intersects [66,76]. Figure 19 shows the difference between traditional ambient occlusion and local ambient occlusion [77].

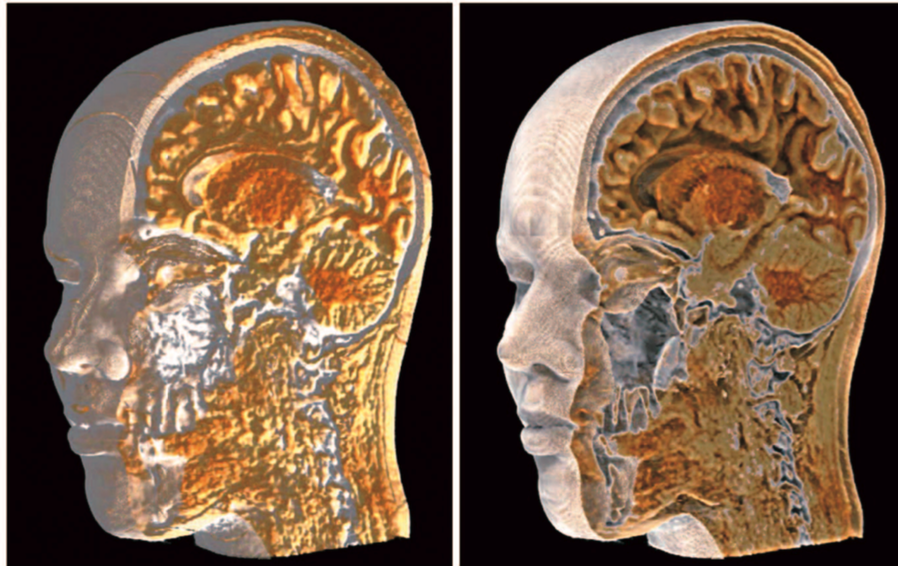


Figure 19: Diffuse illumination (left) versus local ambient occlusion (right) [77].

المنارة للاستشارات

www.manaraa.com

2.4.2 Clipping

Unlike surface models in traditional computer graphics, volumetric data contains internal structures and data. Clipping is a visualization method that allows the exploration of this internal data. It is important enough that almost all volume renderers implement some form of clipping. Clipping planes are by far the most basic and easiest to implement [78,79]. A clipping plane is a geometric plane that acts as a threshold for the volumetric data. All of the voxels on one side of the plane will be transparent and the other side will be visible. This allows parts of the volume to be clipped away to show the internals.

More advanced clipping techniques have been created to allow more complex clipping shapes and provide better understanding of the data. Hinged clipping planes were created to provide better understanding of spatial relationships in the data [80]. Volume sculpting was proposed as a way to generate complex clipping geometry for exploring a volume [81]. Depth-based clipping allows complex geometric shapes to clip the volume [82]. The clipping plane itself can be deformable, allowing for complex clipped shapes [83]. Binary-clip volumes can also be used with depth-based clipping methods for more control over the clipping geometry [34, 80, 82, 83].

A different paradigm for clipping is to use exploded views to see the internals instead of removing external geometry. By moving the external data away from the rest of the data, the internals can be seen in relation to the entire volume. Figure 20 shows an example of a human head being examined by exploded view, where each view going from left-to-right increases the amount the external structure is moved to reveal the internals.

The first uses of exploded views in medical imaging progressively built upon each other from simple spatial transforms of the volume [86] to slicing the volume up into

المنارة للاستشارات
www.manaraa.com

many different parts [87] and then using complex transformations to spread apart and peel away sections of the data [88]. The next big goal with exploded views was to automatically generate these views so the point of interest was not occluded as the volume was interacted with. This was done through modified compositing strategy [89] and intelligent spatial transformations [90].

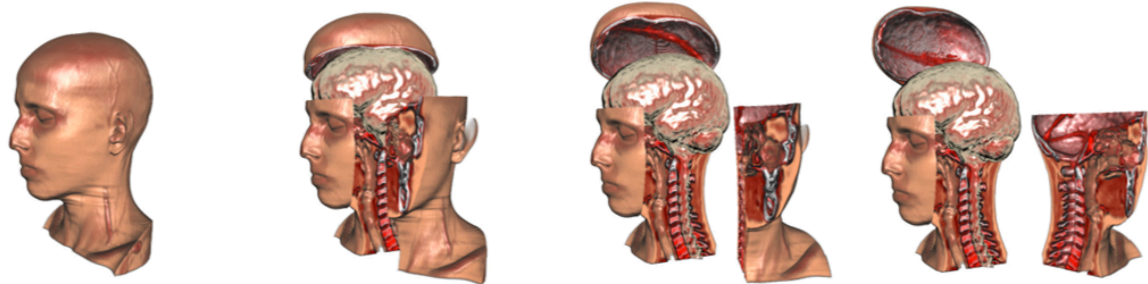


Figure 20: Exploded view of a human head. [90]

المنارة للاستشارات

www.manaraa.com

CHAPTER 3: 4D VOLUME RENDERING

Rendering medical data in 4D presents its own sets of problems and limitations. All of the issues relevant to 3D volume rendering are present in 4D rendering, with added challenges of much more data and rendering multiple volumes simultaneously. There are also new visualization strategies that must be created to reveal important information in the data.

In 3D volume rendering, there is one set of volumetric data for the entire scene. In 4D rendering, there is one set of volumetric data for each sample time causing the amount of data to increase linearly with the number of samples taken. Thus, methods are required to handle the increase in data and allow rendering at interactive speeds.

3.1 Rendering Speed Optimizations

In 3D volume rendering, there is one set of volumetric data for the entire scene. In 4D rendering, there is one set of volumetric data for each sample time causing the amount of data to increase linearly with the number of samples taken. Thus, methods are required to handle the increase in data and allow rendering at interactive speeds.

Two of the most common methods for improving rendering speed with raycasting are early ray termination (also known as adaptive termination) and empty space skipping [91]. The goal of both is to reduce the number of resampling and compositing operations for a ray as it traverses the scene.

Early ray termination was originally proposed by Whitted [92] as an adaptively terminating raytracing algorithm. Levoy [93] took this idea and developed two front-to-back volume raycasting algorithms. The first was a case where the ray traversal should be terminated when an opaque voxel is reached. The second case terminates the ray traversal when the accumulated opacity reaches a user-specified level where additional

المنارة للاستشارات
www.manaraa.com

samples will not dramatically change the final pixel color. In practice, this user-defined value is generally between 0.1 and 0.01.

Empty space skipping is well researched in raycasting. The main idea behind empty space skipping is that most volumetric datasets contain large contiguous regions of voxels with no value, often represented as zero opacity. These large contiguous empty areas are very common in medical imaging where half of the volume space may be representing air. To prevent computational cycles from being spent compositing samples with zero opacity together, the voxel data can be stored in an organized data structure to quickly search for areas with relevant voxel data and to skip the empty areas.

3.1.1 Octrees

Octrees are one of the most common data structures used for empty space skipping. An octree is a tree data structure that recursively subdivides a 3D space into eight parts, which can be evenly spaced or not. An example of octrees and their subdivision strategy can be seen in Figure 21. Each node in the octree contains a value or range of values representing the accumulation of all the nodes' values below it in the tree. This allows fast traversal of the data by starting at the top-level node and traversing downward in the tree structure until the desired value is found, stopping the traversal.

In terms of volumetric data, the lowest level of the tree, the leaf nodes, are the individual voxels of the dataset. All the voxels are grouped with their eight neighbors to form the node one level above them, as shown in Figure 21. This continues until the top-level node, the root node is reached resulting in a single node. The value stored at each node is a binary value representing whether the region contains all empty voxels. Storing

المنارة للاستشارات
www.manaraa.com

the data this way allows the octree to be traversed from top until the first node is reached indicating all subsequent nodes are empty and stopping the traversal.

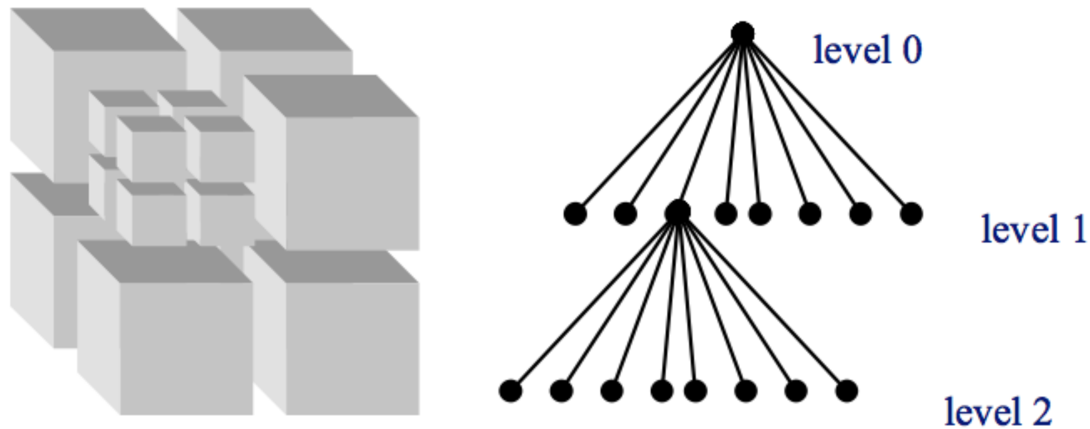


Figure 21: Octree data structure example

The first octree used in volume rendering was created by Meagher to create a condensed representation of the volume, which was traversed in a depth-first manner [94]. Levoy built upon this work by representing the entire volume as a complete octree and rendering in image order by tracing the rays from the observer's position through the octree [93]. This is the method many current techniques are built upon in volume rendering. While the use of octrees for empty space skipping is very common, there are other data structure implementations that have been proposed [91,95–101].

The texture memory on modern GPUs is still a bottleneck for large volumetric datasets, therefore, compression techniques have been investigated to reduce the amount of text data required to describe a volume. Branch-on-need Octrees (BONO) built upon the work done with the original octree to provide a more space efficient structure for octrees that are not power of two. Subdividing the volume data looks to create power of two subdivisions in the first level below the root node. This technique reduces the number of nodes in the tree resulting in a faster traversal and a smaller data

المنارة للاستشارات
www.manaraa.com

structure [102]. BONOs were then extended to hold 4D data by storing each time step in a separate tree, resulting in a Temporal Branch-on-need octree (T-BON) [103]. The T-BON structure can be compressed even further by looking for data that does not change from one time step to the next. When this occurs, a pointer can be used in the node to point back to the previous time step's node instead of maintaining an exact copy. This simple trick reduces the size of the T-BON by the duplicate node data as well as all the children nodes [104].

To speed up rendering, octrees can be used for more than early ray termination. Additional research used octrees to improve rendering speeds by storing aggregate values at each node to represent the values of the child nodes. In this way, a suitable "error" could be set to allow the octree traversal to stop before reaching the leaf nodes if the aggregate value of the node was less than the "error." This method is similar to empty space skipping in that large homogeneous areas of the volume could be assumed to be of similar value and eliminated all at once. Boada looked at using the variance in the data at each node to define a "cut" line across the octree that stopped the tree traversal [105]. Plate instead determined when to stop the tree traversal based on the GPU texture size available and the desired level of detail [106]. Similarly, the octree can improve rendering speeds through the use of different levels of detail, where lower nodes in the tree represent higher levels of detail [107].

3.1.2 KD-Trees

KD-Trees are a multidimensional binary search tree conceived in 1975 by Bentley [108]. This tree data structure works very similar to octrees, except that the multidimensional volume is divided into two sections at each node and not eight as in octrees. Empty space skipping is the primary use of kd-trees in volume rendering, similar

المنارة للاستشارات
www.manaraa.com

to octrees. While the traversal of kd-trees is very fast, they are time consuming to create and are thus typically used for static scenes. There has been some work to overcome this limitation by creating an iterative approach to generating the tree, versus a recursive build [109]. Still, kd-trees are not heavily used in medical volume rendering at this time.

3.1.3 Time Space Partitioning Trees (TSP)

Time Space Partitioning (TSP) trees are an extension of octrees to handle 4D volumetric data conceived by Shen in 1999 [110]. The base of this structure is a BONO, but it then integrates the fourth dimension by adding a binary search tree at each node to represent the different time steps. This means that when searching for voxel data, the search must first traverse an octree to find the desired node and then traverse the binary tree to find the step in time. Each node also stores the error tolerance to allow early traversal termination once the data variance error is below the user specified threshold.

The error tolerance value is traditionally calculated using the intensity value from the volume data. Ellsworth proposed evaluating this error tolerance using the RGB value obtained through the color transfer function [111]. This change was shown to improve rendering speeds by eliminating unnecessary tree traversals.

3.2 Data Compression Techniques

Texture memory on a GPU is one of the largest bottlenecks in volume rendering. If all of the volumetric data cannot be stored in the GPU's texture memory, that data must be transferred back and forth between the hard drive and the GPU. This transfer process is often the reason large volumes cannot be rendered interactively. One way to overcome this limitation is to compress the data in an attempt to store more data in a smaller amount of texture memory.

المنارة للاستشارات
www.manaraa.com

3.2.1 Run Length Encoding (RLE)

Run Length Encoding (RLE) is possibly the simplest compression methods to implement in computing. The method looks for repeating numbers in a sequence, such as empty space in a volume, and stores two numbers, the value itself and the number of times it is repeated in a row. This allows large homogeneous sections of data to be stored as two values. RLE does not support the highest compression ratios, so it is not often used alone. However, multiple compression implementations will use RLE along with another compression method. Neophytou is one of the few methods proposed that used RLE as the primary compression method [112].

3.2.2 Discrete Cosine Transform (DCT)

Discrete Cosine Transform compression is a method of storing a finite set of data points as a sum of different frequency cosine functions. This method of compression was popularly used for compressing image files as the JPEG standard [113,114]. Medical imaging data is a stack of images and thus this technique would appear to be a perfect match. Lum changed the DCT compression from compressing slices to compressing voxels across time [115]. This change produced dramatic compression rates in datasets with high temporal coherence.

3.2.3 Wavelets

Wavelet compression techniques have taken over for DCT in recent years with newer image compression standards like JPEG 2000 using wavelets. There are dozens of different wavelets that can be used in signal processing. Volume rendering compression algorithms typically use the discrete wavelet methods such as Daubechies [116] or Cohen, Daubechies, Feauveau (CDF) [117]. Guthe used wavelets and run length encoding of empty space values to compress 4D volumetric data and decode it

المنارة للاستشارات
www.manaraa.com

on the fly [118]. To alleviate the problem of decompressing each time step, they stored the wavelet coefficients in a hierarchical tree structure, allowing the decompression of only the necessary coefficients at a specific point in time [119]. Other research has achieved similar compression results using different wavelets [120,121].

3.2.4 Index and Data Maps

Another popular compression method with volumetric data is to use two maps to represent the total volume. The first map is an index that samples the volumetric data at regular intervals. The index map points to a position in the data map that holds the actual data. The data map grows as needed to represent all the different types of data. For homogeneous data sets, the sampling window in the index map can be very large, meaning a single index value can represent many voxels. The data map can also be very small because a few chunks can represent all of the different types of data.

Kraus was one of the first to utilize this method in 2002 [122]. They used a fragment shader to decompress the data in real time and were able to achieve large compression ratios for large sampling windows. Binotto used the same compression technique and fragment shader decompression technique a year later with a different implementation [123]. Schneider used vector quantization to compress each slice of data and then used an index and a code book to further compress the textures. Their method used this index and data mapping to attempt to compress the data across the time domain after the vector quantization was used for compression within the time domain [124].

3.3 Multi-Volume Rendering

Four-dimensional volumetric datasets significantly increase the amount of voxel data to be rendered. One strategy for optimizing rendering speeds is to decompose the

المنارة للاستشارات

www.manaraa.com

original data into two individual datasets. The first is a high-resolution scan holding all the structural data that does not change over time. For example, in an fMRI of the brain, this would equate to the structure of the brain itself. The second scan is a lower resolution time-varying scan. In the previous example, this would be the brain activity. This reduces the amount of volumetric data that changes from one frame to the next, allowing for faster rendering because large amounts of voxel data is not being transferred continually.

However, this introduces the challenge of rendering multiple volumes at the same time. The two main challenges are mixing voxel data from multiple volumes into a single render and depth sorting the multiple volumes. Both of these challenges impact the final image representation and thus the interpretation of the data. Adding to this challenge is the fact that sometimes non-volumetric data must be rendered in the scene, such as a trocar for planning a minimally invasive surgery.

Mixing data from multiple volumes into a single coherent image is a complicated problem that has been fairly well defined over the last 20 years. Multiple volume rendering uses the same basic volume-rendering pipeline described in Section 2.2, where volumes are repeatedly sampled, colored, and composited together over multiple points to achieve a single pixel value. The difference in multiple volume rendering is that there is one sample for each volume at each point in space. There are two different methods of combining those samples together. The first is One Property per Point (OPP), where a single volume's sample is selected to be the final value at that point. The second option is Multiple Properties per Point (MPP), where all samples are combined together to achieve a single value for that point [125,126].

There are multiple ways to combine the sample values together in MPP. The most common method is to use a weighted function describing how much each volume

المنارة للاستشارات
www.manaraa.com

contributes to the final value. The biggest difference is typically where in the volume rendering pipeline this occurs. Data mixing can occur in the sampling, coloring, or image stages. When data mixing is performed at the sampling stage, all volumes are sampled at the same point for an intensity value. These intensity values are then combined to obtain a single intensity value for all volumes. The resulting intensity value can then be passed into the color and opacity functions to obtain a final color. When data mixing is performed at the coloring stage, each volume is sampled for an intensity value. The intensity values are then passed to the color transfer function to obtain a color for each volume sample. The colors themselves are then combined to obtain a final color for that sample. When data mixing is performed at the image stage, each volume is rendered separately before combining the individual frame buffers for a final image. Many multivolume renderers use different combinations of these mixing methods to achieve the highest quality visual for their specific application.

Depth sorting of volumes is the other challenge of multiple volume rendering. Without properly sorting depth, the resulting images would not provide an accurate representation of the data. This could potentially be a serious problem if the volume renderer was, for example, being used for surgical planning and the trocar was visualized in front of the heart when it was actually being placed behind. Each rendering method has unique ways of handling depth sorting that will be described in the following subsections.

3.3.1 Multi-volume Texture Slicing

Texture Slicing is one of the easier rendering methods to perform accurate depth sorting because the technique inherently creates slices of data moving away from the viewing plane. Figure 22 shows an example of two sets of volumetric data using texture

المنارة للاستشارات
www.manaraa.com

slicing. The white slices are one volume and the gray slices are another. The image on the right show the slices stacked together in depth sorted order. The only potential issue with depth sorting texture slices is the potential for slices to have the exact same depth and appear at the exact same location.

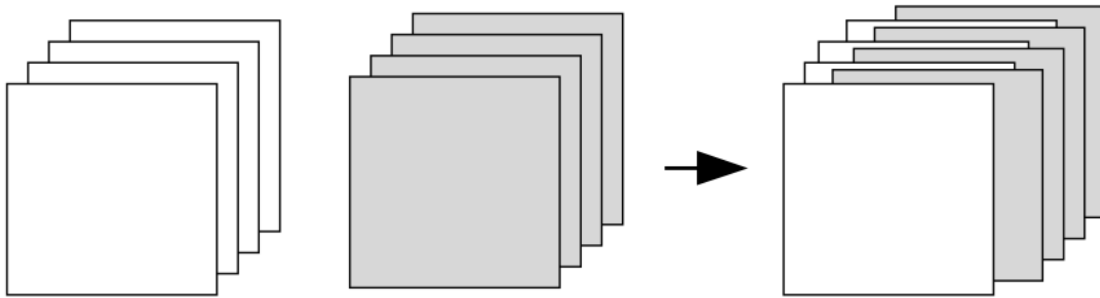


Figure 22: Texture slicing depth sorting example

This method of depth sorting texture slices has been used effectively since the 1990s. VIVIAN was one of the first volume rendering tools that incorporated depth sorting for accurate multivolume rendering on traditional workstations as well as virtual reality hardware [127]. Patel, et al., built a similar system for a virtual reality environment using depth sorted texture slices. Their approach focused on creating useful transfer functions for providing the best visualizations of tumors [128].

Robler, et al., built a two-step renderer that first sliced all the volumes in the scene parallel to the viewing plane before sorting them by depth. In the second step, they combined the slices in a back-to-front order, calling the shader specific to that slice. This allowed them to visualize individual volumes in a specific way while still rendering the entire scene [129]. This method was expanded a year later by mixing direct volume rendering for one volume and iso-surface rendering for the other. The same two step rendering process was performed with the iso-surface also being view-align sliced [130].

المنارة للاستشارات

www.manaraa.com

Wilson, et al., addressed this challenge when they experimented with three different methods of data mixing: 1) chose one volume's value as the only value, 2) use a weighted function to combine all the values, or 3) use a single volume value each for the red, green, and blue color channels. The results of this research showed uses for each of the three mixing methods [131].

One of the first researchers to use raycasting to render multiple objects in a scene was Levoy [132]. Specifically, it was to render polygonal models with volumetric data. This simplified the problem of raycasting multiple objects because depth sorting polygonal data was already well defined.

Kreeger and Kaufman created a rendering method for both a volume and polygonal geometry. When rendering polygonal geometry with volumes, it is typical to slice the geometry into sub-geometries based on the view-aligned slices. This gives a natural depth ordering of objects. Kreeger and Kaufman instead create bins between each slice for the associated geometry. This reduces the number of triangles being sent to the graphics card [133].

3.3.2 Depth Peeling

Unlike texture slicing, it is not as inherently easy to render multiple volumes using raycasting because there is no clearly defined depth order. Depth Peeling is the most commonly used algorithm for depth sorting volumetric data and is critical to raycasting multiple volumes correctly. Depth Peeling was initially conceived to help correctly render 3D scenes with transparency. It requires rendering the same scene multiple times, where each time through, another layer of depth is removed from the scene until the entire scene is described in layers. These layers are then composited together in the

المنارة للاستشارات
www.manaraa.com

correct depth order to show accurate transparency [134]. This allows Depth Peeling to be used for correct sorting of volumetric data.

Plate, et al., used Depth Peeling with their “lenslets” solution for multiple volume rendering. The “lenslets” are created by dividing the volumetric data into geometry containing either one volume or multiple volumes and then depth sorted using Depth Peeling [135]. Dividing the scene up into single or multiple volume geometry allowed single “lenslets” to be rendered with faster traditional raycasting approaches.

Brecheisen, et al., presented another interesting use of Depth Peeling by alternating it with raycasting. Depth Peeling was used to define one layer of depth in the scene that was then raycast. This process was continued until the entire scene was traversed. This is one of the first methods to successfully visualize multiple volumes with transparent geometry and concave clipping objects [85].

Kainz, et al., used the GPU programming language CUDA to create a multi-volume and polygon rendering method using Depth Peeling. The method projected all the polygon triangles into screen space and then used Depth Peeling to depth sort the scene before raycasting [136].

3.3.3 Multi-volume Raycasting

Future researchers approached this problem using the assumption that the multiple volumes were correctly aligned in 3D space and never moved in relation to each other [137–139]. This assumption was built upon the use of multimodal data in medical imaging. For example, an MRI scan may be performed to gather anatomical data and an electroencephalogram (EEG) might be performed to gather brain activity. These two modes of data could be combined to show where exactly on the patient’s brain the activity is happening. Aligning the two datasets is critical to accuracy and can be

المنارة للاستشارات
www.manaraa.com

performed using anatomical or other physical markers in the data. Once the volumes are aligned in 3D space, the ray cast into the volume samples once from each volume at every sampling point along the ray. The problem then became mixing the values from each volume to achieve the best result.

Cai, et al., looked at mixing the volume data in three different ways [137]. The first looked to mix the intensity values from each sample. The second looked to mix the color values from the color transfer functions at each sample. The third method looked to render each volume separately and then combine the resulting images together to form a single image. Manssour, et al., proposed a volume rendering framework that used the second mixing method proposed by Cai, et al., where the two volume values were combined after the color was determined for each sample [138]. The color combination was achieved through the use of a weighting function. Beyer, et al., used the same approach for data mixing but also allowed a single volume's value to be the final sample color instead of mixing the two values [139]. The ability to use only a single volume's value was found to be useful when certain characteristics were more important than others (e.g., EEG brain activity would be more important to visualize than the skull).

All of the previous examples only looked at mixing the color for each sample. The obvious approach is to combine the color channels of the volumes into a single image. It is possible to visualize data using the opacity channel as well. Manssour, et al., mixed two volume datasets by using one volume for opacity and the other volume for color [140]. Wilson, et al., while a texture slicing method, rendered three volumes by using one volume for each of the color channels (e.g., one volume for red, one volume for green, and one volume for blue [131]). While interesting approaches, encoding too much information into a single image can confuse users and hide more data than it reveals if not used carefully.

المنارة للاستشارات

www.manaraa.com

Depth Peeling may be the most popular, but it is not the only method for depth sorting a scene for proper rendering. Grimm, et al., proposed a bricked memory method of storing multiple volumes in a depth sorted way [141]. This method attempted to speed up the raycasting process by breaking up the scene into subsections that held intersecting volumes and subsections that held a single volume. This allowed a faster single-volume renderer to be used when possible and a slower multiple-volume renderer to be used when required.

3.3.4 Scene Graph Methods

Multi-volume rendering has started to face issues with managing and manipulating each of the volumes within the scene. Computer graphics faced a similar problem when trying to render multiple 3D polygons in a scene. It becomes complex to manage and manipulate specific objects within a scene as the number of objects increase. To address this problem, scene graphs were created to manage the many spatial representations of a 3D scene.

A scene graph is a collection of 3D objects, transformations, and graphical elements called nodes. Each node, other than the root node, can have multiple parents and/or children associated with it. This allowed complex geometry to be grouped together by creating a group parent node for all the geometry child nodes. Any manipulation applied to the group node (parent) would impact all the geometry nodes (children). This structure allowed for a clearly defined hierarchy for managing large scenes and for transformations (i.e. rotations, translations, or scaling) to be applied quickly and efficiently.

المنارة للاستشارات

www.manaraa.com

A few researchers have taken this idea of a scene graph and tried applying it to volumetric scenes. If properly implemented, this could potentially take care of all the depth sorting issues and clarify the process of combining volume data together.

Nadeau attempted the first volumetric scene graph in 2000 [142]. The goal was to create a clear graphical way to represent how volume data was being mixed together. Each volume was represented by a leaf node in the scene graph and all mixing operators were group nodes. This graphically showed which volume datasets were being combined using a specific weight function for mixing data. One significant issue with this method is that the entire scene needed to be voxelized before rendering. This required finding a bounding volume for all datasets and decomposing that bounding volume into voxels for rendering. Therefore, voxels comprising each dataset must be converted into scene voxels for rendering, and the mapping is usually not a one to one translation. This process is computationally expensive and must be performed when the scene changes.

Rößler, et al., proposed a method of rendering where a multi-volume scene is broken up into sub-volumes based on where the volumes overlap [143]. A scene graph is used to represent all of these sub-volumes and the information needed to render them, similar to Nadeau. Breaking up the scene into single and multiple volume sections allows optimizations to be made in rendering by using a single volume renderer when possible and a multiple volume renderer only when necessary. This approach works well when the scene does not change frequently. Every time the scene changes, the process of subdividing the volumes into single and multiple must be performed. The scene graph structure also allows unique shaders to be generated for the sub-volumes based on the rendering characteristics required. Therefore, every volume can be rendered differently and the scene graph will manage this.

المنارة للاستشارات

www.manaraa.com

3.3.5 Other Multi-volume Methods

The previous methods all approached the problem of rendering multiple volumes with the assumption that volumes would intersect each other. When that is not the case, a more straight forward volume rendering approach can be used.

Leu and Chen created a tool named TROVE built upon this assumption [144]. In an effort to reduce the memory footprint required for volume rendering, TROVE would only render a single volume at a time. By depth sorting the volumes based on their bounding boxes, TROVE could start rendering each volume in back-to-front order into the final image.

Bruckner and Gröller used a similar approach to create their exploded view volume renderer [90]. Their tool broke up a single volume into multiple sub-volumes and then arranged the sub-volumes in the scene so no sub-volume would occlude the others. For example, in the case of a human head, the skull might be broken up into three sub-volumes and the brain might be one sub-volume. The sub-volumes of the skull could be moved away from the brain to allow someone to view the brain unobstructed. Adjusting the sub-volumes in this way assured that no sub-volume would intersect with another and simplified the rendering process by allowing the sub-volumes to be sorted along a ray and composited to a final image.

3.4 Methods to Improve Understanding of 4D Data

"The ability to extract objective and quantitatively accurate information from 3D biomedical images has struggled to keep pace with the ability to produce the images themselves." –Robb [145]

The rendering methods behind volume rendering are complex and fascinating, but its purpose of conveying useful information must never be forgotten. The ability to

المنارة للاستشارات
www.manaraa.com

capture volumetric data is useless without techniques for effectively exploring the data to discover importance. The use of volume rendering for medical training [146] as well as diagnoses and treatment will only grow in the future. Effective visualization techniques are what allow medical professionals to make life-changing decisions for patients.

The most common volume visualization technique defines color and opacity through the use of transfer functions. The color and opacity transfer functions map a single intensity value to a set of red, green, blue, and alpha (RGBA) values. The purpose of coloring is not to provide photo realistic images, but to provide sufficient contrast to identify features of interest. The use of coloring was discussed in Section 2.3.5 and its implementation can be seen in almost every volume rendering reference in this paper.

Robb provided an excellent analysis of various visualization methods for volumetric data [145]. Volumetric visualization has grown since Robb's analysis in 1999, especially for visualizing 4D data. These new methods can be broken into three different categories, Region of Interest (ROI) visualizations, visualizing 4D data in three dimensions, and 4D animation visualization.

3.4.1 Region of Interest (ROI)

A Region of Interest is a subset of a complete dataset that is identified for a specific purpose. In the case of medical imaging, the ROI is often a tumor or other ailment requiring the attention of a medical professional. In 2D data, attention is often brought to the ROI by highlighting or circling. In 3D and 4D data, this becomes more complex, especially with a dense block of data as in the case of volume data.

VROOM was one of the first rendering tools to look at visualizing an ROI differently than the rest of the volume. The ROI was rendered at a higher visual quality than the remainder of the volume. This provides a clearer representation of the ROI and a fuzzy

www.manaraa.com

representation of the remaining volume [147]. Zhang, et al., used a similar approach with raycasting, where the sampling size decreased to provide a higher resolution image for the ROI and increased for the rest of the volume [148].

Hauser, et al., used Maximum Intensity Projection (MIP) to render the ROI and Direct Volume Rendering to render the rest of the scene [149]. Traditionally, MIP is designed to make the maximum intensity values stand out. In this case it was used to make an ROI stand out against the rest of the volumetric data. This approach was tested on both 3D and 4D animated data with relative success. This research was later extended using the "Focus-Plus-Context" visualization method, which makes non-important areas of data transparent to provide context while highlighting the important parts. In this case, transparency was applied to the non-ROI areas of the volume while the ROI was emphasized as non-transparent [150].

To study cardiac motion, Enders, et al., assumed that the ROI was anything that moved in the scene [151]. They proposed a method of visualization that clipped out any data that did not meet a specific threshold for movement. This allowed them to visualize a heart beating without distraction from any of the surrounding tissue.

In a slightly different approach, Bruckner and Gröller emphasized the ROI by generating an exploded view instead of clipping the surrounding tissue [90]. The ROI was revealed by subdividing the volume into sections that either were the ROI or were not. The non-ROI sections were moved away to reveal the ROI but not removed to provide spatial context.

3.4.2 Compressing 4D Volume Data for 3D Visualization

It has become more common for volumetric data to include a time component making it 4D. This trend has only accelerated in recent years making visualizing 4D

المنارة للاستشارات
www.manaraa.com

volumetric data increasingly important. The first attempts to visualize 4D volumetric data relied on techniques for compressing the data into single 3D “timesteps”. The first compression techniques like contour lines [152] were borrowed from other areas of data visualization. The 4D compression techniques have become more complex with the use of hyper-slicing and advanced compositing methods. Hyper-slicing works on the principle of projecting data from a higher dimension to a lower one. This technique projects a 3D volume onto a 2D image plane for viewing. With 4D data, a 3D hyper-plane is defined and the 4D data is projected to this plane. A 3D hyper-plane is essentially a 3D volume that can be rendered using traditional volume rendering techniques.

Neophytou and Mueller presented Body Centered Cubic (BCC) sampling, a change to the traditional Cartesian sampling of a volume space, to compress the memory footprint [112]. This method of hyper-slicing no longer provided a full 4D view; requiring motion blurs to be applied to visualize changes in time.

Woodring, et al., also used hyper-slicing to visualize 4D data as a 3D object [153]. They combined the hyper-slicing with color interpolation to better visualize how the data changed over time. The oldest time step was colored blue, the most recent was colored red, and the medial was green. All other time steps were interpolated between those three colors [154]. This work was integrated into a visualization tool using boolean operators to combine multiple time steps into a single 3D scene. A user interface allowed boolean operators to be combined to create complex volume results. These were then visualized using color and opacity interpolation methods to show changes over time [155].

Other compression techniques involve using flow lines to show how fluid moves over time. Helgeland and Elboth used an anisotropic diffusion method, which smeared

المنارة للاستشارات

www.manaraa.com

an input-texture along a vector field to produce flow lines [8]. Other techniques use silhouettes, transparency, "speedlines," or glyphs to represent changes in properties over time [156].

3.4.3 4D Animations

Another approach to visualizing 4D data is to use animations to represent the time component of the data. While animation is logical to visualize time varying data, it is not always the most efficient given the amount of time required to watch the animations or effective given the difficulty in comprehending trends between multiple timesteps. Therefore, animations must be used judiciously for the best result.

The first implementations animating volumetric data fought with rendering speeds and used isosurface rendering and compression algorithms to achieve acceptable frame rates [157]. Sonograms were a popular volumetric technology to take advantage of 4D volume rendering [158,159]. The research showed animating the volumetric data had a significant benefit for medical applications like identifying congenital heart defects [158]. Later research used GPU shaders to animate 4D CTs paired with electrocardiograms (ECGs) to visualize cardiac processes with similar benefits [148].

Tory, et al., combined animation with color interpolation and glyphs techniques in an attempt to create more useful visualizations [160]. Figure 23 shows the direct volume rendering of the anatomical kidney data combined with changes in time represented by color. Figure 24 shows the use of glyphs to represent flow data overlaid on the same kidney data. Surprisingly, this research found that color and glyphs were only useful for a specific type of changes in the data. Color was useful for representing small, localized changes in the data, but was not ideal for large, global changes. Glyphs on the other

المنارة للاستشارات
www.manaraa.com

hand, were not effective for small, local changes because the changes were too small to be seen.

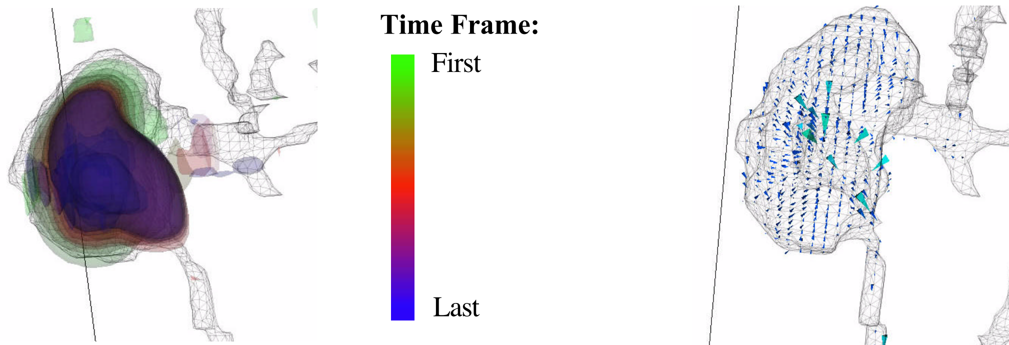


Figure 23: Kidney CT scan with time varying data represented by color [160]

Figure 24: Kidney CT scan with time varying flow data represented by glyphs [160]

The visualization methods presented show many different approaches to visualizing volumetric data with no one best solution. Johnson looked at multiple case studies using visualization for medical data in 2012 and concluded that visualization capabilities lag far behind the ability to produce data [161]. This shows there is still a need for good visualization tools and new visualization methods.

3.5 Current Volume Rendering Tools

The potential for volume rendering has not gone unnoticed in the academic or commercial markets. There have been multiple volume renderers created both academically and for the commercial market, with most focused on medical imaging applications. Typically, advanced volume rendering techniques are designed in academic arenas where the implementation is a one-off solution. Unfortunately, these one-off solutions rarely make their way into commercial tools where medical professionals can make use of them. The tools are rarely built with end developers in mind, so the interfaces are difficult to work with and extend to other applications. When commercial companies do implement an advanced rendering method in their tool, it is

المنارة للاستشارات
www.manaraa.com

generally locked down to extending functionality and medical professionals are charged large sums to use them.

Volume rendering tools are generally broken down into two types, static or dynamic, based on the type of data they visualize. Static volume renderers are typically used to visualize anatomical data in medical applications. Static volume data is comprised of an *x,y,z* position and a property value, *v*. Dynamic volume renderers are typically used to visualize time-varying data like flow or brain activity. Dynamic volume data adds an additional time, *t*, component to the static volume data. The renderers can also be broken down as open source or closed source implementations.

3.5.1 Static Data Volume Renderers

Static data renderers visualize a single high-resolution scan for anatomical purposes. This is the most common type of volume renderer as the number of static scans performed currently greatly outnumber dynamic scans.

One commercially available and open source renderer is Slicer. Slicer is a cross-platform (BSD-style license) renderer working on Mac, Linux, and Windows. Slicer provides extensibility through the use of plugins. It is an open source project that is updated approximately once a year.

There are many more commercially available, closed source, volume renderers today than there were even five years ago. Some of the more notable closed source commercial renderers are BodyViz, CTVox, DICOM-AVC, and ScanIP. BodyViz is a Windows and iOS volume renderer used in universities for anatomy instruction and in hospitals for surgical planning. CCTV and DICOM-AVC are both mobile device volume renderers available for free on iOS. The customer reviews for both volume renderers are below three out of five stars as of this writing, indicating there is some improvement

المنارة للاستشارات
www.manaraa.com

needed in the mobile volume rendering area. ScanIP is a Windows desktop volume rendering solution focusing on Computer Aided Design (CAD) models and Computational Fluid Dynamics (CFD) models as well as medical data. The focus on CAD and CFD is apparent in their interface design choices and the powerful analysis tools available with the software.

Academics have been creating volume rendering tools for years that rival the commercial tools in many respects. Sinus Endoscopy is a volume rendering tool developed at Otto-von-Guericke University of Magdeburg to plan endoscopic sinus surgeries. GPU raycasting is used to provide an accurate representation of the complex inner workings of the patient's nasal cavity [162]. Academics at Vienna University in Austria and Christian Michelsen Research in Norway created a virtual reality volume renderer that integrates CT scans and simulated radiotherapy dose distribution data into a single rendered volume for advanced treatment visualization [128]. ImageVis3D was created at the University of Utah and is a free cross-platform volume renderer available on Windows, Mac, Linux, and iOS. The program was developed by the NIH/NIGMS Center for Integrative Biomedical Computing (CIBC) to be open source and useful for developers to extend the functionality [161]. Voreen is another open source volume renderer that began as an academic project at the University of Munster, Germany and supports Windows, Linux, and Mac operating systems [156]. The open source and cross-platform nature of Voreen has made it a popular tool for academics looking to do research in medical visualization with over 400 articles returned by Google Scholar.

The Volume Image Processing and Rendering Engine (VIPRE), was work created at Iowa State University to attempt to address the issue of moving academic research in volume rendering into common use. VIPRE was designed on an open source core with

المنارة للاستشارات
www.manaraa.com

the ability to run across desktops, iOS devices, and immersive virtual reality clusters [2,163,164].

3.5.2 Dynamic Data Volume Renderers

Dynamic data renderers, often referred to as 4D volume renderers, visualize multiple lower resolution scans to visualize things like blood flow or brain activity over time. With dynamic volume renderers, the size of the data alone is a difficult problem to solve with a linear increase in the size of the data with each new time step. Another challenge is the different ways 4D data can be visualized. Some of the more common 4D volume renderers are Siemens' syngo, EnSight, Osirix, MeVisLab, and Vaa3D.

From a closed source, commercial standpoint, Siemens' syngo, Amira, and EnSight are all popular tools. Siemens is very involved with medical visualization technology with activities ranging from building MRI scanners to developing software to visualize the scans themselves. Syngo is their visualization software used for processing and visualizing fMRI data. It is a Windows based software, but can also be viewed across platforms using a web browser.

Amira, is a software package developed by FEI Visualization Sciences Group for visualizing 3D and 4D micro-CT, PET, and Ultrasound data. One of the nicer features is their automatic segmentation algorithm for dividing up volumetric data into objects of interest. It is also one of the few software packages to claim support for Virtual Reality CAVE systems and stereo rendering. However, it looks like both of these features are not easily enabled out of the box according to their documentation.

EnSight, is a Computer Aided Engineering (CAE) tool used for visualizing flow data in engineering analysis such as Computational Fluid Dynamics (CFD) simulations. This is fundamentally an engineering tool and does not claim support for medical

www.manaraa.com

imaging data, even though it would potentially work to view it in the correct file format. EnSight is the other volume rendering software that claims support for VR CAVES and stereoscopic rendering. Again, this functionality does not come standard and is an increase in the licensing cost.

Fortunately, there is free and open source software available for volume rendering including OsiriX, MeVisLab, and Vaa3D. OsiriX is a FDA approved volume renderer that started as an academic project and then moved to an open source development strategy supporting Mac OS X and iOS. OsiriX is popular with Mac users in the medical community, but their lack of cross-platform support and extensibility, prevent it from becoming more widely adopted, especially by developers.

MeVisLab is an open source visualization tool that is built in a modular fashion to allow developers and researchers to extend the software's abilities by adding or replacing modules. It provides cross-platform support for Windows, Mac, and Linux PCs. From an open source standpoint, it is well maintained with feature releases coming approximately every year. In 2011, a paper was wrote about using MeVisLab on six different 4D MRI use cases [165]. The additional module for 4D data visualization is unfortunately restricted to fluid flow visualization, which is not helpful in fMRI use.

Vaa3D is a bit different in than it does not use direct volume rendering or raycasting, but uses ios-surface rendering algorithms like marching cubes and marching tetrahedrons [166,167]. It is a cross platform software application that provides extensibility through plugins. Recently it appears to have gone the way of so many open source projects and has lost contributors with the last software update published in 2012.

Most dynamic volume rendering applications are aimed at the traditional desktop computer because of the ease of programming and the advanced graphics hardware

المنارة للاستشارات
www.manaraa.com

capabilities currently available. NeuroPub is an fMRI visualizer that is built for iOS mobile devices. However, this implementation is extremely limited with specific requirements such as supporting 32-bit 64x64 MRI slices. Without providing the source code or plugin capabilities, there is no way to extend the functionality. While providing a 4D volume rendering capability on a mobile device is excellent, it is too limited to be useful in its current state.

All of the above software applications provide 4D volume rendering functionality in some capacity or another. Most of these tools do not use raycasting for the renderer, but instead rely on more computationally efficient rendering methods. This is especially true when discussing mobile devices and immersive virtual reality, where there are currently no 4D volume raycasters available.

3.6 Research Issues

The literature review of volume rendering tools has shown that significant gaps exist for general 4D volume visualization. There are currently no open source or commercial 4D volume raycasters available for virtual reality or mobile devices. It is therefore necessary to investigate the feasibility of visualizing generic 4D functional volumetric data across dissimilar platforms (desktop, mobile, and virtual reality). Out of this general need, four research issues have been identified that will be addressed in this research:

1. **Explore the feasibility of a generic NIfTI data input capability on desktop, VR, and mobile device platforms.**

   Volume rendering of functional imaging data is often limited to the datasets available. NIfTI data is one of the most common file types used to store

المنارة للاستشارات
www.manaraa.com

functional data. There is currently no object-oriented tool available for developers to use for loading generic NIfTI data.

2. **Assess the feasibility of displaying functional medical data across desktop, VR, and mobile device platforms.**

   One of the most powerful aspects of functional data is the ability to observe changes in the human body over time. There have been many methods proposed to visualize the time component in functional data. This research will develop a method for animating functional brain activity data.

3. **Real-time visualization of high-resolution structural data and low-resolution functional data at the same time.**

   Due to hardware limitations of scanning technologies like MRI, fMRIs can consist of a single high-resolution structural volume and a series of low-resolution functional volumes. Effectively combining both structural and functional data into a single visual is a great challenge that can provide significant advantages in understanding the data. Techniques of multiple volume rendering will be investigated and one method will be implemented and tested for real-time interaction.

4. **Develop a GPU-based 4D volume raycaster for mobile devices supported by the iOS platform.**

   Mobile devices are carried by almost every medical professional around the world. Providing a mobile device solution for viewing 4D volumetric data would allow medical professionals to show patients their data to explain ailments in the exam room. Building an interactive 4D volume renderer for mobile devices is challenging because of the underpowered processors and small amounts of memory. NeuroPub Visualizer is the only

المنارة للاستشارات

www.manaraa.com

iOS application with support for 4D volumetric data, but the implementation is very limited with restrictions like requiring power of two data only. Additionally, the methods used by NeuroPub Visualizer do not appear to be volume raycasting. Multiple techniques for 4D volume rendering will be developed and examined to determine the best method for 4D mobile volume rendering in real-time.

المنارة للاستشارات

www.manaraa.com

CHAPTER 4: VISUALIZING FMRI DATA USING VOLUME RENDERING IN VIRTUAL REALITY

Published in the 2015 Interservice/Industry Training, Simulation and Education Conference

Joseph Holub, Eliot Winer

Iowa State University
Ames, Iowa

jholub@iastate.edu, ewiner@iastate.edu

4.1 Abstract

Medical imaging technology has changed patient diagnosis since the first x-ray in 1895 [4]. Powerful imaging technologies like Computed Tomography (CT), Ultrasound, and Magnetic Resonance Imaging (MRI) are now used daily. One study showed preoperative imaging for potential appendicitis reduced unnecessary surgeries by 87% [168]. With the 2015 Defense Budget including $47.4 billion for the Military Health System [169], enhanced use of imaging for improved patient care and cost reduction is critical.

More recently, functional MRI (fMRI) technology was developed to extend medical imaging beyond 3D static models to capture physiological changes over time. Currently, fMRI is used for applications from examining beating hearts to mapping brain activity in real-time. fMRI has the potential to dramatically change how illnesses are diagnosed, planned for, and treated.

Methods created for visualizing fMRI data in the academic realm have rarely made their way into commercial software toolsets. For example, there are no software libraries available for researchers to create their own fMRI visualization tools. Another consideration needs to be the visual manner (i.e., 2D, 3D, or 3D stereo) in which these

www.manaraa.com

visual representations are created. Previous research on visualizing medical data has demonstrated improved understanding of spatial relationships when using stereoscopic 3D over traditional 2D representations. This indicates that virtual reality may be a superior medium for visualizing fMRIs.

This paper presents research to: 1) make readily available fMRI software libraries and 2) use these libraries to visualize fMRI data in immersive VR. The method was tested on a desktop computer as well as a large multi-walled VR system running off a cluster of computers. Preliminary results have indicated that visualizing fMRI data in VR can be done in a computationally efficient manner. Multiple fMRI datasets were used for evaluation by measuring load times and frame rates.

4.2 Introduction

Medical imaging was first discovered in 1895 with the first x-ray [4] and has dramatically changed the way medical professionals diagnose and treat patients. Newer imaging technologies like Computed Tomography (CT), Ultrasound, and Magnetic Resonance Imaging (MRI) are now used daily for patient diagnosis. The use of medical imaging has been shown to reduce errors during diagnosis. One study showed preoperative imaging for potential appendicitis reduced unnecessary surgeries by 87% [168]. With the 2015 Defense Budget including $47.4 billion for the Military Health System, enhanced use of imaging for improved patient care and cost reduction could not only improve patient care, but reduce costs for unnecessary procedures as well [169].

One medical imaging technology that has seen increased use in the last 20 years is functional Magnetic Resonance Imaging (fMRI). Functional imaging allows physiological changes of a patient to be viewed over a period of time. Specifically, an MRI scan captures 4D time-varying dynamic views rather than 3D static ones. Currently,

المنارة للاستشارات

www.manaraa.com

fMRI is used for applications from examining beating hearts to mapping brain activity. fMRI has the potential to dramatically change how illnesses are diagnosed, planned for, and treated.

One issue with the expanding use of fMRI technology is the lack of visualization tools available. Most advanced visualization methods for fMRI are created in academic research but have rarely made their way into commercial software toolsets. The toolsets that do provide fMRI visualization do not allow researchers to create new methods within their system. This results in researchers building custom solutions to test new methods, often only working on a specific dataset or those from a specific fMRI machine.

Another issue to consider with fMRI data is the visual manner that the data is presented. Most toolsets use 3D visuals to display fMRI data but very few are designed for stereoscopic 3D. This is an oversight considering previous research on visualizing medical data has demonstrated improved understanding of spatial relationships when using stereoscopic 3D (i.e., immersive virtual reality) over traditional 3D representations [170]. This research indicates that immersive virtual reality (VR) may be a superior medium for visualizing fMRI data.

There are many different types of hardware to consider when working with VR. High-end cave automatic virtual environments (CAVEs™) and head mounted displays (HMDs) provide the highest quality immersive experience at a prohibitive cost. Lower cost virtual reality solutions such as the Oculus Rift provide an acceptable VR experience at a much more obtainable cost. Companies like Facebook are jumping into the low-cost market [34] potentially opening the doors to experience VR on more platforms than ever before.

This paper presents work done to widen the use of fMRI by creating the Volume Image Processing and Rendering Engine for fMRI (VIPRE-fMRI), a readily available

المنارة للاستشارات
www.manaraa.com

software Application Programming Interface (API) to render, view, and interact with functional medical imaging data in immersive VR. The challenges of designing a library to work across multiple operating systems (e.g., Windows, OS X, and Linux) as well as multiple hardware platforms (e.g., immersive 6-sided CAVE™ and desktop computer) will be discussed in detail. The resulting applications will then be assessed for load times, frame rates, and other metrics across multiple fMRI datasets.

4.3 Background

MRI imaging technologies generate 2D cross-sectional slices of a patient's body along a single axis, typically from head to feet. These 2D slices can then be combined to create a single 3D block of data. This is referred to as volumetric data. Figure 25 show the process of capturing 2D anatomical slices along the axis of a body and combining them together to form a single 3D block of volumetric data.

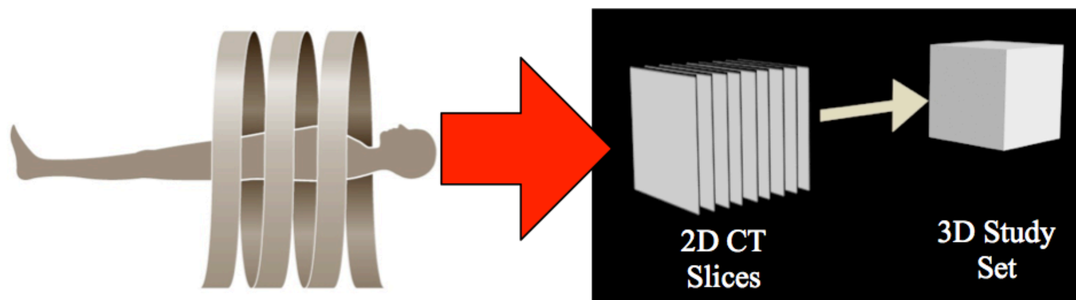


Figure 25. Process of obtaining a 3D volumetric dataset from 2D anatomical slices

MRI technology was first discovered in 1946 by Felix Bloch and Edward Purcell, but was not used for imaging purposes until the early 1970s [6]. MRIs use strong magnetic fields and radio waves to measure tissue densities in the human body. The density information is visualized as a set of 2D slice images of the patient. The radio waves are used to resonate magnetically charged nuclei like Hydrogen (1H) and the resulting resonance is used to create the 2D slices [7].

المنارة للاستشارات
www.manaraa.com

While MRI imaging technology has been around for the last 50 years, fMRI technology has only been available for the last 20. Functional MRIs are most often used for brain activity scans, but that is far from their only use [28]. This technology has impacted research in areas such as science, clinical practice, cognitive neuroscience, mental illness [30].

4.3.1 Volume Rendering

A volumetric dataset, like fMRI data, does not contain any defined surfaces or edges, therefore surface rendering techniques are inadequate to use for visualizing volumetric data. Instead, a different rendering method known as volume rendering is required.

There are five main categories of volume rendering with their own advantages and disadvantages. Iso-surface rendering attempts to reduce the complexity of volume rendering by representing the volume data as a surface comprised of geometric primitives [11,12]. Image Splatting is a direct volume rendering method using overlapping basis functions to represent the voxels [13–15,171]. Shear Warp determines the face of the volume data that is most parallel to the viewing plane and then casts rays orthogonally from the base plane through the data. The resulting image is then projected onto the viewing plane [18,19]. Texture Slicing generates viewport-aligned slices parallel to the viewing plane and then composites them together for a final image [20,21].

Raycasting is a direct volume rendering method that casts rays in the viewing direction from each pixel of the screen. Figure 26 shows a simplified version of raycasting with the user's eye on the left looking at the viewing plane. The rays are cast through the volume starting at the viewing plane and moving right through the volume. The rays sample the volume and composite these samples to obtain a final pixel value.

المنارة للاستشارات

www.manaraa.com

Of the five methods described, raycasting is generally considered to achieve the best visual representation. This research will utilize a raycasting implementation [22,23].

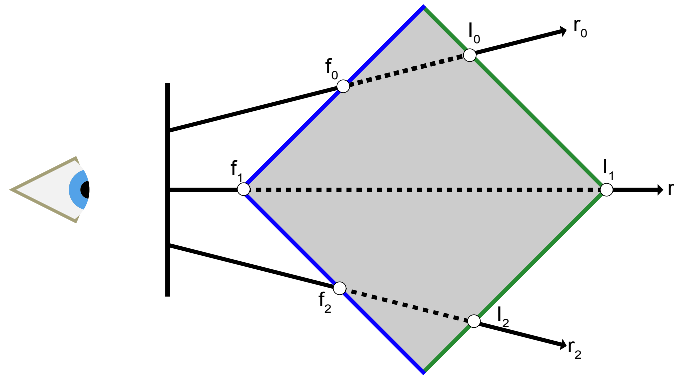


Figure 26. Example of raycasting with rays being cast from the viewing plan on the left through the volume on the right

Volumetric raycasting typically goes through a rendering pipeline similar to that seen in Figure 27. The actual pipeline itself might differ depending on the application or the type of data being used, but the basic pipeline is the same for all volume renderers. Segmentation looks at dividing the volume data into sub volumes. This is often used for things like identifying tumors. Gradient computation finds all the edges in the volume to be used in the shading step to improve depth perception. Resampling is the process of stepping through the volume and sampling the volume intensity at each step along the ray. This sample is then classified to determine whether it should be used in the final image. Classification is typically done using opacity transfer functions that map the sample intensity with an opacity, how transparent the sample will be in the final image. The sample can then be given a red, green, and blue, color value using any number of different color transfer functions. The goal of coloring is not to achieve photorealism but to make specific anatomical features stand out during viewing. The sample can then be shaded using any number of shading techniques with Phong shading being a popular

المنارة للاستشارات
www.manaraa.com

choice for its computational efficiency. The final step composites the current sample with the previous samples taken along the ray to achieve a final pixel color to display.

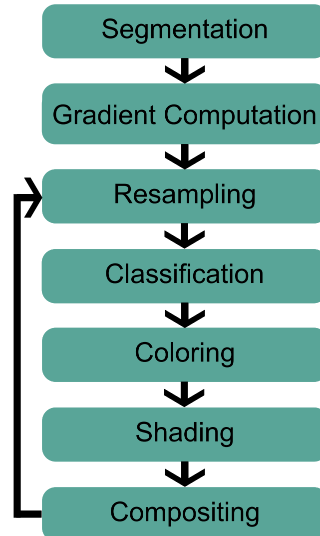


Figure 27. Volume rendering pipeline example

4.3.2 4D Volume Rendering Tools

An effective way to categorize the tools available for visualizing 4D volumetric data is closed source and open source. Siemens' syngo, Amira, and NeuroPub are closed source visualization tools. Syngo is Siemen's visualization software used specifically for processing and visualizing fMRI data. It is a Windows based software, but can also be viewed across platforms using a web browser. Amira is a software package developed by FEI Visualization Sciences Group for visualizing 3D and 4D micro-CT, PET, and Ultrasound data. NeuroPub is the lone mobile application that could be found at the writing of this paper to provide fMRI visualization. However, the implementation is extremely limited with requirements like only supporting 32-bit 64x64 pixel slices.

المنارة للاستشارات
www.manaraa.com

There are three commonly used open source 4D volumetric visualization tools, Osirix, MeVisLab [165], and Va3D. OsiriX is an FDA approved volume renderer that started as an academic project and then moved to open source development supporting Mac OS X and iOS. The MeVisLab tool is built in a modular fashion to allow developers and researches to be able to extend the software's abilities by adding or replacing modules. The module for 4D data visualization is unfortunately restricted to fluid flow visualization, which is not entirely helpful for visualizing fMRI data. Vaa3D is a cross platform tool that uses iso-surface rendering instead of direct volume rendering algorithms [166,167].

All of the above tools provide 4D volume rendering in some capacity or another. In general the closed source tools are limited in their extensibility because they do not provide source code nor plugin capabilities. However, the closed source tools are designed to specifically visualize fMRI data unlike the open source tools Osirix and MeVisLab. Vaa3D and MeVisLab allow extension through modules and plugins respectively, but this functionality is still limited to within the context of the application itself. Even if a suitable plugin could be created, the application is limited by a user interface that is not optimized for interacting with fMRI data, resulting in a poor user experience. There is also the possibility of a tool becoming outdated or discontinued as appears to be the case with Vaa3D where the latest recorded software update was published in 2012.

All of these 4D volume rendering tools have their drawbacks. There is no single cross platform tool that provides advanced rendering functionality in an easy to implement and extensible library or API. Such a tool would allow researchers to build new rendering methods off a common platform while providing developers with a tool for creating new types of fMRI applications.

المنارة للاستشارات
www.manaraa.com

4.3.3 4D Volume Rendering in VR

Amira and EnSight are the only tools researched at the time of this paper that provided 4D volume rendering support in an immersive VR environment. Neither tool is designed to specifically visualize fMRI data, but they both look to have the necessary building blocks. The VR support for both systems does not work out of the box, but requires extensive configuration. In the case of EnSight, a whole new licensing cost is associated with allowing VR support. Both the low number of tools and the additional support costs speak to the difficulty in creating a 4D volume renderer for immersive VR systems.

Previous academic research on 4D volume rendering medical data in immersive VR systems looked primarily at volume rendering multiple volumes. The VIVIAN system [127] and the work done by Patel, et al., [128] used an orthogonal texture slicing volume renderer in a CAVE™ environment. Academics at Vienna University in Austria and Christian Michelsen Research in Norway created a virtual reality volume renderer that integrates CT scans and simulated radiotherapy dose distribution data into a single rendered volume for advanced treatment visualization [128].

The volume rendering technologies outlined in this section show a sampling of the many advances in the field. The advanced rendering methods from research rarely make it into commercial applications because researchers built custom one-off solutions. The rendering tools available are not easily extensible to implement new methods. These commercial and open source rendering tools fail to provide advanced rendering methods in a cross platform tool allowing development for multiple devices. If a developer wants to build a fMRI visualization application for VR they must learn and use one rendering library, but they must learn a different rendering library if they want to build the same application for a mobile device or a desktop computer. A single cross

المنارة للاستشارات
www.manaraa.com

platform volume renderer that allows researchers to test new rendering methods as well as providing developers with tools for developing cross platform fMRI applications is needed.

4.4 Methodology

VIPRE-fMRI was created to address this need for a single cross platform 4D volume rendering application programming interface (API). An API is a set of routines, protocols, and tools that are used to build software applications. They can be thought of as building blocks for a software developer that can be combined in an infinite number of ways to create an application. In this case, VIPRE-fMRI provides routines for things like processing fMRI files, creating new coloring modes, adjusting the rendering loop, and many more. Figure 28 shows an overview of the VIPRE-fMRI library architecture. The boxes at the bottom of the figure represent the lowest level systems and libraries being used. The higher boxes are built using the lower boxes. For example, VIPRE builds upon OpenSceneGraph (OSG) which builds upon OpenGL. At the very top is the final visualization application using all the components.

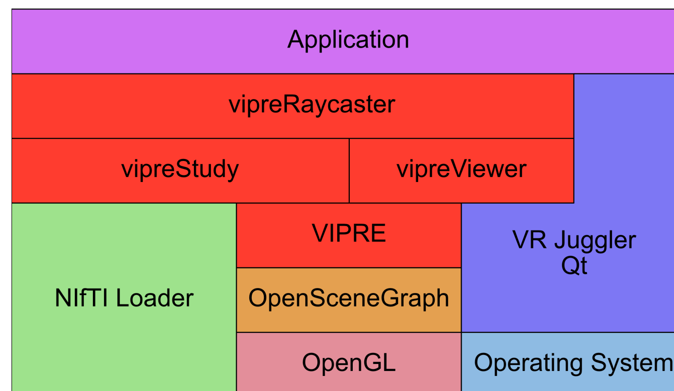


Figure 28. Simplified library architecture

There are many different aspects of designing a cross-platform 4D volume rending API that must be considered to ensure compatibility and ease of use. When considering

المنارة للاستشارات

www.manaraa.com

design requirements, it was important to consider the hardware constraints faced both by academic researchers and military field personnel. Low cost commodity hardware is typically used by both groups with multiple operating systems powering this hardware. The harsh conditions faced by military personnel require reliable systems that can be supported for many years. Five design requirements were defined based on these constraints:

1. Cross-platform support for Windows, Mac OS X, and Linux
2. Stable API
3. Real-time rendering (efficient)
4. Support for desktops, laptops, and immersive VR platforms
5. Encapsulate platform customization at the engine level

The rendering engine's graphics core was the decision with the most impact on all five requirements. On desktop computers, the low level rending API choice is typically between DirectX [172] and OpenGL [173]. DirectX is a Windows only API, while OpenGL is cross-platform supporting Windows, Mac, and Linux as well as mobile devices through OpenGL Embedded Systems (OpenGL ES). OpenGL has been around for over 20 years and is a C language based state machine implementation. The cross-platform support and long history makes OpenGL the best choice for VIPRE-fMRI.

The low level nature of OpenGL provides developers with lots of customization options to harness every bit of computational power from the graphic processing unit (GPU). The downside to using OpenGL is that a code optimized for one GPU will not be optimized for a different GPU and may not work at all without coding changes. The complexity of OpenGL must be abstracted from the application level to make OpenGL code more portable between GPUs. VIPRE-fMRI does this by encapsulating OpenGL in

المنارة للاستشارات
www.manaraa.com

another API, OpenSceneGraph. This encapsulation allows a developer to call the same OSG routines on all devices and OSG will handle calling the correct OpenGL routines to achieve the optimal result on every GPU.

4.4.1 Third-party APIs

The third-party APIs were selected to provide OpenGL encapsulation as well as additional functionality. When considering third-party APIs for VIPRE-fMRI, it was important to consider licensing and the proprietary nature of the APIs so they could be used by both the military and academics. A key consideration for military use is they require a higher level reliability over longer periods of time. Therefore, for a third-party library to be considered it must meet these four requirements:

1. Free and proprietary licensing terms (LGPL, BSD, MIT, etc.)
2. Cross-platform support for Windows, Mac OS X, and Linux
3. Large active development community
4. 5+ years old

The licensing restrictions for each API chosen were one of the most important elements to consider. Using free and proprietary licensing allows both academics, military, and commercial developers to take advantage of the outcomes of this research and helps to foster broader adoption.

The last two listed requirements were added to ensure the third-party libraries used in VIPRE-fMRI will be supported and stable in the future. Large development communities and long life spans reduce the risk that third-party libraries might disappear in the near future. Based on these four requirements, OpenSceneGraph and VR Juggler were chosen for inclusion in VIPRE-fMRI.

المنارة للاستشارات
www.manaraa.com

4.4.2 OpenSceneGraph

OpenSceneGraph (OSG) is an open-source, cross-platform graphics API for high-performance applications. First created in 1999 by Don Burns and Robert Osfield [174], OSG has grown steadily with their latest stable release (Version 3.2.1) including contributions from 519 developers.

The main reason OSG was selected for this project is that it encapsulates OpenGL functionality in an object-oriented framework that focuses on performance, scalability, portability, and productivity. OSG supports view-frustum culling, occlusion culling, OpenGL Shader Language (GLSL), and display lists, which are required for GPU raycasting implementations. Combining OSG's windowing system independence and support for OpenGL means OSG will work across all required software and hardware platforms.

Game engines, such as Unity3D, were considered instead of OpenSceneGraph because of their graphics power and intuitive graphical user interface. Unity3D is freely available and cross platform with an active development community. However, Unity3D was removed from consideration because it did not provided support for large graphics clusters like those that power VR CAVE™ systems.

4.4.3 VR Juggler

VR Juggler is a cross-platform, open-source virtual reality software development environment designed for executing immersive applications across multiple hardware platforms [175]. Established in 1997 at Iowa State University's Virtual Reality Applications Center, VR Juggler has seen continued use and development. VR Juggler can be integrated into many existing systems with support for multiple rendering APIs including OpenSceneGraph.

المنارة للاستشارات

www.manaraa.com

The main function of VR Juggler is supporting display and device abstraction for immersive hardware systems. This abstraction layer allows VR Juggler applications to be compiled once and run on multiple hardware configurations without recompiling. VR Juggler is also extremely efficient as it was shown to be one of the fastest cluster synchronization APIs available [176]. Large visualization clusters, like those used to power VR CAVE™ facilities, are efficiently synchronized through the use of swap barriers, which ensures all cluster nodes swap their front and back buffers simultaneously.

4.4.4 NIfTI File Reader

4D volumetric data can be stored in various file formats such as, Analyze/SPM, MINC, AFNI, and NIfTI [31]. For fMRI brain scans, NIfTI is the most commonly used file format. A NIfTI file reader library was created to read in the available fMRI data. The library is built using C++ and the C++ Standard Library to be both lightweight and cross-platform. The library converts a file path into a study object containing access to the header information as well as a C array of the raw image data. For this research, the library was extended to support retrieving the imaging data as a set of *osg::Image* objects for use with the VIPRE-fMRI rendering engine. The architecture allows developers to make calls to *vipreStudy* that in turn use the NIfTI file reader library to load and store fMRI datasets in a VIPRE-fMRI optimized way.

4.4.5 VIPRE-fMRI Framework

The third-party APIs were combined into VIPRE-fMRI, a cross platform 4D volume rendering API. The original VIPRE was proposed by Noon [2] but was limited to volume rendering a single 3D volume. This work extends VIPRE by adding 4D volume raycasting. VIPRE-fMRI supports Windows, Linux, and Mac OS X. The rendering is

المنارة للاستشارات

www.manaraa.com

abstracted from the developer, allowing different windowing APIs like Qt, Cocoa Touch, and VR Juggler to be used on their respective hardware platforms. A simplified version of the VIPRE architecture was previously shown in Figure 28.

To test the validity of the VIPRE-fMRI architecture, two different applications were built. One for a large CAVE™ environment and one for a commodity desktop computer. Both were built using the same base raycasting code with the differences being in the interaction and windowing systems. Both applications will be discussed in depth in the following sections.

4.4.6 CAVE™ Implementation Details

The CAVE™ implementation set up an OSG scene graph with clipping planes, a bounding box, and fMRI data groups under a series of transform nodes for manipulating the scene. There were also standard light and camera nodes. Within OSG, calls are made to set up GPU raycasting by creating a 3D texture from the fMRI volume data, and loading the vertex and fragment shader programs. OSG then handles all the OpenGL calls itself, abstracting the complexity from the developer. The result is a single OSG scene designed to raycast fMRI data.

VR Juggler is then used to visualize this fMRI scene in a CAVE™ environment. VR Juggler works by having a single instance of the application running on each node of the cluster. One of the nodes acts as the “master” node, in charge of synchronizing data across the rest of the “slave” nodes. The “slave” nodes are responsible for drawing their part of the scene based on a unique view frustum. A VR Juggler configuration file defines the nodes in the rendering cluster and passes in a view frustum for the instance of the application on each node to render.

المنارة للاستشارات

www.manaraa.com

Navigating around the scene and controlling the renderer is done through the use of a Logitech gamepad controller. The dual joysticks are ideal for navigating around 3D environments and the buttons allow quick access to change rendering features like rendering modes, windowing, coloring, and clipping of the dataset. VR Juggler is built on a rendering loop comprised of preframe, late preframe, and draw method calls. This allows the “master” node to synchronize gamepad input across the cluster using preframe and late preframe methods before redrawing the scene.

Figure 29 shows a user standing inside the C6 six-sided CAVE™ environment looking at MRI data using the immersive demo application. The user is navigating around using a wireless Logitech gamepad controller.

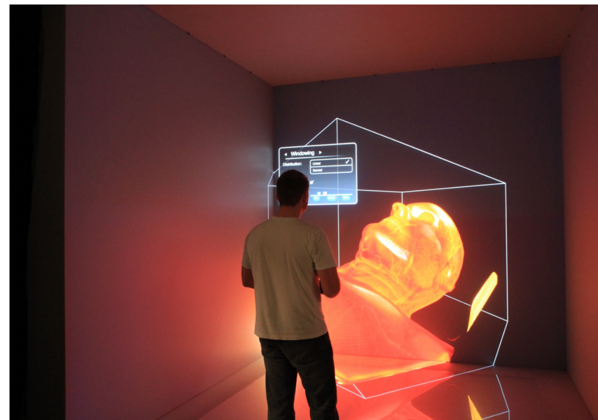


Figure 29. Immersive VR application in a 6-sided CAVE™ environment

4.4.7 Desktop Implementation

The desktop application was built off the same raycasting renderer code with the only modifications being the windowing system used to display the results. Qt is a freely available user interface library that was used as the windowing system to display the raycasting results. OSG’s window independence allows VIPRE-fMRI to be rendered directly within an OpenGL widget provided by Qt. No other changes were required to the

www.manaraa.com

VIPRE-fMRI rendering codebase. A graphical user interface was created around the Qt OpenGL widget to all users to adjust things like coloring and tissue density through the use of interface elements like button and slider bars. The mouse and keyboard interactions were also handled by Qt and passed to the VIPRE-fMRI renderer as needed. An example of the desktop application viewing fMRI data can be seen in Figure 30.

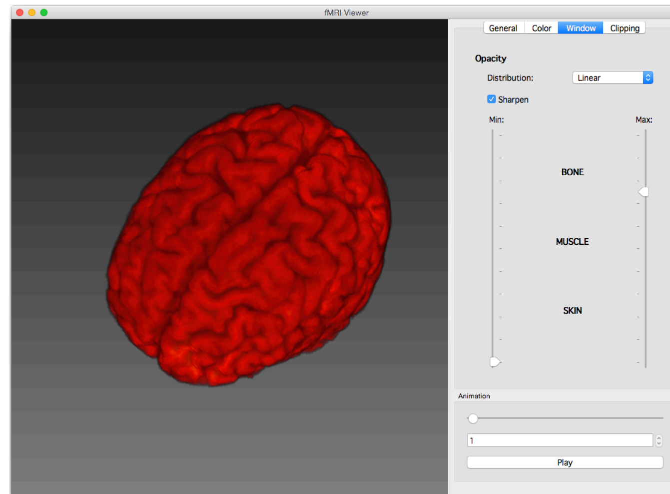


Figure 30. Desktop application visualizing a human brain

4.5 Results

The proposed 4D volume rendering library was tested on two different hardware configurations, a 2013 MacBook Pro and the C6. The C6 is the world's highest resolution six-side VR CAVE™ located at Iowa State University. Twenty-four Sony 4K projectors achieve 96 million pixels per eye. A 96 node rendering cluster comprised of NVIDIA Quadro 6000 graphics cards is required to feed the 24 projects. The C6 is uses the Red Hat Enterprise Linux operating system. The size of the cluster provides an excellent test bed for the 4D volume rendering library's performance. The C6 uses an

www.manaraa.com

Intersense ultrasonic tracking system for tracking objects in the CAVE™, specifically, the user's head position.

The immersive VR application was compared to the desktop application. The desktop application was evaluated on a 2013 MacBook Pro running OS X 10.10.3 with a 2.6GHz Intel Core i7 processor, 16 GB of RAM, and a NVIDIA GeForce GT 750M graphics card. The desktop application was used as a baseline to judge the efficiency of the immersive VR application as well as showing the library's cross-platform support includes a CAVE™, desktop computers, and two different operating systems.

The efficiency of the 4D volume rendering library was testing using two different NIfTI data sets. The first data set is a single time step scan of a human brain comprised of 128 slices of 256 by 256 pixel data and will be referred to as "Dataset 1." The second dataset is a fMRI brain scan comprised of 126 time steps with 24 slices of 64 by 64 pixels at each time step and will be referred to as "Dataset 2." Frame rates were tested both while rendering a single time step, referred to as "Static", and while animating through the time steps, referred to as "Dynamic".

The data load times for all three systems can be seen in Table 1. The Dataset 2 had a longer load time on average than Dataset 1. This is to be expected with Dataset 2 being roughly 50% larger than Dataset 1. The load times for the C6 were three to four times longer than the Desktop condition. Evaluating the time difference between Desktop and C6 must consider that each node in the C6 using network storage instead of local storage. Given the C6 can load data from the network drive synchronously, the load time difference can be attributed to the network speeds.

www.manaraa.com

Table 1. Average data set load times in milliseconds (ms)

	Desktop	C6 CAVE™
Dataset 1	3352.8	11257.9
Dataset 2	5195.6	15535.6

The average frame rate for each application is shown in Table 2. Frame rate can be used to determine the computational efficiency of a system. Interaction is key to medical imaging and frame rates of 30 frames per second (fps) or higher are desired for good interaction. Both systems were tested using Dataset 1 as well as the Dataset 2 while viewing a single static time step as well as animating through the time steps referred to as dynamic. The C6 application did not achieve the desired 30 fps, but still produced respectable 20 fps that allow real-time navigation with a slight lag. Neither implementation showed much difference in frame rate between datasets with the Desktop steady around 60 fps and the C6 steady around 20 fps. This would indicate that the raycasting implementation is similarly efficient regardless of rendering a static dataset (MRI) or animating a dynamic dataset (fMRI). While this result is interesting, it should be noted that the results could drastically change depending on the datasets used.

Table 2. Average application frame rates in frames per second (fps)

	Desktop	C6 CAVE™
Dataset 1Static	54.993783	20.000000
Dataset 2 Static	60.935748	19.562180
Dataset 2 Dynamic	60.223459	19.657370

المنارة للاستشارات
www.manaraa.com

4.6 Discussion

This research presented VIPRE-fMRI, a cross platform 4D fMRI volume rendering library for visualizing fMRI data on multiple immersive VR hardware platforms. Two sample applications were built to test VIPRE-fMRI. The first platform was the world's highest resolution six-sided VR CAVE™ and the second platform was commodity desktop computer. The sample applications demonstrate the success of creating a single 4D volume rendering library capable of being used on multiple hardware configurations and operating systems.

The most difficult aspect of building a cross-platform 4D volume rendering library was selecting tools (software libraries and languages) that would work across all the variety of systems. OpenGL and the GLSL shader language were chosen for this reason. OpenSceneGraph and VR Juggler were both chosen to abstract hardware differences from the software developer. VIPRE-fMRI utilizes OSG's window independence in a way that allows developers to wrap the renderer in an OpenGL based windowing system with minimal effort.

Frame rates show the library to be able to perform raycasting at 60 fps on commodity desktop hardware and 20 frames per second on a 96 node graphics cluster. The frame rates could be increased with several optimizations. Currently, the raycasting shader is a single program with multiple computationally expensive conditional statements. These conditional statements rely on user input to determine exactly how to render the data (e.g., Minimum Intensity Projection versus Compositing). Breaking the shader up into multiple shader programs and loading the program that is currently needed would eliminate the conditionals would improve frame rates. Neither application had any difficulties in loading in the datasets and in particular, synching that data across

المنارة للاستشارات

www.manaraa.com

a cluster. The frame rates and load times indicate that visualizing fMRI data in VR can be done in a computationally efficient way.

The potential for medical imaging to reduce unnecessary surgeries can have a significant impact on the $47.4 billion 2015 budget for the Military Health System [169]. This potential depends on military medical professionals obtaining excellent diagnostic tools using advanced visualization tools. The last decade of mobile “app” development has proven that giving developers accessible development tools and libraries can produce thousands of new and innovative applications. Most current 4D volume rendering tools are closed systems that do not allow developers access to the underlying visualization methods. VIPRE-fMRI is designed to be an accessible cross platform library that will allow developers to build new and innovative visualization tools that can be used by military medical professionals to improve the health of active and retired military while reducing the overall health care cost.

Future work on this library will look to use low-cost head mounted display (HMD) technologies like the Oculus Rift. Low-cost HMDs will greatly expand the reach of VR. Testing VIPRE-fMRI on Oculus would ensure the library is capable of handling a VR hardware platform that may dominate military and civilian life in the near future. VR Juggler is ideally suited to handle implementing VIPRE-fMRI on a HMD because it is designed to abstract the display hardware from the code. A single VR Juggler configuration file would be needed to move the immersive VIPRE-fMRI sample application for the C6’s displays to the Oculus display. To make the application immersive, a VR Juggler plugin would be needed for the Oculus Rift’s head tracking hardware (accelerometer, gyroscope, manometer, and near infrared tracker). This plugin would allow a user to move their head around and have the medical data respond like it was floating right in front of them.

المنارة للاستشارات
www.manaraa.com

CHAPTER 5: ENABLING REAL-TIME VISUALIZATION OF FUNCTIONAL MAGNETIC RESONANCE IMAGING ON AN IOS DEVICE

Article in Review to the Journal of Digital Imaging

Joseph Holub
Graduate Student, Virtual Reality Applications Center, Iowa State University

Dr. Eliot Winer
Associate Director, Virtual Reality Applications Center. Professor of Mechanical Engineering, Iowa State University

5.1 Abstract

Powerful non-invasive imaging technologies like Computed Tomography (CT), Ultrasound, and Magnetic Resonance Imaging (MRI) are used daily by medical professionals to diagnose and treat patients. While 2D slice viewers have long been the standard, many tools allowing 3D representations of digital medical data are now available. The newest imaging advancement, functional MRI (fMRI) technology, has changed medical imaging from viewing static to dynamic physiology (4D) over time, particularly to study brain activity. Add this to the rapid adoption of mobile devices for everyday work and the need to visualize fMRI data on tablets or smartphones arises. However, there are few mobile tools available to visualize 3D MRI data, let alone 4D fMRI data. Building volume rendering tools on mobile devices to visualize 3D and 4D medical data is challenging given the limited computational power of the devices. This paper describes research that explored the feasibility of performing real-time 3D and 4D volume raycasting on a tablet device. The prototype application was tested on a 9.7" iPad Pro using two different fMRI datasets of brain activity. The results show that mobile

المنارة للاستشارات
www.manaraa.com

raycasting is able to achieve between 20 and 40 frames per second for traditional 3D datasets, depending on the sampling interval, and up to 9 frames per second for 4D data. While the prototype application did not always achieve true real-time interaction, these results clearly demonstrated that visualizing 3D and 4D digital medical data is feasible with a properly constructed software framework.

5.2 Introduction

The last twenty years have seen medical imaging technology expand beyond the traditional viewing of anatomical features inside the body to functional medical imaging that looks at these features over time. Functional magnetic resonance imaging (fMRI) is by far the most common functional imaging modality and is heavily used in identifying brain activity [28].

The impact of fMRI on the areas of science, clinical practice, cognitive neuroscience, mental illness, and society have been significant [30]. In 1992, there were zero publications with the word fMRI in the title, abstract, or a keyword. In 2005, there were close to 2500 research publications meeting those criteria, showing the growth in fMRI research being performed [29].

As the use of functional imaging technology increases, it is important to provide visualization tools that allow researchers and medical professionals to harness the power of fMRI. This research presented in this paper focuses on visualizing fMRI data obtained from brain activity studies and stored in the NIfTI file format. However, any functional medical data stored in the NIfTI format can be visualized and the general methods themselves can be applied to any type of functional medical data.

المنارة للاستشارات
www.manaraa.com

5.2.1 Magnetic Resonance Imaging

Most medical imaging technology can obtain data from any direction or angle. MRI technology typically samples the body in an orthogonal grid starting at the head or feet and moving along the axis of the body. A depiction of this “scanning” process is shown in Figure 31. The scanner generates a series of 2D “slices” orthogonal to the axis of the body. This series of 2D slices can then be combined to create a single 3D block of data representing the entire scan of the patient. This 3D block of data is known as a volume representation.

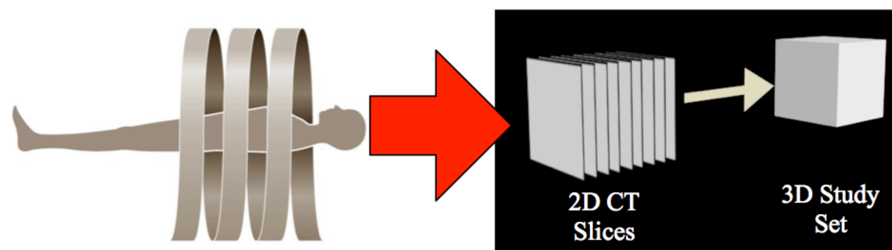


Figure 31: Volumetric data captured from CT and MRI machines.

5.2.2 3D Volume Rendering

The volume representation must then be visualized to enable medical professionals to gain insight about the patient. Volumetric data is different from the surface data traditionally used in computer graphics programming, such as computer games, because it does not contain any defined surfaces or edges. Therefore, surface rendering techniques are inadequate to use for visualizing volumetric data. Volumetric data requires a different type of graphics rendering technique, known as volume rendering, for proper visualization in 3D. Unlike surface rendering that represents an object as a series of vertices making up an outer shell to show the object’s shape, volume rendering sees an object as a three-dimensional lattice of vertices similar to a Rubik’s Cube. This allows volume rendering to visualize not only the shell of the data,

المنارة للاستشارات
www.manaraa.com

but the values inside the volume as well. Figure 32 shows the difference between surface rendering, in the left image, where only the faces of the data can be seen and volume rendering, in the right image, where the internal data can be viewed.

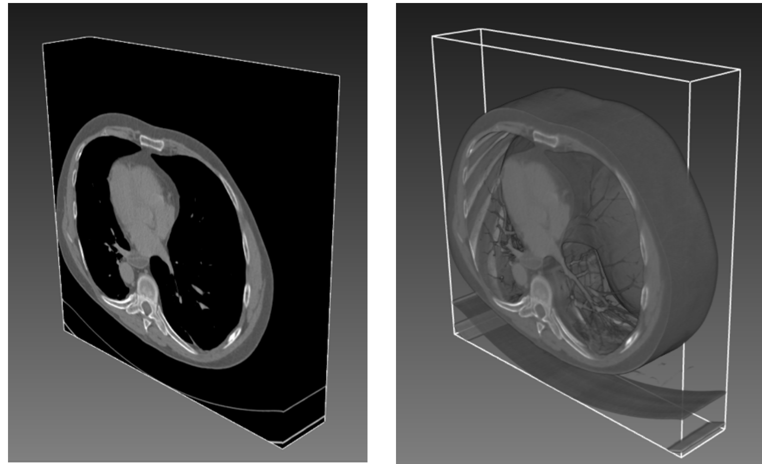


Figure 32: Visualizing the surface data (left) versus volume rendering to see the internal data (right).

Volume rendering can generally be broken down into four main techniques, 1) image splatting [13–16], 2) shear warp [18,19], 3) texture slicing [20,21], and 4) raycasting [22,23]. Of all these methods, raycasting produces the most accurate visual representation, but at a high computational cost.

5.2.3 Volume Raycasting Pipeline

Raycasting is a direct volume rendering technique that involves casting linear segments (i.e. “rays”) from each pixel in the frame buffer through the volume in the view direction [36]. Points along the path of the ray that intersect with the volume are sampled and composited together to generate the final pixel color.

The volume rendering process begins with acquiring a volumetric dataset. The volumetric data is comprised of a set of samples, known as voxels, in three dimensions

www.manaraa.com

(i.e. *x*, *y*, and *z*). Each voxel contains a measured value, *v*. In medical imaging, the voxel values typically represent tissue densities relative to a known substance such as air or water, and are one-dimensional values. In fMRI brain scans, the voxel value is a one-dimensional value representing the blood oxygenation level-dependent (BOLD) signal change [28]. This is used because neural activity has been linked with local changes in brain oxygen content [46]. The change of oxygen levels over time also requires the addition of a time component, *t*, to the volume data sample, resulting in five values per voxel (*x,y,z,t,v*).

The volume rendering pipeline defines the steps taken by the computer to translate volumetric data into a computer generated 3D representation. Rendering pipelines differ depending on the application or the type of data being used. The pipeline used in this research consists of Resampling, Classification, Coloring, and Compositing steps [11]. These four steps make up the core of the volume rendering pipeline. Additional steps for gradient computation and shading were not needed as medical imaging is typically not concerned with photorealistic rendering. Instead, medical imaging is concerned with providing contrast in the data to help identify physiological objects. Thus, the steps chosen for this research are common to many volume renderers available as open source or commercially.

5.2.3.1 Resampling

Resampling is the first step in the render pipeline. The goal of resampling is to measure the volume data, comprised of voxels, at different positions in three-dimensional space. Each voxel is commonly represented as a cube in medical imaging where each vertex represents a value. When sampling the volume, it is rare that a ray can sample a voxel's vertex directly. More commonly, the sampling point resides within

المنارة للاستشارات
www.manaraa.com

the voxel. Trilinear interpolation can be used to approximate the value given its position within a voxel. Trilinear interpolation was chosen for this research due to its reasonable visual quality and minimal computational requirements [27].

The sampling techniques used in resampling are one of the primary differences between different volume renderers. For raycasting, an imaginary ray is cast into the volume from each screen pixel and samples are taken along the ray at evenly spaced intervals. Figure 33 shows a 2D representation of raycasting where the person's eye is on the left looking at the computer screen represented by the black line, as rays are cast from each pixel into the gray volume.

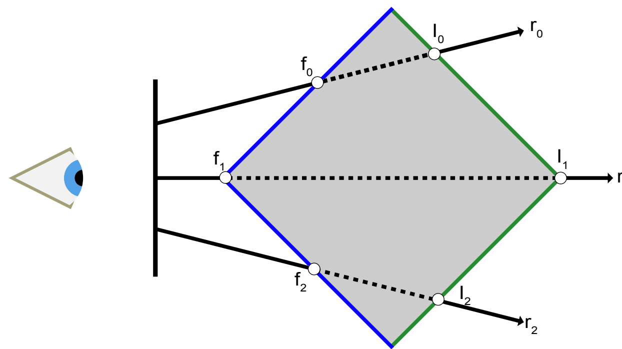


Figure 33: 2D example of raycasting.

For speed considerations, it was important to first determine the bounds of the volume before casting the rays. This cuts down on the total number of samples taken by the ray and thus speeds up rendering. The bounds of the volume were found by sending imaginary rays, r_i, from each pixel through the scene in the viewing direction. Each ray looks for the first intersection with the volume, the blue boundary f_i, and the last intersection with the volume, the green boundary l_i. Rays that did not intersect with the volume were rendered the background image color.

www.manaraa.com

Rays that intersect the volume take samples at specified intervals, shown by the dashed lines in Figure 33, where each dash would represent a single sample. The sampling interval can be changed to accommodate different implementation goals. The tradeoff is smaller sampling intervals result in higher quality images at the expense of more computations. The sampling interval used in this implementation was set to the width of a single voxel, after ad-hoc testing. This value can be changed in further implementations if desired.

5.2.3.2 Classification

Classification determines the subset of samples that make up the final image by mapping the sample's intensity to an opacity value between zero and one. This range indicates how much of that voxel's data should be included in the final image with zero being completely transparent and one being completely opaque. The mapping between voxel intensity and opacity is known as an opacity transfer function [50,54]. Creating an opacity transfer function can be very complex depending on the type of data being viewed. The opacity transfer function used in this implementation is a normal distribution with the high and low values equating to completely transparent and the median equating to completely opaque [55].

5.2.3.3 Coloring

Coloring is used to map a sample's intensity to a color using color transfer functions. The purpose of coloring is not always to provide photo realism, but to provide contrast within the data to help identify desired features. Coloring is challenging because volume data typically assigns a single value (intensity in medical imaging) to each voxel and not a traditional three-component color, like red, green, and blue. Therefore, coloring methods must intelligently convert a single value into three different values in a

المنارة للاستشارات
www.manaraa.com

way that makes desired structures more visible. This conversion is typically accomplished by using different color transfer functions (a.k.a color lookup tables [177]) for each color channel. Different organizations and institutions, such as the National Institute of Health (NIH), create their own color transfer functions for different types of data. This research used some of these institutional color transfer functions.

5.2.3.4 Compositing

Compositing is the final step in the rendering pipeline and is the process of taking all the values sampled by the ray and combining them into a single color to be used for that pixel. The front-to-back compositing method used Equation 1 to calculate the final intensity value, *I(x,y)*, for each ray. The final intensity value is a sum of the sample point intensities, I_i, multiplied by all the transparencies *(1-αj)* encountered previously along the ray. Put another way, each voxel sample can be thought of as a pane of colored glass that has some opacity, α. If the first pane of glass is completely opaque, the following panes of glass cannot be seen. If the first pane of glass is 75% opaque, the second pane of glass can be seen but its color will only be 25% visible at best. Compositing works the same way, the higher the opacity of a sample, the less the following sample's intensity can contribute to the final intensity value.

$$I(x,y) = \sum_{i=0}^{n} I_i \prod_{j=0}^{i-1} (1 - \alpha_j) \qquad 1)$$

Each voxel sample's intensity value is a combination of the color, C_i, from the color transfer functions and the opacity, α_i, from the opacity transfer function. Equation 2

المنارة للاستشارات
www.manaraa.com

shows how these two values are multiplied together to compute the voxel sample's color. The higher the opacity, the more intense the resulting color contribution is.

$$I_i = C_i \times \alpha_i$$ 2)

Front-to-back compositing is continuously evaluating the current voxel sample's intensity and blending it with previous samples. This constant evaluation is what allows front-to-back compositing to achieve a performance benefit from early ray termination. When the cumulative opacity reaches 1.0, there is no need to continue compositing along the ray because the contributions of all subsequent samples would be zero. Thus the render loop can be exited for that ray with no impact to the final pixel color.

5.2.3.5 Raycasting Optimizations

Much research has been performed to handle the enormous computational resources required by raycasting and improve its speed. One of the most common methods is early ray termination (also known as adaptive termination) [91]. Early ray termination was originally proposed by Whitted [92] as an adaptively terminating raytracing algorithm. Levoy [93] took this idea and developed two front-to-back volume raycasting algorithms. The first was a case where the ray traversal should be terminated when an opaque voxel is reached. The second case terminates the ray traversal when the accumulated opacity reaches a user-specified level where additional samples will not dramatically change the final pixel color.

Another significant challenge of raycasting is the limited texture memory available in commodity computers. Traditional 3D MRI datasets can be on the order of 512x512 pixels per slice, which equates to 0.78 megabytes (MB) per slice if each pixel is 3 bytes (e.g., JPEG photos). It is common to have datasets on the order of 300 to 1000 slices,

www.manaraa.com

resulting in 236 to 786 MB per dataset, respectively. It is common to store the slice data as a 32-bit per pixel texture for the graphics card, resulting in a required memory of 786 MB to 1.05 gigabytes (GB). A dedicated graphics card is typically necessary to handle this amount of texture memory effectively and most commodity computers do not come with dedicated graphics cards.

5.2.4 4D Volume Raycasting

Moving from 3D to 4D to accommodate functional data presents its own sets of problems and limitations. All of the issues relevant to 3D volume rendering remain present in 4D rendering, but new ones get added to the list. In 3D volume rendering, there is one set of static volumetric data. Organs are caught in a single position and represented in that "frozen" state. In 4D rendering, there is volumetric data over time (i.e. per a defined time step). This causes the amount of data to increase linearly with the number of time steps taken. Using the previously discussed 3D dataset as an example, if the same 236 to 786 MB 3D volume was recorded for 100 time steps, the amount of memory increases to 23.6 to 78.6 GB. Thus, methods are required to handle the increase in data, along with other issues to be discussed later, to allow rendering at interactive speeds. The most common method for dealing with the increase in data when moving from 3D to 4D data involves decomposing the original data into two individual datasets. The first is a high-resolution scan holding all the structural data that does not change over time. For example, in an fMRI of the brain, this would equate to the structure of the brain itself. The second scan is a set of lower resolution scans capturing changes. In the previous example, this would be the brain activity.

Dividing the fMRI data into two scans reduces the total amount of volumetric data that changes from one frame to the next, but introduces the challenge of rendering

المنارة للاستشارات
www.manaraa.com

multiple volumes at the same time. Some researchers approached this problem with the assumption that multiple volumes were correctly aligned in 3D space and never moved in relation to each other [137–139]. This assumption works well for data sets like fMRI brain activity where the functional and structural data never move independently of each other. Once the volumes are aligned in 3D space, the ray cast into the volume samples once from each volume at every sampling point along the ray. The process of scaling and aligning two volumes of different resolutions will be discussed in more depth in section Combining 3D and 4D data into one representation.

Multiple volume rendering uses the same volume rendering pipeline described earlier, with the exception of sampling all volumes at each point in space during resampling. There are two different methods of combining those samples together. The first is One Property per Point (OPP), where a single volume's sample is selected to be the final value at that point [139]. The ability to use only a single volume's value was found to be useful when certain characteristics were more important than others (e.g., EEG brain activity would be more important to visualize than the skull).

The second option is Multiple Properties per Point (MPP), where all samples are combined together to achieve a single value for that point [125,126]. The biggest difference in MPP is typically where in the volume rendering pipeline the samples are combined together. Data mixing can occur in the sampling, coloring, or image stages depending on the desired output [137]. Manssour, et al., proposed a method where two volume values were combined after the sample's color was determined for each volume and combined using a weighting function [138]. Wilson, et al., experimented with three different methods of data mixing: 1) chose one volume's value as the only value, 2) use a weighted function to combine all the values, or 3) use a single volume value each for the red, green, and blue color channels [131]. The advantage to MPP is the ability to mix

المنارة للاستشارات
www.manaraa.com

the samples from both volumes into the final color, versus the OPP method that must choose one volume per sample. This provides more control over the final sample color and allows more complex visualization strategies to be used. However, the visualizations can become too complex to decipher if too many volumes are mixed or if the data mixing is not done in a consistent and intuitive way.

5.2.5 Mobile Raycasting

Advances in computational and graphics processing speeds allow for real-time volume raycasting on desktop and immersive VR hardware platforms [164,178]. However, there has been a dramatic shift in the US and worldwide toward mobile devices in recent years [33]. There is a need to expand the reach of medical imaging visualization technologies toward supporting these types of devices as they become more common in the medical field.

Volume raycasting for mobile devices has been a challenge due to the limited hardware computational power and a lack of support for 3D textures. Mobile devices are generally underpowered compared to desktop computers because they intentionally sacrifice computational power for size to keep devices thin and light and provide ample battery life. However, the largest limitation came from the lack of 3D texture support that required 3D volume data to be stored as a series of 2D textures. Storing volume data in 2D textures required a greater number of graphics operations to map 3D voxel locations to a pixel in in set of 2D textures. Additionally, raycasting samples generally fall between voxels requiring some method of interpolation to obtain a correct value. Interpolating 3D positions within a set of 2D textures required a prohibitively large number of graphics operations. It was just not possible to perform all of these calculations quickly enough for real-time applications on a mobile device [2].

المنارة للاستشارات
www.manaraa.com

The computing power provided by mobile devices and specifically the graphics processing units (GPU) has changed in recent years with mobile GPU technology approaching the performance seen in some laptop computers. There has also been an increase in support for 3D textures that have proven critical for volume raycasting. New GPU languages like Metal [179] and Vulcan [180] reduce computational overhead while increasing draw rates over previous software libraries like OpenGL ES [181].

To date, there has not been a successful real-time volume raycasting implementation for mobile devices. Achieving real-time volume rendering has required using a less computationally expensive method, such as orthogonal texture slicing [163]. The only available application for visualizing fMRI data in 3D on a mobile device is NeuroPub [182], an application for Apple's iOS platform. NeuroPub does not use volume raycasting and limits the functional data to only 32-bit NIfTI format with 64x64 pixel slices. The lack of available fMRI volume raycasting applications indicates the difficulty inherent in building a mobile raycaster that can handle generic fMRI data robustly with real-time interactivity for a user.

The goal of the research presented in this paper was to explore the feasibility of real-time visualization of fMRI data on a mobile device using volume raycasting. The application was developed to support any 3D static anatomical and 4D fMRI brain activity data that can be generically stored in the NIfTI file format.

5.3 Materials and Methods

A prototype application was built for iOS tablets using the new Metal graphics programming language to test the feasibility of implementing a 3D and 4D volume raycasting method on a mobile device. The goal of the prototype was to develop an efficient raycasting algorithm that could be run on mobile devices and observe real-time

المنارة للاستشارات
www.manaraa.com

interaction when viewing both 3D static and 4D dynamic volumetric data. Currently, there is no standard for what frame rates qualify as real-time, so it is difficult to quantify whether an application is real-time or not. Miller identified 0.1 seconds as an acceptable graphical response time after input from a light pen [1]. Therefore, a framerate of 10 frames per second or higher will be considered acceptable as real-time for this research. Volumetric data load times were also considered given they are critical to determining the usability of an application in everyday medical situations. Medical professionals are not likely to wait significant periods of time (e.g., 5-10 minutes for more) to load a single dataset in their everyday work flow. The following sections will describe the implementation of the raycasting algorithm used in this research, along with the challenges and contributions to the field.

5.3.1 Raycasting with the iOS Metal Shading Language

As mentioned in the Mobile Raycasting section, 3D volume raycasting on mobile devices has historically been a challenge due to the limited hardware computational power and a lack of support for 3D textures. Moving to 4D volume rendering of fMRI data increases the amount of data and the number of volumes required to be stored in textures.

The Metal shading language was introduced for iOS devices with support for 3D textures and a focus on low level control of the graphics pipeline. Metal provides the capabilities to develop a more advanced volume rendering method for mobile devices than was previously possible. Implementing the volume raycasting algorithm using Metal required development and optimization of both the application initialization and the render loop for best performance. The initialization of the application required the setup of the Metal graphics pipeline. This includes steps such as initializing the vertex buffers,

المنارة للاستشارات
www.manaraa.com

textures, and compiling the shaders for use. Shaders are small programs that can be written to run on a GPU and is where the volume rendering logic is written. The render loop consisted of passing in the vertex buffers, textures, and uniforms to the shader to process. A shader uniform is simply a value passed into the shader from the application. In general, it is desirable to move as many operations as possible into the initialization step to allow faster drawing in the render loop step. However, doing so sometimes requires increasing the memory footprint on a limited device. The volume raycasting pipeline, as implemented using Metal, can be seen in Figure 35. The pipeline is broken up into the Application Compilation, Application Initialization, and Render Loop steps. The entire raycasting pipeline implementation using Metal will be discussed in more depth in the following sections.

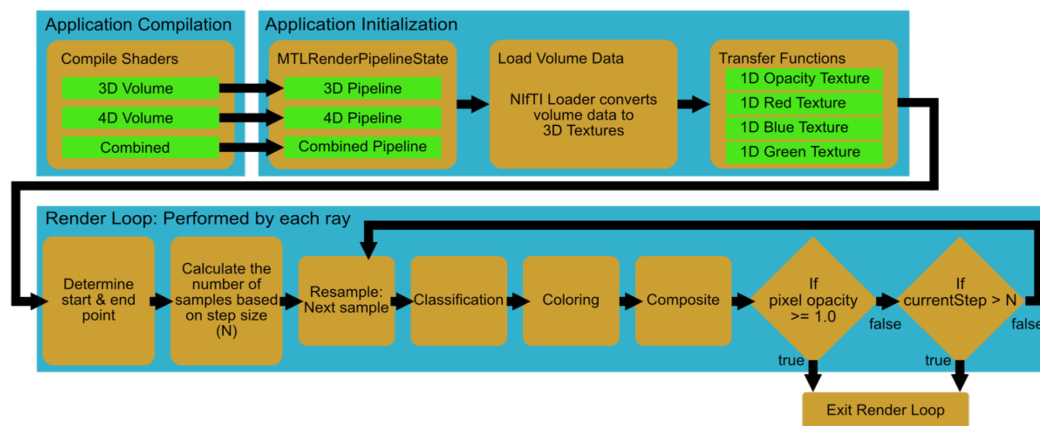


Figure 34: Volume raycasting pipeline implementation using Metal.

5.3.1.1 Initializing the Volume Raycaster

Optimization of the volume renderer began by moving as many operations out of the run loop as possible. For example, Metal provides the option of precompiling the shaders during application compilation. This is different from the previous standard, OpenGL ES, that required compiling the shader code on application launch or

المنارة للاستشارات

www.manaraa.com

dynamically as the application ran. Moving the shader compilation into application compilation, improved both application launch speed and frame rates when changing between different shader programs during application runtime. Three different shader programs were written to render the 3D and 4D data.

Traditionally, a shader accepts the volumetric data as an input in the form of a texture or a set of textures. In this research, each volume is passed into the shader as a 3D texture. Each texel in the texture represents a single voxel's intensity value. The high-resolution structural data is stored in a single 3D texture and the low-resolution functional data is stored in a series of 3D textures where each texture represents a single snapshot in time.

Each shader program was included within a unique *MTLRenderPipelineState*, which is a Metal object that encapsulates the rendering configuration used during a graphics rendering pass. Three *MTLRenderPipelineState* objects were created upon initialization with the first for only 3D structural data, the second for only 4D functional data, and the third for a combination of the two. Creating individual *MTLRenderPipelineState* objects allowed the rendering process to be optimized for each type of data. *MTLRenderPipelineState* objects are expensive to create but efficient to swap, therefore it made sense to create the difference states at initialization and swap them as needed at run time.

Loading data is another time consuming process that is best performed upon initialization. When loading the volume data, there were two options for storing it. The first was to store each volume as a 3D texture and the second was to store the volume as an array of values. The second method used less memory and was faster at initialization, but required converting the array of values for a volume into a 3D texture anytime that volume was to be rendered. Both of these methods were implemented and

المنارة للاستشارات

www.manaraa.com

tested to determine if the rendering speed gains were significant enough to overcome the increased memory footprint and increased application initialization time.

Another operation that was moved from the render loop to initialization was the generation of opacity and color transfer functions. A 1D texture was used to store the opacity transfer function and a set of three 1D textures were used to store each color transfer function. Opacity transfer function textures are traditionally created/updated every time the desired range of voxel values to visualize changes, which is an expensive process. Instead, a single texture was created at initialization and a user defined minimum and maximum intensity value were passed to the shader through a uniform. Using the minimum and maximum intensity values, the shader can determine whether a voxel value falls within the range and then use the opacity transfer functions to determine an opacity, otherwise the opacity is zero. This method is more efficient than rebuilding the texture after every change.

5.3.1.2 The Raycasting Render Loop

Once the graphics pipeline is initialized, the rendering loop can begin drawing. The shader starts by determining the start and end points of the ray for the current pixel. It is important to calculate the start and end points of the ray to determine the number of times the ray must sample. The number of samples the ray takes is directly proportional to rendering time. The more samples the ray takes, the longer it takes to render a frame and the lower the frame rates.

The shader then starts sampling at the start point and continues along the ray until it reaches the end point. The ray is sampling in 3D world space, but the volume data is defined as a 3D texture with its own coordinate system. Sampling the volume requires both the ray and the volume texture to be in the same coordinate space. In computer

المنارة للاستشارات
www.manaraa.com

graphics, the model-view matrix is used to map a 3D model, the volume, from its local coordinate system to a position and orientation relative to the camera, the screen. To map the ray's world location relative to our 3D texture, the inverse of the model-view matrix was applied to the ray.

The converted ray can then be used in the raycasting loop to step along the ray and sample the volume at different points. Each sample along the ray checks if the intensity value was within the minimum and maximum intensity value bounds defined by the user. Anything outside those bounds was set to zero. The intensity value can then be used to sample the opacity transfer function texture for the correct opacity. If the opacity is not greater than zero, the ray moves to the next sample. If the opacity is greater than zero, the intensity value is used to determine a color using the color transfer function texture. The opacity and color are then combined to form the sample's contribution to the final pixel color. Equation 2 defines how the individual components are combined together to achieve a final pixel color. This sampling process continues until the end of the ray is reached or the pixel color becomes completely opaque.

5.3.1.3 4D Volume Rendering with Metal

Moving from 3D rendering to 4D rendering required creating a different shader to handle changing volume data. The interest in 4D data comes from the change in activity from the baseline at different moments in time. Therefore, it was necessary to create a fragment shader that accepted two 3D textures, one representing the baseline activity and the second representing the time step of interest. The 4D fragment shader samples both the baseline and the current time step textures. The two intensity values are differenced to visualize the current activity at that time step.

المنارة للاستشارات

www.manaraa.com

The amount of noise associated with the functional data necessitated an activity threshold to be implemented to only show changes in activity that reached a certain magnitude. The threshold value was set with a slider in the user interface and the value was passed into the shader as a uniform. The threshold method was chosen due to its simplicity and computational efficiency compared to other filtering methods.

The functional data did not go through the same coloring process as the structural data. Instead, the color was determined by multiplying the color red by the magnitude of the activity. This results in higher levels of activity being displayed as more red in the final image. Any color could be used for this specific representation.

5.3.1.4 Combining 3D and 4D Data into One Representation

The third fragment shader program combined both the 3D structural data and the 4D functional data into one representation. This required passing in one 3D texture for the structural volume, one for the functional baseline volume, and one for the functional current time step volume. The implemented method for combining multiple volumes was built on the assumption that none of the volumes move independent of each other. This assumption allowed the efficient casting of a single ray and sampling all three textures at the same location. While this assumption is a gross simplification of the general multi-volume rendering problem, it is not incorrect with fMRI data, where the structural and functional volumes never move independently.

The challenge with this multi-volume rendering method comes from mapping the sample point in 3D world space to a point on each texture in their own space. This becomes especially tricky when each volume contains textures with different resolutions and scalings. Figure 35 shows a 2D example of the mapping issue. The left-side of the figure shows the structural (blue) and the functional (red) textures that represent the

المنارة للاستشارات

www.manaraa.com

voxel data of their respective volumes. Both textures use a normalized coordinate system with (0,0) being the upper left corner and (1,1) being the lower right corner. Each box in the image represents a voxel. The right-side of the image shows the world space where both volumes are aligned and scaled with the functional volume being scaled up equally in both directions. The world space uses a coordinate system where the center of the volumes is at (0,0) and the height and width of the structural volume is 10. If the renderer samples the volume at world space location (0,0), the mapped texture position would be (0.5, 0.5) for both the structural and the functional texture. However, if the renderer samples the volumes at (5,-4), the sample falls outside the bounds of the functional volume but not the structural volume. Therefore, the structural texture is sampled at (1, 0.9) and the value for the functional volume is set to zero as it does not exist at that point.

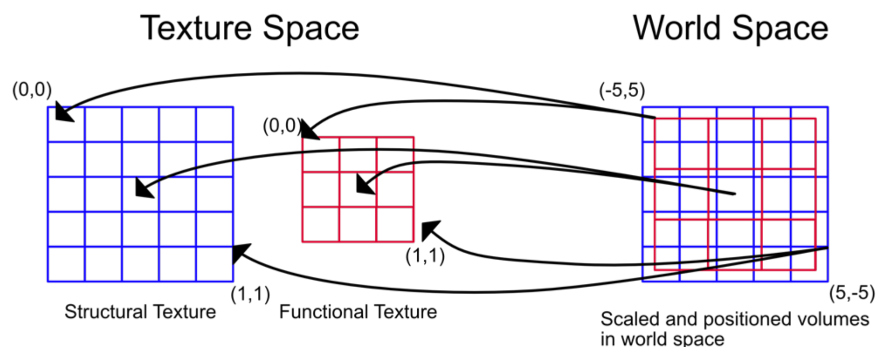


Figure 35: Mapping sampled points in 3D world space to 2D textures.

Once both volumes are correctly sampled, the intensity values must be classified and colored. The structural volume's intensity value went through the same opacity and coloring operations as previously described for 3D rendering. The functional data went through the same coloring process as described above for 4D rendering where the

المنارة للاستشارات

www.manaraa.com

activity level is directly proportional to the intensity of a single color, in this case red. The structural and functional color components were then added together to obtain a final color value for that sample. This process was then repeated for every sample along the ray. The result can be seen in Figure 36 where the structural data is colored using a blue coloring scheme and the functional data is represented in red.

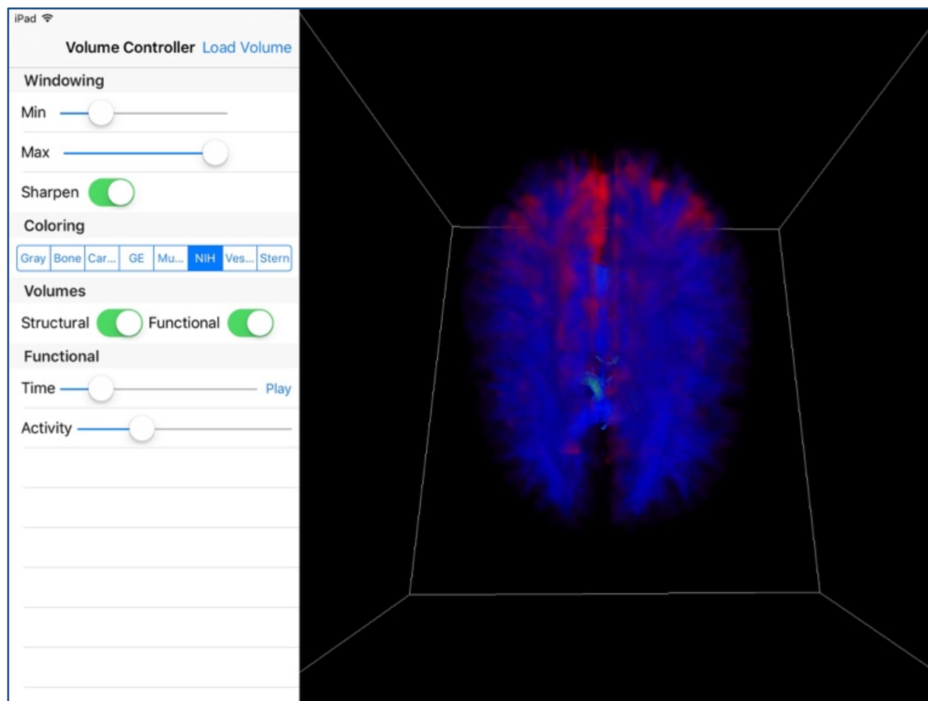


Figure 36: Combined structural and functional data for the brain.

5.4 Results

Evaluating the feasibility of this real-time volume raycasting implementation relied on measuring data initialization time, frame rates, and the memory footprint. The prototype raycasting application was tested on a 2016 9.7" iPad Pro running iOS 9.3.4. The iPad had a 64-bit Apple A9X dual core processor with 2 GB of built in memory to power a 2048 x 1536 pixel screen.

المنارة للاستشارات

www.manaraa.com

Two representative brain activity fMRI datasets were used for testing. Each dataset consisted of a high-resolution structural volume and a set of low-resolution functional volumes. For both datasets, the structural component was 256 by 256 pixels per slice with 128 slices. The functional component of Dataset 1 was 64 by 64 pixels per slice, with 24 slices per time step and 126 time steps total. The functional component of Dataset 2 was 80 by 80 pixels per slice, with 41 slices per time step and 114 time steps total. All fMRI data was stored in the NIfTI file format with 16 bits per voxel. The functional data was captured with a three second time interval for Dataset 1 and 2.5 second time interval for Dataset 2.

The prototype application was evaluated for data initialization times using these representative fMRI datasets. Data initialization included the setup of the rendering pipelines and creation of all necessary Metal objects. Table 3 shows the measured results of the data initialization time compared to the resulting memory footprint. The first condition looked at the implementation where only a single time step is converted into a 3D texture upon initialization. The second condition stores all time steps as 3D textures. For each condition, the application was launched and data was loaded ten times to obtain an average.

Table 3: fMRI data initialization times in seconds

	Dataset 1		Dataset 2	
	Single functional texture	All functional textures	Single functional texture	All functional textures
Average (s)	16.4	29.8	18.1	47.8
Standard Deviation	0.272	0.268	0.317	0.337
Variance	0.074	0.072	0.101	0.113
Memory (MB)	362	408	670	958

المنارة للاستشارات

www.manaraa.com

Frame rates were tested between the three different shader configurations for structural, functional, and combined data (both structural and functional). Within the functional and combined conditions, the framerates were measured as a single time step (Single time step) and as animating through time steps (All time steps). The All Time Steps condition was used to determine the performance impact when "playing" back the data as it would have been captured. The frame rates were also compared between the implementation where textures were created upon demand (Single Texture) and when they are initialized upon startup (Preload Textures). Each condition consisted of five runs measuring 100 consecutive frames each with no user interaction. The mobile device was rebooted between runs. Those frames were then averaged for the final frame rates. Table 4 shows the resulting frame rates for the different conditions.

Table 4: Average frame rates in frames per second for a one voxel sampling step

		Structural	Functional		Combined	
			Single time step	All time steps	Single time step	All time steps
Dataset 1	Single Texture	17.908	45.096	4.723	11.845	4.699
	Preload Textures	17.920	51.787	9.316	11.972	9.515
Dataset 2	Single Texture	17.952	31.528	1.983	9.846	1.979
	Preload Textures	17.984	37.345	3.873	11.705	3.864

The results shown in Table 4 were obtained with no user input, and therefore, the volume did not move. A second condition was tested, where the volume was rotated around the vertical axis, with one full rotation every 10 seconds. The results comparing the framerates while rotating the volume with the non-rotation condition can be seen in Table 5. The difference in the results was considered negligible, as it is less than 10% for the conditions tested. This indicates that user interaction does not have much, if any,

المنارة للاستشارات
www.manaraa.com

affect on the framerates, that the size of the dataset, etc. are more influential factors. This will be discussed more in the Discussion section to follow. Therefore, further results only used one condition (no user input) and the results were assumed to not have changed for the rotation condition.

Table 5: Frame rates when rotating and not rotating the volume.

		Structural	Functional		Combined	
			Single time step	All time steps	Single time step	All time steps
Dataset 1	No Rotation	17.920	51.787	9.316	11.972	9.515
	Rotation	18.866	47.854	9.704	11.016	8.217

Framerates were again tested when the sampling step size of the raycasting algorithm was increased from the width of one voxel to the width of two voxels. Increasing the sampling step size will reduce the image quality and result in missing small anatomical features. However, the tradeoff between speed and image quality may be desirable in some situations. Table 6 shows the results of the frame rates when increasing the sampling step size to the width of two voxels for the preloaded texture condition. The change in image quality between the two sampling rates can be seen below in Figure 37.

Table 6: Average frame rates in frames per second for a two voxel sampling step

		Structural	Functional		Combined	
			Single time step	All time steps	Single time step	All time steps
Dataset 1	Preload Textures	41.20	60.03	9.64	22.60	9.45
Dataset 2	Preload Textures	41.04	60.02	3.81	21.55	3.87

المنارة للاستشارات
www.manaraa.com

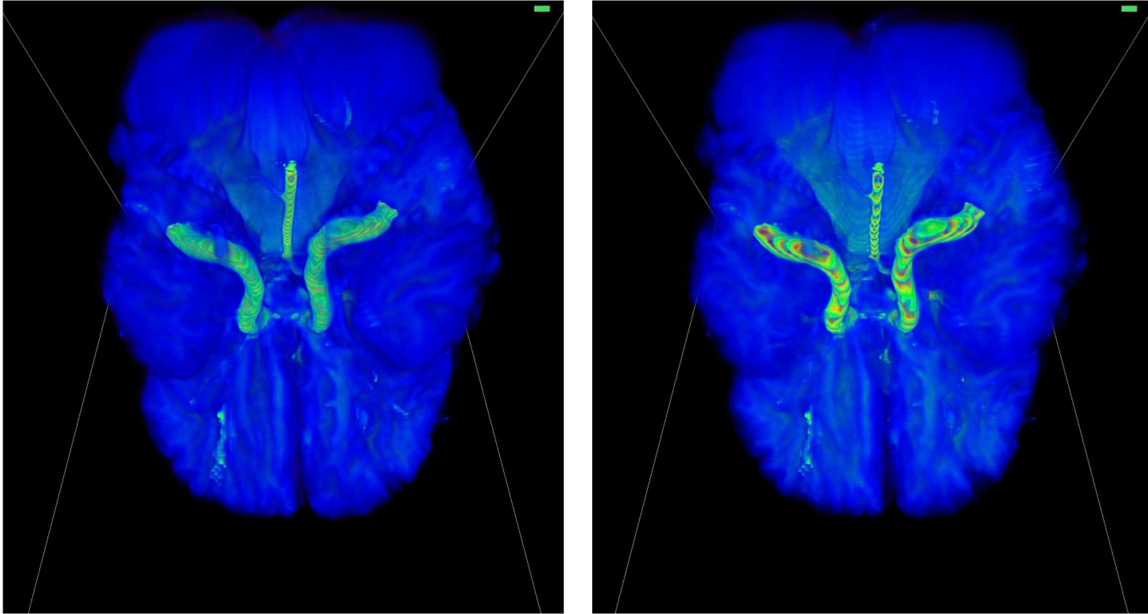


Figure 37: One voxel width sampling step size (left) versus two voxel width sampling step size (right).

5.5 Discussion

This research built a prototype volume raycasting application for 3D and 4D medical imaging to determine the feasibility of visualizing fMRI data on mobile devices. Feasibility was determined by data initialization times, frame rates, and memory consumption. It is impossible to compare this method with others, because there currently is no real-time volume raycaster that support 3D and 4D data. Therefore, the results of this research were compared, when possible, with a fMRI desktop implementation [183] similar to this work as well as a previous iPad volume renderer using orthogonal texture slicing instead of raycasting [163].

The data initialization times were 16.4 and 29.8 seconds with Dataset 1 and 18.1 and 47.8 for Dataset 2 using two different implementations for storing the volume data. The difference in load times is a result of loading all the functional volume data into a set of 3D textures, which is computationally expensive. For comparison, the data loading time of the desktop application using the same fMRI data set was 8.55 seconds [183]. In

www.manaraa.com

general, lower times indicate better usability in the real world. However, initialization happens once upon loading data and does not significantly impact the real-time nature of the application once loading is complete. Since the longest initialization time was only approximately 30 seconds, it is considered near real-time.

Storing all volume data in 3D textures instead of generating those textures on demand increases the random access memory (RAM) footprint of the application from 362 MB to 408 MB for Dataset 1 and from 670 MB to 958 MB for Dataset 2. This is not an insignificant increase in memory given many mobile devices are limited to one to two GB of memory and the operating system will shut down the application if it uses too much of the available memory. The desirability of this memory increase can only be considered when looking at the impact on frame rates.

This research defined real-time interaction as being 10 frames per second [1]. The Single time step frame rates were between 11 and 51 frames per second for Dataset 1 and between 9 and 37 frames per second for Dataset 2. The All time steps conditions were between 4.7 and 9.5 frames per second for Dataset 1 and between 1.9 and 3.8 frames per second for Dataset 2. The Single time step conditions do consistently meet the 10 frames per second goal across both datasets. The All time steps condition was not able consistently meet the 10 fps, but Dataset 1 was close at 9 fps using the pre-initialized textures.

Comparing the frame rates when raycasting Dataset 1 on the desktop application saw the structural condition with 54.9 fps, functional Single time step with 60.93 fps, and functional All time steps with 60.22 fps [183]. The improved frame rates are expected with a 2.6GHz Intel Core i7 processor, 16 GB of RAM, and a NVIDIA GeForce GT 750M graphics card. The comparison iPad application used orthogonal texture slicing and a similar dataset of 256 x 256 pixels per slice and 128 slices and saw frame rates between

المنارة للاستشارات
www.manaraa.com

45-50 fps for the structural static condition [163]. The improved frame rates can largely be attributed to the use of a less computationally expensive orthogonal texture slicing algorithm, even running on an iPad 2.

When increasing the sampling step size to twice the width of a single voxel, the frame rates increased as expected. The Single time steps conditions consistently saw double the frame rates. However, the All time steps condition saw no increase in framerates whatsoever. This would indicate that switching 3D textures representing the functional volumes is the bottleneck in the rendering process. This is not consistent with traditional views on 3D raycasting that identify the number of rays and the number of samples as the bottleneck.

The image quality differences between the single voxel width sampling step size and the double voxel width step size is minimal. It is possible to observe a lighter coloring of the volume due half the number of samples being composited into a final color. It is not recommended to use the double step size for most medical visualization application, as small anatomical objects could be missed. However, for a general viewing, the tradeoff between speed and visual quality may be acceptable.

One previous attempt at volume raycasting on a mobile device did attempt to use an iPad 2 to render a volume consisting of 64 slices at 512 x 512 resolution. The implementation used 2D textures as 3D textures were unavailable, and achieved frame rates under 1 frame per second. This would roughly equate to the structural condition in this research that saw 17 frames per second. This jump in frame rates occurred in a 5-year time span and would indicate higher frame rates are not far way with the rate that computing power is increasing in mobile devices.

The tradeoff between memory footprint and frame rates can be seen in the result of a 46 MB increase for Dataset 1 and a 288 MB increase for Dataset 2 resulting in

www.manaraa.com

double the frames for the All time steps conditions. While memory is at a premium on mobile devices, the doubling of frame rates appears to outweigh the memory increase. Therefore, it is recommended to store all volume data as 3D textures when 4D functional data is used.

5.6 Conclusions

This research looked at the feasibility of implementing a real-time 3D and 4D fMRI volume rendering method on a mobile device given the recent updates in graphic computational performance and new graphics languages. The application initialization, frame rates, and memory footprint results indicate that it is feasible to implement a raycasting method on a mobile device, but the frame rates are still not consistently high enough to be considered real-time. This is especially true for the 4D functional data which saw low frame rates. However, the performance will improve as hardware continues to improve.

Improvements in frame rates can be gained by implementing other performance techniques like empty space skipping through the use of octree data structures. Another possible improvement for brain activity visualization would be to calculate the difference in activity between the baseline and the current time step upon initialization and store the difference in a 3D texture. This would take more time for data initialization, but would reduce the number of textures passed into the fragment shader as well as remove the fragment operations currently used to perform this at run time. This method would also allow more sophisticated filtering of the noisy functional data.

المنارة للاستشارات
www.manaraa.com

CHAPTER 6: 4D FMRI VOLUME RAYCASTING ON DISSIMILAR PLATFORMS

Article in Review to the Journal of Healthcare Engineering

Joseph Holub
Graduate Student, Virtual Reality Applications Center, Iowa State University

Dr. Eliot Winer
Associate Director, Virtual Reality Applications Center. Professor of Mechanical Engineering, Iowa State University

6.1 Abstract

Medical imaging technology has begun embracing the power of 4D functional MRI (fMRI) to diagnose physiological changes over time. Currently, fMRI is heavily used to study brain activity. The tools available for 3D fMRI visualization are currently limited with the majority restricted to desktop machines. With the rapid adoption of mobile and virtual reality devices in the commercial space, the move to the medical industry is not far off. It is therefore important to investigate the feasibility of performing 4D volume rendering on these types of devices. The increase in the amount of data from 3D to 4D fMRI is one challenge. It is also necessary to employ multi-volume rendering to visualize both anatomical and functional data. This paper describes research to explore the challenges of building a real-time 4D volume raycaster for desktop, virtual reality, and mobile platforms. Three prototype applications were built for a laptop, iPad, and a six-sided CAVE (immersive VR) system. The desktop application was used as a baseline comparison of the other two modes. The results show that VR and mobile frame rates still lag behind the desktop application, but they do achieve frame rates between 10 and 20 frames per second.

المنارة للاستشارات

www.manaraa.com

6.2 Introduction

Technology use in the medical field has greatly expanded in the last twenty years. The use of advanced medical imaging techniques has provided unparalleled non-invasive examination methods. The advances in laptops and mobile devices allow medical professionals to expand their use of computers in their everyday workflows. The next frontier of medical devices looks to be virtual reality (VR) and augmented reality (AR), where these technologies are being used to improve learning [184]. Similarly, medical imaging technology has expanded beyond traditional static 2D grayscale images to 3D and 4D functional imaging that show how organs function and move over time. Functional medical imaging has impacted many areas of science, clinical practice, cognitive neuroscience, mental illness, and society [30]. Functional magnetic resonance imaging (fMRI) is the most commonly used form of functional imaging. It's prominent use is in studying brain activity [28].

The increased use of functional imaging technology requires visualization tools that allow researchers and medical professionals to harness the power of fMRI on emerging hardware platforms like mobile and VR. This research looks at the challenges of visualizing fMRI brain activity data across the dissimilar computational platforms of desktop, mobile, and immersive virtual reality. One of those challenges involves the various different formats that functional data can be stored and read. There are multiple different file types, such as DICOM [185] and NIfTI [31], and different data compression techniques, such as JPEG, that can be used. This research will focus on using generic fMRI brain activity stored in the NIfTI file format for sample data. However, the volume rendering techniques described, can be applied to any type of functional medical data.

المنارة للاستشارات
www.manaraa.com

6.2.1 Magnetic Resonance Imaging

Magnetic resonance imaging (MRI) technology can obtain data from any direction or angle, but typically samples the body in an orthogonal grid starting with cross-sectional slices along the long axis of the body (i.e. head to feet). The scanning process generates a series of 2D “slices” orthogonal to the axis of the body as seen in Figure 38. The set of 2D slices can be combined to create a single 3D block of data representing the entire scan of the patient. This 3D block of data is referred to as volumetric data.

A volumetric dataset is comprised of a set of samples, known as voxels, in three dimensions (i.e. *x*, *y*, and *z*). Each voxel contains a measured value, *v*. The voxel values typically represent tissue densities relative to a known substance such as air or water in medical imaging. Neural activity has been linked with local changes in brain oxygen content [46]. Therefore, brain scans look for changes in the blood oxygenation level-dependent (BOLD) [28] within areas of the brain. The change in BOLD is stored as a one-dimensional value, *v*, in the voxel. Measuring the change of oxygen levels over time requires the addition of a time component, *t*, to the volume data sample, resulting in five values per voxel (*x,y,z,t,v*).

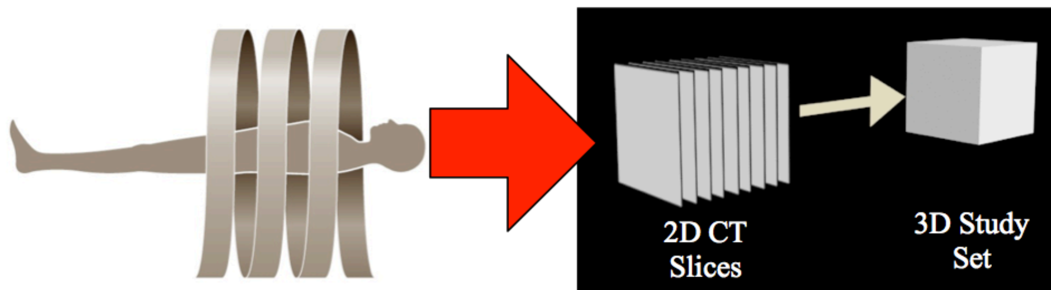


Figure 38: Capturing volumetric data from CT and MRI machines.

المنارة للاستشارات
www.manaraa.com

6.2.2 3D Volume Rendering

The volumetric data must then be visualized for medical professionals to gain any insight. Traditional computer graphics programming (e.g., computer games) use surface rendering methods that rely on a set of vertices to define the outside surface of objects. Volumetric data is different from surface data because it does not contain any defined surfaces or edges. Volumetric data requires volume rendering, a different type of graphics rendering technique, for proper visualization in 3D. Volume rendering sees an object as a three-dimensional lattice of vertices similar to a Rubik's Cube. With a volume containing data inside the object, it is important that volume rendering is capable of visualizing not only the shell of the data, but the inside values as well. Figure 32 shows an example of the difference between surface rendering, in the left image, where only the faces of the data can be seen and volume rendering, in the right image, where the internal data can be viewed.

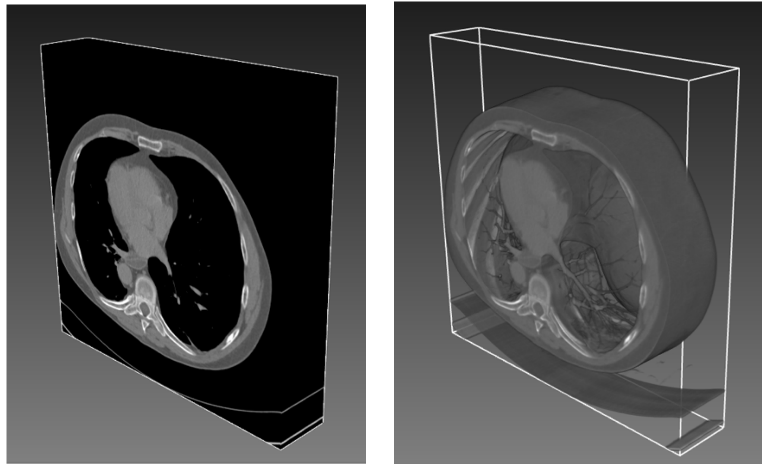


Figure 39: Visualizing the surface data (left) versus volume rendering to see the internal data (right).

المنارة للاستشارات
www.manaraa.com

There are four main volume rendering techniques, 1) image splatting [13–16], 2) shear warp [18,19], 3) texture slicing [20,21], and 4) raycasting [22,23]. Each volume rendering technique implements its own rendering pipeline, which defines the steps taken by the computer to translate the volume into a 3D representation. The application type and volume data being visualized determine the best technique to use. Of all these methods, raycasting produces the most accurate visual representation, but at a high computational cost. Raycasting is a direct volume rendering technique that involves casting linear segments (i.e. “rays”) from each pixel in the frame buffer through the volume in the view direction [14]. Points along the path of the ray that intersect with the volume are sampled and composited together to generate the final pixel color.

6.2.3 Volume Raycasting Pipeline

The volume raycasting pipeline used in this research consists of four steps, 1) Resampling, 2) Classification, 3) Coloring, and 4) Compositing [11]. These four steps make up the core of the volume rendering pipeline. Additional steps can be added to the pipeline to improve photorealism, such as gradient computation and shading. However, medical imaging is typically more concerned with providing contrast in the data to help identify physiological objects than photorealism. Thus, the steps chosen for this research are typical of basic volume renderers currently available.

6.2.3.1 Resampling

Resampling measures the voxel values at different points in three-dimensional space within the volume. For raycasting, an imaginary ray is cast into the volume from each screen pixel and samples are taken along the ray at evenly spaced intervals. Figure 33 shows a 2D representation of raycasting where the person’s eye is on the left

المنارة للاستشارات

www.manaraa.com

looking at the computer screen represented by the black line, as rays are cast from each pixel into the gray volume.

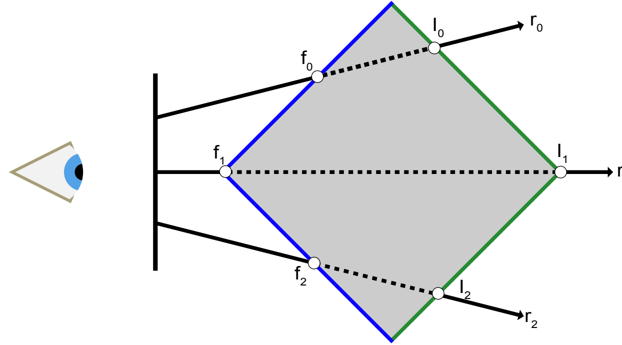


Figure 40: 2D example of raycasting.

Determining the bounds of the volume before casting the rays cuts down on the total number of samples taken by the ray and thus speeds up rendering. The bounds of the volume were found by sending imaginary rays, r_i, from each pixel through the scene in the viewing direction. Each ray's first intersection with the volume represents the starting point, the blue boundary f_i, and the last intersection with the volume represents the end point, the green boundary l_i. Rays that do not intersect with the volume can be ignored in the rendering process and set to the background image color.

The resampling process starts the ray sampling at the starting point, f_i, and continues at specified intervals until reaching the end point, l_i. The sampling points are represented as a single dash of the dashed line in Figure 33**.** The sampling interval used in raycasting can be increased or decreased based on the implementation's goals. Decreasing the sampling size results in higher quality images, but lower rendering speeds. Sampling intervals greater than the width of a single voxel are not recommended due to the tendency to miss smaller anatomical features. Two sampling

المنارة للاستشارات
www.manaraa.com

intervals were tested in this research. The first was set to the width of a single voxel and the second was twice the width of a voxel. These values were chosen to test rendering performance based on sample size.

During raycasting, it is rare for the sampling points along the ray to fall directly on a voxel's vertex. More commonly, the sampling point falls within the voxel. Therefore, interpolation methods must be used to determine the appropriate value at that point in 3D space. Trilinear interpolation was chosen for this research due to its reasonable visual quality and minimal computational requirements [27].

6.2.3.2 Classification

Classification takes each sample from resampling and determines whether it should be included in the final image. It does this by mapping the sample's value to an opacity between zero and one. The opacity determines how much of that voxel's data will be included in the final image with zero being completely transparent and one being completely opaque. The mapping between the sample's value and the opacity is known as an opacity transfer function [50,54]. Creating an opacity transfer function can be very complex depending on the type of data being viewed. The opacity transfer function used in this implementation is a normal distribution with the high and low values equating to completely transparent, zero, and the median equating to completely opaque, one [55].

6.2.3.3 Coloring

Coloring takes each sample and applies a color based on its value using transfer functions. The purpose of coloring is providing contrast within the data to help identify features. Creating coloring schemes that effectively provide contrast is a challenge with traditional data visualization. This challenge is even more difficult with medical imaging because each voxel has a single value that must be translated into a multi-component

المنارة للاستشارات
www.manaraa.com

color, such as red, green, and blue. The mapping of a single value to a color component is typically accomplished by using different 1D color transfer functions (i.e. color lookup tables [177]) for each color component. Different organizations and institutions, such as the National Institute of Health (NIH), create their own color transfer functions for different types of data. This research used some of these institutional color transfer functions.

6.2.3.4 Compositing

Compositing takes all the values sampled by the ray and combines them into a single color to be used for that pixel. The most common compositing method for raycasting is front-to-back compositing as described in Equation 1, where the final intensity value, $I(x,y)$, for each ray is a sum of the sample point intensities, I_i, multiplied by all the transparencies $(1-\alpha j)$ encountered along the ray.

$$I(x,y) = \sum_{i=0}^{n} I_i \prod_{j=0}^{i-1} (1-\alpha_j) \qquad 1)$$

Each sample's intensity, I_i, is a combination of an opacity, α_i, and a color, C_i, as shown in Equation 2. The color and opacity values are obtained from the Classification and Coloring steps through the transfer functions. The higher the opacity, the more intense the color contribution is to the final pixel color, $I(x,y)$.

$$I_i = C_i \times \alpha_i \qquad 2)$$

Put another way, each sample, I_i, can be thought of as a pane of glass with an opacity, α_i, and a color, C_i. The higher the opacity of the first pane of glass, the less of the subsequent panes can be seen. A completely opaque first pane of glass would

المنارة للاستشارات
www.manaraa.com

prevent any other panes from being seen. A 75% opaque first pane would allow only 25% of the color of the second pane of glass to be seen. Front-to-back compositing works the same way, where the opacity of the first samples impacting how much the color from the subsequent samples can be seen. Similarly, once the cumulative opacity reaches 1.0, there is no need to continue compositing because any additional samples would not affect the final pixel color. This method is referred to as early ray termination and is unique to front-to-back compositing.

6.2.3.5 Raycasting Optimizations

The large computational overhead associated with volume raycasting has created many different areas of research aimed at optimizing rendering speeds. One of the most common methods was described in the previous section, early ray termination (also known as adaptive termination) [91]. This concept was originally proposed by Whitted [92] as an adaptively terminating raytracing algorithm. Levoy [93] further developed this concept by terminating the ray traversal when the accumulated opacity reaches a user-specified level. Levoy's method is what was implemented in this research.

Limited graphics memory is another significant challenge with raycasting. Traditional 3D MRI datasets can be on the order of 512x512 pixels per slice with 250 to 1000 slices. Storing a single 512 x 512 slice of data using a 3 byte per pixel storage scheme (e.g., JPEG) equates to 0.78 megabytes (MB) per slice. This results in datasets between 195 to 786 MB per dataset on disk. When volume raycasting, it is common to store the volume as a 32-bit per pixel 3D texture for the graphics card, resulting in a required texture memory of 262 MB to 1.05 gigabytes (GB). A dedicated graphics card is typically necessary to handle this amount of texture memory effectively.

المنارة للاستشارات
www.manaraa.com

6.2.4 4D Volume Raycasting

Moving from 3D to 4D volume raycasting to accommodate functional data presents its own sets of problems and limitations. 3D volume rendering deals with a single set of static volumetric data. Anatomical features are caught in a single position and represented in that “frozen” state. 4D rendering looks are volumetric data over time (i.e. per a defined time step). Therefore, the amount of data associated with a 4D volume increases linearly with the number of time steps. Using the previously discussed 3D dataset as an example, the same 195 to 786 MB volume recorded for 100 time steps would increase the total memory to 19.5 to 78.6 GB. Thus, methods are required to handle this increase in data to allow rendering at interactive speeds. Decomposing the original dataset into two individual datasets is the most common method for dealing with the increase in data. The first dataset is a high-resolution scan containing all the structural data that does not change over time. Using an fMRI of the brain as an example, this would equate to the structure of the skull and brain tissue. The second dataset is a set of lower resolution scans capturing changes. In the fMRI example, this would be the brain activity as represented by BOLD.

Dividing the fMRI data into two datasets reduces the amount of volumetric data changing each frame, but requires rendering both the structural and functional volumes at the same time. Multi-volume rendering is required whenever there are two or more volumes in the scene at the same time. The general multi-volume rendering problem allows volumes to move around independently of each other, causing issues with depth sorting of data. When looking at fMRI brain activity data, the general multi-volume rendering problem can be simplified with the assumption that the volumes never move independently. The brain activity volume should always be correctly aligned with the structural volume of the brain. Researchers have used this assumption to create

المنارة للاستشارات
www.manaraa.com

innovative methods of multi-volume rendering that work well for this type of fMRI data [137–139]. With the volumes properly aligned in 3D space, the ray can sample once from each volume at every sampling point along the ray.

There are two different methods of combining the samples from both volumes together. The first is One Property per Point (OPP), where a single volume's sample is selected to be the final value at that sample point [139]. This method is useful when certain characteristics are more important than others (e.g., EEG brain activity would be more important to visualize than the skull).

The second method is Multiple Properties per Point (MPP), where samples from each volume are combined together into a single value for that point [125,126]. MPP can be used to mix volume samples at various points in the rendering pipeline. Data mixing can occur in the resampling, coloring, or image stages depending on the desired output [137]. Manssour, et al., proposed a method of combining the volumes using a weighting function after the sample's color was determined [138]. Wilson, et al., experimented with mixing data in three different ways: 1) chose one volume's value as the only value, 2) use a weighted function to combine all the values, or 3) use a single volume value each for the red, green, and blue color channels [131]. MPP provides greater flexibility and can handle more complex visualization strategies than OPP allows. However, the more complex visualization strategies can become too complex to decipher if too many volumes are mixed or if the data mixing is not done in a consistent and intuitive way.

6.2.5 Functional Imaging Tools

While there have been some 3D volume rendering tools available for a while, the move to 4D has been slower. There are only a handful of 4D volume rendering tools on the market and many are proprietary, expensive, and do not support raycasting.

المنارة للاستشارات
www.manaraa.com

Siemens' Syngo [186] and Amira [187] are popular commercial volume rendering tools for medical data. Syngo is a Windows tool specifically designed to visualize 4D fMRI data. Amira is designed for visualizing 3D and 4D micro-CT, PET, and Ultrasound data. Amira is one of the few tools that claims support for virtual reality CAVE systems and stereo rendering. However, both of these features are not easily enabled out of the box according to documentation.

There are three free and open source 4D volume rendering tools available, including OsiriX [188,189], MeVisLab [190], and Vaa3D [166,167]. OsiriX is a FDA approved volume renderer that supports MacOS and iOS platforms. OsiriX does not support NIfTI data, which is more common for fMRI than the DICOM standard, and the mobile platform does not support 4D rendering or volume raycasting. MeVisLab provides cross-platform support for Windows, MacOS, and Linux desktop PCs, and supports 4D MRI visualization of fluid flow, but lacks support for fMRI data [165]. Vaa3D is a cross-platform volume renderer that uses iso-surface rendering to visualize 3D and 4D volume data [166,167]. None of the open source tools currently provide any support for virtual reality systems or stereoscopic visualization.

There currently exists only one mobile application on the iOS App Store that allows volume rendering of 4D fMRI data, NeuroPub [182]. However, this implementation is extremely limited with specific data requirements such as requiring 64 x 64 pixel slices at 32-bits each. The visualization method appears to use (since source code or a detailed code description are not publicly available) a single surface model of the brain and map the functional data onto this model, meaning the tool does not use raycasting or the actual structural data of the patient in the visualization.

The lack of tools available across desktop, virtual reality, and mobile devices indicates there are challenges associated with building 4D real-time volume raycasting

المنارة للاستشارات
www.manaraa.com

tools. This is extremely evident in mobile where there are currently no tools that use raycasting for 3D data, much less 4D. Previous research has shown 3D monoscopic (3D without glasses) and 3D stereoscopic (3D requiring glasses) displays to increase users' understanding of shapes [191,192]. This was proven beneficial in mammogram lesion detection where new lesions were detected with stereo mammograms that were not detected in traditional film [193]. Increasing the tools available to medical professionals that allow the 3D monoscopic and 3D stereoscopic viewing of medical data will benefit medical diagnosis and treatment. Therefore, this research will look at the feasibility and challenges of building a real-time 4D volume raycaster for desktop, virtual reality, and mobile platforms.

6.3 Materials and Methods

Determining the feasibility of implementing a real-time 4D volume raycaster across desktop, virtual reality, and mobile platforms has many challenges. Differences in libraries, coding languages, and hardware across these platforms make it difficult to develop a single codebase that will work well for everything. From a developer standpoint, it was important to develop the volume renderer to use the same function calls across all three platforms. This provides consistency from the user perspective, and allows the developer to make the same function calls on every platform, even if the volume renderer is performing different actions within these function calls. It was important to implement the same raycasting pipeline across all three platforms to provide the same visuals and for evaluation. The coding languages and libraries used to build all three prototype applications will be discussed in the following sections, starting with the desktop and virtual reality prototypes.

المنارة للاستشارات

www.manaraa.com

6.3.1 Desktop and Virtual Reality Prototypes

The desktop and virtual reality applications were built using C++ as the base coding language and the basis for all libraries. C++ was chosen due to its wide adoption in building software libraries and it's support across most platforms.

The low level graphics rendering Application Programming Interface (API) is one of the most important decisions for achieving computational efficiency. On desktop computers, the low level rending API choice is typically between DirectX and OpenGL. Both OpenGL and DirectX are generally considered comparable in their performance. However, DirectX is limited to Windows, while OpenGL is cross-platform, supporting Windows, Mac, and Linux. OpenGL was chosen because it provided a common graphics language across both desktop and virtual reality platforms and introduced no loss in performance on Windows hardware.

The low level nature of OpenGL provides developers with many customization options for harnessing every scrap of computational power from the graphic processing unit (GPU). The downside to using OpenGL is that a code optimized for one GPU will not be optimized for a different GPU and may not work at all without changes. It was therefore decided to abstract the complexity of using OpenGL by encapsulating it within another API, OpenSceneGraph. This encapsulation allows the same coding routines to be called across different devices with OpenSceneGraph handling the correct OpenGL calls to achieve optimal results.

Loading the volumetric data generically was a significant challenge. There are multiple possible file types used to store 4D volumetric data, such as DICOM, Analyze/SPM, MINC, AFNI, and NIfTI [31]. Formats like DICOM were designed for storing 3D data, thus when DICOM is used to store 4D volumetric data, there is no standard way of reading it. For fMRI brain scans, NIfTI is the most commonly used file

المنارة للاستشارات
www.manaraa.com

format. NIfTI is a superior choice for storing functional data as it was specifically designed for that purpose. However, not everyone uses the standard in a consistent way, so there are specific challenges with generically inputting data prior to visualization.

Unlike more established formats, like DICOM, there are currently no open source software libraries available to read NIfTI data. Therefore, this research created a basic NIfTI file reader library capable of loading generic fMRI NIfTI files for volume rendering. The library was built using C++ and the C++ Standard Libraries to be both lightweight and cross-platform.

The NIFTI file reader library was set up with a singleton called NIfTILoader as the main class and the starting point for developer interaction. The developer can pass in a list of NIfTI files to NIfTILoader and it will return a list of abstract Volume objects. The NIfTILoader class delegates file processing to the correct classes based on the version number of the NIfTI file. The abstract Volume class is designed as a container for all the information about the NIfTI data including relevant patient information, file header data, and the volume slice data. By default, a Volume object is returned with the volume slice data stored as a C-array. However, the Volume class was designed to be extended to support different types of graphics programming objects. For this research, the Volume class was sub-classed to create an OSGVolume and MTLVolume to store the volume slice data as a set of OpenSceneGraph images and Metal 3D Textures respectively.

The desktop and immersive VR prototypes required using different windowing systems. The desktop prototype used Qt [194] for a windowing system and user interface. Qt is a freely available cross-platform software development kit. OpenSceneGraph's window independence allows it to be rendered directly within an OpenGL widget provided by Qt and displayed within a user interface.

المنارة للاستشارات

www.manaraa.com

The immersive virtual reality prototype required a windowing system that would work on large scale rendering clusters like those seen in CAVE™ platforms. VR Juggler was selected as the VR windowing system because it is open source and has been proven to have superior frame rates when compared to other similar systems [176].

6.3.2 Mobile Prototype

The choice with mobile development is currently between the iOS and Android platforms. There are other mobile operating systems out there, but these two make up the majority of mobile devices. Both systems have their advantages, but iOS was chosen over Android because of their new low level graphics language, Metal, that replaced the previous standard, OpenGL ES.

Metal is a C based shading language that has shown a lower overhead than OpenGL ES that results in, according to Apple documentation, up to 10 times higher frame rates. Given that volumetric raycasting is a computationally expensive process, any additional performance increase is helpful. Metal achieves this by moving as many operations as possible outside of the render loop. One example is the focus on precompiling shaders during application compilation and then swapping them when needed during run-time. This is in stark contrast to OpenGL ES that compiles the shaders from source during run-time. Metal was also introduced with support for the 3D textures that are critical to efficient volumetric raycasting.

Cross-platform libraries such as Qt and game engines like Unity3D were considered for this research because of their support for both iOS and Android. Native libraries are generally better optimized for the specific platform and therefore more computationally efficient. Thus, the decision was made to use as many native tools for

المنارة للاستشارات
www.manaraa.com

the prototype as possible including using Swift 3.0 programming language and Cocoa Touch for the windowing system and user interface elements.

6.3.3 4D Volume Raycasting

The volume raycasting pipeline described earlier needed to be modified to expand the functionality from 3D to 4D volume data. The fMRI brain activity data that was the focus of this research was broken into a high-resolution structural volume and a series of low-resolution functional volumes. Therefore, it was necessary to both expand the raycaster to visualize brain activity over time and to visualize multiple volumes at the same time.

The brain activity in fMRI data is correlated with the increase in oxygenation in areas of the brain. This change in oxygen levels over time is what is stored in the functional volume. There is typically a baseline brain activity scan and then a series of brain activity scans over time. Visualizing the brain activity at a single time step required calculating the difference between the baseline brain activity and the current time step's brain activity. From a programmatic stand point, this required storing both the baseline volume and the current time step volume as a 3D texture. The difference between the two volumes was defined as the activity at that time step.

There is noise associated with fMRI brain activity data that can make it challenging to determining where brain activity is occurring. The amount of noise associated with the functional data necessitated the use of a brain activity threshold. The brain activity threshold worked similar to the classification stage in the rendering pipeline. The activity needed to exceed the threshold before being included in the final visualization. If the magnitude of the activity was below the threshold, the opacity for that sample was set to

المنارة للاستشارات
www.manaraa.com

zero so it would not contribute to the final pixel color. The activity threshold was a user specified value that could be changed interactively through a user interface slider bar.

Obtaining a single cohesive visualization required rendering the structural and functional volumes together. The multi-volume rendering method used assumes that neither volume moves independently of each other. This assumption allows the efficient casting of a single ray and sampling of all three volumes (structural, baseline functional, and current time step functional) at the same location in 3D space. Using this method requires the volumes to be properly aligned and scaled relative to each other at the beginning of the process, as both the structural and functional volumes will have different scaling factors.

The resampling process used a single ray to sample all volumes at the same location in 3D space to obtain a sample for each volume. Each sample went through its own classification and coloring steps before being combined with the other volumes' samples to achieve a single color and opacity for that sample point in 3D space.

The classification and coloring steps differed between samples from the structural or functional volumes. The structural samples went through the same process as if it were a single 3D volume to achieve a color and opacity. The functional samples calculate the difference between the current time step activity and the baseline activity. The difference in these two values represents the magnitude of the brain activity at that point. An opaque red color is multiplied by this magnitude to obtain a final color and opacity for that sample. This results in higher levels of activity being displayed as more red in the final image.

The resulting prototype applications can be seen in Figure 41. The top image is the mobile application, the middle image is the desktop application, and the bottom image is the immersive VR application running the C6.

المنارة للاستشارات

www.manaraa.com

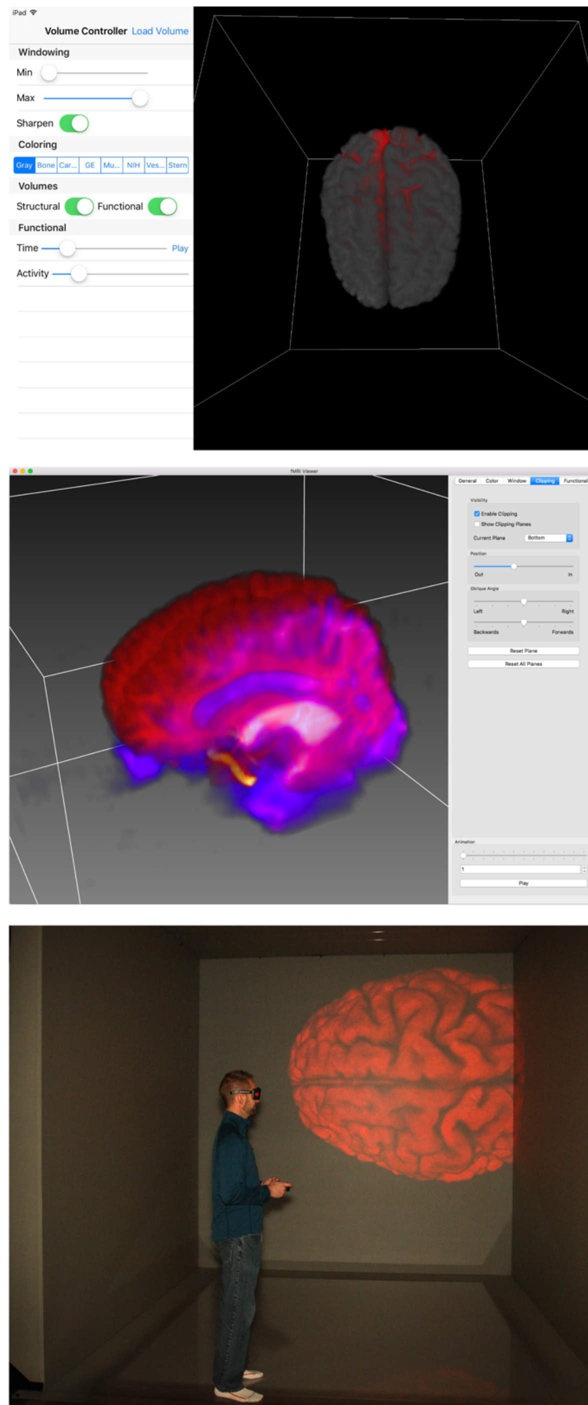


Figure 41: Prototype applications. From top to bottom, Mobile, Desktop, Immersive VR.

المنارة للاستشارات
www.manaraa.com

6.4 Results

The goal of this research was to investigate the feasibility of real-time volumetric raycasting of 3D and 4D fMRI data. Unfortunately, there is no industry wide standard for what qualifies an application as real-time, making it difficult to quantify whether an application is real-time or not. However, previous research performed by Miller identified 0.1 seconds as an acceptable graphical response time after an input from a light pen [1]. Using this value, a frame rate of 10 frames per second (fps) or higher will be considered acceptable as real-time for this research. Application initialization time, including volumetric data loading, was also used as a measure of the usability of the application in everyday medical situations. Medical professionals are not likely to wait extended periods of time (e.g., 5-10 minutes or more) to load a single fMRI dataset in their everyday work flow.

The volume raycast prototypes were tested on three different hardware platforms. The desktop version was evaluated on a 2013 MacBook Pro running OS X 10.12 with a 2.6 GHz Intel Core i7 processor, 16 GB of RAM, and a NVIDIA GeForce GT 750M graphics card. The desktop's performance was used as a baseline to compare the immersive VR and mobile prototypes.

The immersive VR prototype was evaluated using the C6, the world's highest resolution six-side VR CAVE™ located at Iowa State University. Twenty-four Sony 4K projectors achieve 96 million pixels per eye. A 96 node rendering cluster comprised of NVIDIA Quadro 6000 graphics cards is required to feed the 24 projects. The C6 uses the Red Hat Enterprise Linux operating system. An Intersense ultrasonic tracking system is used to track objects like the user's head position. The 96 graphic node cluster provides an excellent testbed for the scalability of the implementation.

المنارة للاستشارات
www.manaraa.com

The mobile prototype was evaluated on a 2016 9.7" iPad Pro running iOS 9.3.4. The iPad had a 64-bit Apple A9X dual core processor with 2 GB of built in memory to power a 2048 x 1536 pixel screen. This hardware was the most powerful iOS device available at the time.

A single representative brain activity fMRI data set was used for testing all three platforms. The dataset consisted of a high-resolution structural volume and a set of low-resolution functional volumes, with each volume representing a different moment in time. The structural volume was comprised of 128 slices at 256 x 256 pixels per slice. The functional data included 126 individual time step volumes with each volume comprised of 24 slices at 64 x 64 pixels per slice. There was a 3 second time interval between individual time steps.

The application initialization time was measured to determine if the wait time for a user would be prohibitively long in a real world situation. The initialization timings include reading the NIfTI files, converting the volumetric data into 3D textures, and setting up the graphics pipeline. Table 7 shows the average application initialization time based on ten trials per platform.

Table 7: Application Initialization Times in Seconds

	Desktop	Mobile	VR
Average	6.90	29.79	17.94
Standard Deviation	0.062	0.268	1.740
Variance	0.004	0.072	3.027

The desktop prototype had the lowest application initialization at an average of 6.9 seconds. The immersive VR application saw more than double the initialization time using almost identical libraries and code at 17.9 seconds. The timing points in all three

المنارة للاستشارات
www.manaraa.com

prototypes were chosen to eliminate measuring any user interface setup time, to keep the comparisons as close as possible. The raw computing power of the VR hardware surpasses that of the desktop hardware, therefore the time difference can be attributed to the synchronization of data across the computer cluster. The mobile platform, as expected, proved the slowest of the three platforms clocking in at over four times that of the desktop at almost 30 seconds. While there are differences in the software stack being used, the large difference is most likely due to the difference in raw computational power.

The frame rates of the prototype applications were evaluated with the 10 frames per second goal for real-time interaction. The prototypes were tested visualizing only the structural data, only the functional data, or both (combined). The functional data could also be visualized statically (Single time step), or by animating through the time steps (All time steps). This resulted in five different testing conditions for the different platforms. The VR platform was tested with the volume raycaster running on all 96 nodes of the CAVE™ system as well as running on a single standalone node. Each condition consisted of five runs measuring 100 consecutive frames. These conditions were then ran while simulating no user interaction (No Rotation) and while simulating a user rotating the volume around the vertical axis once every 10 seconds (Rotating). All devices were rebooted between runs. Those frames were then averaged for the final frame rates shown in Table 8.

المنارة للاستشارات

www.manaraa.com

Table 8: Frame Rates for Desktop, VR, and Mobile.

		Structural	Functional		Combined	
			Single time step	All time steps	Single time step	All time steps
No Rotation	Desktop	46.6	56.1	57.6	35.9	36.0
	VR (1 node)	59.9	59.9	59.9	29.9	29.9
	VR	20.0	15.8	15.4	10.3	10.3
	Mobile	17.9	51.5	9.4	11.9	9.3
Rotating	Desktop	50.0	59.9	59.2	33.0	34.9
	VR (1 node)	59.9	60.0	60.0	29.9	30.0
	VR	16.3	10.7	10.8	7.1	7.2
	Mobile	18.8	47.8	9.7	11.0	8.2

The single computer configurations of the desktop and the single VR node provided the best performance in frame rates. Both platforms were able to achieve the desired 10 frames per seconds across all five testing conditions. The functional data alone was rendered at approximately 60 frames per second for both the Single time step and All time steps conditions. Combining the data saw the lowest frame rates at 33 frames per second for the desktop and 29 frames per second for the single VR node. Still above the target goal for real-time interaction.

The VR and Mobile platforms were not consistently above the desired 10 frames per second, but were very close across all conditions. The VR platform failed to meet the real-time goal when visualizing the combined condition and around 7 frames per second. The mobile platform failed to meet the real-time goal when animating through the time steps at just under 10 frames per second.

The same platforms and conditions were tested using a sampling step size of twice the width of a voxel and simulating no user interaction. Using a larger sampling step size will increase frame rates at the cost of a less accurate representation. The larger sampling size produces a good macro visual, but is not recommended for medical

المنارة للاستشارات
www.manaraa.com

diagnosis. The question then becomes, if the increase in frame rates is significant enough to mitigate the lower visual resolution for some situation.

Table 9: Frame Rates for Desktop, VR, and Mobile with double the sampling step size.

		Structural	Functional		Combined	
			Single time step	All time steps	Single time step	All time steps
No Rotation	Desktop	59.675	59.952	60.010	59.938	59.923
	VR (1 node)	59.839	59.949	59.952	59.956	59.951
	VR	30.006	19.997	24.682	20.004	19.933
	Mobile	41.201	60.030	9.641	22.603	9.451

The results of doubling the sampling step size is roughly a doubling of all frame rates across all platforms. The doubling of the frame rate is to be expected given the number of times the volume is sampled is cut in half. The only exception is the All time steps condition for the mobile device that stayed roughly constant. This may indicate that there is a bottleneck in the mobile render loop when it comes to switching textures. However, these two conditions are the only ones not achieving the real-time interaction goal.

6.5 Discussion

Application initialization and data loading results show that all platforms are capable of loading the 4D functional data in an acceptable time period. The desktop application was the fastest at 6.9 second and the mobile application was the longest at 29.8 seconds. The VR application's initialization time was over twice that of the desktop application at 17.9 seconds. The additional initialization time can be attributed to synchronizing the data across the 96 node cluster.

The mobile application initialization times were not prohibitive to a user at 30 seconds, but they are not ideal. The decision was made to preload all the volume data

المنارة للاستشارات
www.manaraa.com

into 3D textures in the initialization phase because generating those textures is a time consuming process. This allowed the 3D textures to be swapped at run time for the functional data instead of generating the 3D textures at run time as needed. In the initial stages of prototype development, preloading the textures doubled the frame rates across the board when visualizing functional data and was therefore considered an acceptable tradeoff. It is important to consider that preloading the textures increases the memory footprint of the application, which is not ideal on a mobile device that is already memory limited. A large enough medical data set would cause the application to run out of memory and crash.

The frame rate results using the ideal sampling step size indicates that immersive VR and mobile devices are not yet capable of consistently performing 4D functional volume raycasting in real-time. However, the functional raycasting conditions for both VR and mobile were at worst 7.1 frames per second, which is approaching the real-time frame rate goal. It is also important to note that all platforms proved capable of 3D volume raycasting at real-time frame rates.

Doubling the sampling step size improved frame rate performance by roughly double for the desktop and VR platforms. The mobile All time steps conditions were the only conditions unable to achieve the desired frame rates. However, these two conditions are close to the 10 frames per second at 9.6 and 9.4 fps respectively. Therefore, doubling the sampling step size would recommended for any situation requiring looking at gross anatomy.

There are several techniques that could be applied to improve frame rates that were not explored in this research. One being to use octrees [93,94] to provide an additional efficiency of empty space skipping, where areas of the volume where no data exists could be completely skipped in the resampling stage. Another option would be to

المنارة للاستشارات
www.manaraa.com

reduce the number of rays being cast into the volume, which would have a similar effect to increasing the sampling step size. This could be accomplished by casting one quarter of the current rays and then scaling up the resulting texture to fill the screen. This would reduce the visual quality but would increase frame rates.

The frame rates on the immersive VR system could be increased by reducing the size of the compute cluster. The decrease in performance from the desktop baseline is primarily due to the need to synchronize data and frame swapping across multiple computers. Decreasing the size of the cluster would decrease the need for synchronization. Moving to a single computer head mounted display (HMD) system would also eliminate synchronization and improve frame rates closer to the desktop results.

6.6 Conclusions

This research focused on the feasibility of implementing a real-time 4D volume raycasting algorithm on desktop, virtual reality, and mobile platforms. The frame rate goal of 10 frames per second was unanimously achieved when visualizing 3D structural data. The real-time functional volume raycasting was consistently achieved on the desktop and single VR node platforms. However, both the immersive VR and mobile platforms are approaching the ability to achieve the desired frame rates consistently, but they are not quite there yet.

All the necessary hardware and software technology is now available in commercial devices to perform volume raycasting across these dissimilar computational platforms. The historical trend of increases in computational power would indicate that consistent real-time functional volume raycasting across all these platforms will be possible in the near future.

المنارة للاستشارات

www.manaraa.com

There are still several techniques that could be implemented to improve frame rates, like empty space skipping through the use of octree data structures. Another possible optimization would be to calculate the difference in brain activity on application initialization. This would increase the application initialization time, but would reduce the number of textures passed into the shader and eliminate a few graphics operations currently needed to perform this calculation at run time.

المنارة للاستشارات
www.manaraa.com

CHAPTER 7: SUMMARY AND FUTURE WORK

7.1 Summary

This research investigated the feasibility of creating a generic software capability to perform real-time 4D volume rendering on desktop, mobile, and immersive virtual reality platforms. Three different prototype applications were built across desktop, mobile, and immersive virtual reality platforms. Each used programming languages and libraries native to the platform to implement the same volume raycasting pipeline. The goal was the achieve real-time performance while providing the same visualization features across all three platforms.

Achieving this goal required addressing four different research issues. The first research issue was to explore the feasibility of a generic NIfTI data input capability on desktop, VR, and mobile device platforms. The NIfTI Loader, was developed to accomplish this. The NIfTI Loader takes in generic NIfTI files and return a C-based data array containing the volume data along with a header object defining the parameters of the volume. The design allows developers to extend the functionality of the classes to return their own datatypes, such as a Metal 3D texture or an OSG Image instead of the default C-based array.

The second research issue was assessing the feasibility of displaying functional medical data across desktop, VR, and mobile device platforms. The use of OpenGL and Metal shading languages were used to build the same volume raycasting pipeline across all three platforms. The results indicate that all three platforms are capable of consistently achieving real-time interaction for 3D medical data and are close to achieving consistent real-time performance on the VR and mobile devices. Improvements in computational power in the next few years should consistently achieve real-time interaction.

www.manaraa.com

The third research issue was the real-time visualization of a high-resolution structural volume and a low-resolution functional volume at the same time. Multi-volume rendering is a complex issue with many possible solutions. This research simplified the problem by assuming the functional brain activity volume and the structural brain volume never moved independently. The volumes could then be aligned and scaled relative to each other in 3D space. This allowed the raycasting implementation to sample both volumes at the same point in 3D space. A custom compositing method was then used to mix the structural and functional data together into a cohesive visualization.

The fourth research issue was to develop a GPU-based 4D volume raycaster for mobile devices supported by the iOS platform. The Metal shading language was used to build the iOS based volume raycasting implementation. Previous raycasting research achieved less than one frame per second for a single 3D volume [2]. This implementation consistently achieved real-time interaction for 3D volumes with better than ten frames per second. The 4D functional data was close to achieving consistent real-time interaction with frame rates between two and forty frames per second. The frame rate performance was dependent on the size of the dataset and whether the raycaster was rendering just functional data or a combination of the structural and functional data.

Based on this research, it can be concluded that it is feasible to create a generic software capability to perform real-time 4D volume rendering on desktop, mobile, and immersive virtual reality platforms. All three platforms were able to consistently achieve real-time frame rates when visualizing 3D volumes at frame rates between 18 and 50 frames per second. The ability to raycast 3D volumes across all three platforms is something that was not previously possible. When raycasting 4D volumes, the desktop platform was capable of consistently achieving real-time frame rates. The mobile and

المنارة للاستشارات
www.manaraa.com

immersive virtual reality platforms were able to achieve real-time frame rates most of the time, but it was not consistent. However, with the rates at which computational power increases, it is possible to claim that consistent real-time interaction across all three platforms is not far off.

These results indicate that it is feasible to create a generic software capability to perform real-time 4D volume rendering on desktop, mobile, and immersive virtual reality platforms. All prototypes in this research used the same basic raycasting pipeline to achieve the same volume raycasting visualizations, while using very different rendering methods on the backend. For example, OpenGL was used for both the desktop and immersive VR implementations while Metal was used for the mobile implementation.

7.2 Future Work

This research has proven that it is feasible to create a generic software capability to perform real-time 4D volume raycasting, but there is still more work that can be done. The immersive virtual reality and mobile platforms are still not consistently achieving real-time frame rates and therefore, methods to improve rendering efficiency should be investigated. Empty space skipping techniques using octrees, kd-trees, or time space partitioning (TSP) trees are logical first steps. These methods have shown the ability to improve volume rendering speeds by eliminating rendering passes in areas with no data. This can be beneficial in medical data where large parts of the volume can be the air around the body.

Another efficiency improvement can be achieved by calculating the difference in functional brain activity and storing that in a set of 3D textures upon application initialization. A 3D texture would be required to store the difference between the baseline activity and the activity for each time step. This would eliminate one texture lookup call

المنارة للاستشارات

www.manaraa.com

and the calculation for the activity difference from the shader program at each sampling point. While this might not seem like much, these two operations are performed hundreds of times per frame during ray casting in the resampling step. Thus, eliminating these calculations would speed up the render.

Another area of future work would be to expand the different platforms being tested. Currently, there are several other sizes and types of mobile devices beyond a single 9.7” iPad Pro, specifically smart phones and Android based mobile devices. Testing on different hardware and software platforms will be necessary to ensure this method works consistently across a range of mobile devices. The immersive virtual reality platform used in this research was an excellent testbed because of its large computing cluster and networking complexity. However, there are other types of virtual reality systems commercially available, including low-cost head mounted displays (HMDs). Commercial companies are investing significant resources in these low-cost HMDs [34]. A different rendering backend than the current VR prototype would be necessary for these HMDs. They divide the single screen in half and render each half from that eye’s position. This allows a double buffer graphics card to be used versus the immersive VR systmes requiring a quad buffer to generate stereoscopic visuals.

Another issue with fMRI data is the noise inherent in the data. A threshold was used in this research to filter the noise from the brain activity. However, there are better filtering methods for noise out there that could improve the visual quality and accuracy of the fMRI render. The challenge is in using a method that is computationally efficient enough to maintain real-time frame rates.

The final challenge with this work will be actually building a cross-platform API that will use the lessons learned here to improve the adoption of fMRI volume rendering. It was stated earlier in this research, that cutting-edge volume rendering advances

المنارة للاستشارات
www.manaraa.com

typically never make it into commercial tools to be used by developers and taken advantage of by medical professionals. This research has proven that it is feasible to create a generic 4D volume raycasting capability across dissimilar platforms. It is possible to incorporate this research into building a single API for cross-platform volume raycasting. The API could provide a unified interface for developers to use regardless of the hardware and software platforms they are using. Developers could make the same function call on any device and achieve the same results. The next step is designing and building that API so developers and, ultimately, medical professionals and patients can benefit from this game changing technology.

المنارة للاستشارات
www.manaraa.com

ACKNOWLEDGEMENTS

I would like to thank everyone who has helped me over the course of my time in graduate school. First, I would like to thank my parents Scott and Gerilyn who have provided me with love and support my entire life. You are both amazing role models and I would not be where I am without your guidance. Thank you to my sister Jamie who has constantly challenged me to be better and provided me with the encouragement to get there. Thank you to the rest of my family and all my friends for the support, encouragement, and patience. You all helped me to achieve this goal in one small way or another.

I would like to thank Eliot Winer, my major professor, who has continued to guide me as a researcher and future professional. I have accomplished far more outside of this research because of the opportunities you presented me. Thank you to my committee members Stephen Gilbert, Phillip Jones, James Oliver, and Rafael Radkowski for your encouragement and support of my research.

A huge thank you to the staff at VRAC for always being willing to drop what they were doing to help me out. Finally, a thank you to the members of my research group, past and present. You are all amazing individuals that made coming to work a blast.

المنارة للاستشارات

www.manaraa.com

REFERENCES

[1] R.B. Miller, Response time in man-computer conversational transactions, Proc. December 9-11, 1968, Fall Jt. Comput. Conf. Part I. (1968) 267–277. doi:10.1145/1476589.1476628.

[2] C.J. Noon, A Volume Rendering Engine for Desktops , Laptops , Mobile Devices and Immersive Virtual Reality Systems using GPU-Based Volume Raycasting by, Iowa State University, 2012.

[3] W. Raghupathi, V. Raghupathi, Big data analytics in healthcare: promise and potential, Heal. Inf. Sci. Syst. 2 (2014) 3. doi:10.1186/2047-2501-2-3.

[4] W.C. Rontgen, On a New Kind of Rays, Science (80-.). 3 (1896) 227–231. doi:10.1126/science.3.59.227.

[5] D.J. Goodenough, K.E. Weaver, Overview of Computed Tomography, IEEE Trans. Nucl. Sci. 26 (1979) 1661–1667. doi:10.1109/TNS.1979.4330458.

[6] P.A. Bottomley, Nuclear magnetic resonance: Beyond physical imaging: A powerful new diagnostic tool that uses magnetic fields and radio waves creates pictures of the body's internal chemistry, IEEE Spectr. 20 (1983) 32–38. doi:10.1109/MSPEC.1983.6369002.

[7] K.K. Shung, M.B. Smith, B. Tsui, Principles of Medical Imaging, Academic Press, Inc., San Diego, CA, 1992.

[8] A. Helgeland, T. Elboth, Hurricane Visualization Using Anisotropic Diffusion and Volume Rendering, in: 5th Annu. Gather. High Perform. Comput., 2005: pp. 3–4.

[9] R. a. Ketcham, W.D. Carlson, Acquisition, optimization and interpretation of X-ray computed tomographic imagery: applications to the geosciences, Comput. Geosci. 27 (2001) 381–400. doi:10.1016/S0098-3004(00)00116-3.

[10] K. Seo, J. Frank, Attachment of Escherichia coli O157: H7 to lettuce leaf surface and bacterial viability in response to chlorine treatment as demonstrated by using confocal scanning laser microscopy, J. Food Prot. 62 (1999) 3–9. http://www.ingentaconnect.com/content/iafp/jfp/1999/00000062/00000001/art00001 (accessed October 7, 2014).

[11] A. Kaufman, K. Mueller, Overview of volume rendering, in: Vis. Handb., 1st ed., Elsevier Inc., 2005: pp. 127–174.

[12] W.E. Lorensen, H.E. Cline, Marching cubes: A high resolution 3D surface construction algorithm, ACM SIGGRAPH Comput. Graph. 21 (1987) 163–169. doi:10.1145/37402.37422.

[13] L. Westover, Footprint evaluation for volume rendering, ACM SIGGRAPH Comput. Graph. 24 (1990) 367–376. doi:10.1145/97880.97919.

[14] M. Botsch, L. Kobbelt, High-quality point-based rendering on modern GPUs, 11th Pacific Conf. onComputer Graph. Appl. 2003. Proceedings. (2003). doi:10.1109/PCCGA.2003.1238275.

[15] N. Neophytou, K. Mueller, GPU accelerated image aligned splatting, Fourth Int. Work. Vol. Graph. 2005. (2005) 197–242. doi:10.1109/VG.2005.194115.

[16] T. Moller, R. Machiraju, K. Mueller, R. Yagel, Classification and local error estimation of interpolation and derivative filters for volume rendering, in: Proc. 1996 Symp. Vol. Vis., San Fransisco, CA, 1996: pp. 71–78. doi:10.1109/SVV.1996.558045.

[17] K. Mueller, T. Moller, R. Crawlis, Splatting without the blur, Proc. Vis. '99. (1999) 1–9. doi:10.1109/VISUAL.1999.809909.

[18] H. Pfister, J. Hardenbergh, J. Knittel, H. Lauer, L. Seiler, The VolumePro Real-Time Ray-casting System, in: Siggraph 1999, Comput. Graph. Proceedings, 1999: pp. 251–260. doi:10.1145/311535.311563.

[19] Y. Wu, V. Bhatia, H. Lauer, L. Seiler, Shear-image order ray casting volume rendering, in: Proc. 2003 Symp. Interact. 3D Graph. - SI3D '03, 2003: pp. 152–162. doi:10.1145/641508.641510.

المنارة للاستشارات
www.manaraa.com

[20] C. Rezk-Salama, K. Engel, M. Bauer, G. Greiner, T. Ertl, Interactive volume on standard PC graphics hardware using multi-textures and multi-stage rasterization, in: Proc. ACM SIGGRAPH/EUROGRAPHICS Work. Graph. Hardw., 2000: pp. 109–118. doi:10.1145/346876.348238.
[21] K. Engel, M. Kraus, T. Ertl, High-quality pre-integrated volume rendering using hardware-accelerated pixel shading, in: Proc. ACM SIGGRAPH/EUROGRAPHICS Work. Graph. Hardw., 2001: pp. 9–16. doi:10.1145/383507.383515.
[22] H. Pfister, Hardware-Accelerated Volume Rendering, in: Vis. Handb., Elsevier, 2005: pp. 229–258. doi:10.1016/B978-012387582-2/50013-7.
[23] D. Weiskopf, GPU-Based Interactive Visualization Techniques, Springer Berlin Heidelberg, 2006. doi:10.1007/978-3-540-33263-3.
[24] R. Westermann, T. Ertl, Efficiently using graphics hardware in volume rendering applications, in: Proc. 25th Annu. Conf. Comput. Graph. Interact. Tech. - SIGGRAPH '98, 1998: pp. 169–177. doi:10.1145/280814.280860.
[25] D. Robertson, W. Johnston, W. Nip, Virtual frog dissection: interactive 3D graphics via the Web, Comput. Networks ISDN Syst. 28 (1995) 155–160. doi:10.1016/0169-7552(95)00117-6.
[26] R.J. Jansen, M. Poulus, W. Taconis, J. Stoker, High-resolution spiral computed tomography with multiplanar reformatting, 3D surface- and volume rendering: A non-destructive method to visualize ancient Egyptian mummification techniques, Comput. Med. Imaging Graph. 26 (2002) 211–216. doi:10.1016/S0895-6111(02)00015-0.
[27] B. Lichtenbelt, R. Crane, S. Naqvi, Introduction to volume rendering, Prentice-Hall, Inc., Upper Saddle River, NJ, 1998.
[28] F. Di Salle, E. Formisano, D.E. Linden, R. Goebel, S. Bonavita, a Pepino, et al., Exploring brain function with magnetic resonance imaging., Eur. J. Radiol. 30 (1999) 84–94. http://www.ncbi.nlm.nih.gov/pubmed/10401589.
[29] P. Bandettini, Functional MRI today., Int. J. Psychophysiol. 63 (2007) 138–45. doi:10.1016/j.ijpsycho.2006.03.016.
[30] B.R. Rosen, R.L. Savoy, fMRI at 20: has it changed the world?, Neuroimage. 62 (2012) 1316–24. doi:10.1016/j.neuroimage.2012.03.004.
[31] Neuroimaging Informatics Technology Initiative (NIfTI), (2011). http://nifti.nimh.nih.gov.
[32] ANALYZE 7.5 File Format, Time. (20015). https://rportal.mayo.edu/bir/.
[33] M. DeGusta, Are Smart Phones Spreading Faster than Any Technology in Human History?, MIT Technol. Rev. (2012).
[34] B. Solomon, Facebook Buys Oculus, Virtual Reality Gaming Startup, For $2 Billion, Forbes. (2014). http://www.forbes.com/sites/briansolomon/2014/03/25/facebook-buys-oculus-virtual-reality-gaming-startup-for-2-billion/ (accessed January 1, 2015).
[35] D. Shreiner, OpenGL Programming Guide, 7th ed., Addison-Wesley Professional, 2009.
[36] S. Roettger, S. Guthe, D. Weiskopf, T. Ertl, W. Strasser, Smart Hardware-Accelerated Volume Rendering, in: IEEE TCVG Symp. Vis., 2003: pp. 231–238.
[37] W.M. Hsu, Segmented ray casting for data parallel volume rendering, in: Proc. 1993 IEEE Parallel Render. Symp., 1993: pp. 7–14. doi:10.1109/PRS.1993.586079.
[38] K. Ma, Parallel volume ray-casting for unstructured-grid data on distributed-memory architectures, in: IEEE Symp. Parallel Render. 1995, 1995: pp. 23–30. doi:10.1145/218327.218333.
[39] K.-L.M.K.-L. Ma, T.W. Crockett, A scalable parallel cell-projection volume rendering algorithm for three-dimensional unstructured data, in: Proc. IEEE Symp. Parallel Render., 1997: pp. 95–104. doi:10.1109/PRS.1997.628300.
[40] J. Sanders, E. Kandrot, CUDA by Example, 1st ed., Addison-Wesley, 2010.
[41] J.D. Owens, D. Luebke, N. Govindaraju, M. Harris, J. Krüger, A.E. Lefohn, et al., A survey of general-purpose computation on graphics hardware, Comput. Graph. Forum. 26 (2007) 80–113. doi:10.1111/j.1467-8659.2007.01012.x.

المنارة للاستشارات
www.manaraa.com

[42] Y. Heng, L. Gu, GPU-based Volume Rendering for Medical Image Visualization., in: Conf. Proc. IEEE Eng. Med. Biol. Soc., 2005: pp. 5145–5148. doi:10.1109/IEMBS.2005.1615635.
[43] OpenCL, (2015). https://www.khronos.org/opencl/.
[44] CUDA, (2015). http://www.nvidia.com/object/cuda_home_new.html.
[45] GLSL, (2015). https://www.opengl.org/documentation/glsl/.
[46] P.T. Fox, M.E. Raichle, Focal physiological uncoupling of cerebral blood flow and oxidative metabolism during somatosensory stimulation in human subjects., Proc. Natl. Acad. Sci. U. S. A. 83 (1986) 1140–1144. doi:10.1073/pnas.83.4.1140.
[47] D.L. Pham, C. Xu, J.L. Prince, Current methods in medical image segmentation., Annu. Rev. Biomed. Eng. 2 (2000) 315–337. doi:10.1146/annurev.bioeng.2.1.315.
[48] T. McInerney, D. Terzopoulos, Deformable models in medical image analysis: a survey., Med. Image Anal. 1 (1996) 91–108. doi:10.1016/S1361-8415(96)80007-7.
[49] J.L. Foo, A Framework for Tumor Segmentation and Interactive Immersive Visualization of Medical Image Data for Surgical Planning, Iowa State University, 2008.
[50] M. Levoy, Display of surfaces from volume data, IEEE Comput. Graph. Appl. 8 (1988) 29–37. doi:10.1109/38.511.
[51] A. Pommert, U. Tiede, G. Wiebecke, K.H. Hohne, Surface shading in tomographic volume visualization: a comparative study, in: Proc. First Conf. Vis. Biomed. Comput., 1990: pp. 19–26. doi:10.1109/VBC.1990.109297.
[52] M. Steyaert, W.M. Sansen, Design of multi-bit delta-sigma A/D converters, 2002.
[53] R. Keys, Cubic convolution interpolation for digital image processing, IEEE Trans. Acoust. 29 (1981) 1153–1160. doi:10.1109/TASSP.1981.1163711.
[54] R. a Drebin, L. Carpenter, P. Hanrahan, Volume Rendering, Proc. SIGGRAPH '88. Comp Graph. 22 (1988) 65–74. doi:10.1145/54852.378484.
[55] T. He, L. Hong, A. Kaufman, H. Pfister, Generation of transfer functions with stochastic search techniques, in: Proc. 7th Conf. Vis. '96, San Fransisco, CA, 1996: pp. 227–234. doi:10.1109/VISUAL.1996.568113.
[56] V. Šoltészová, D. Patel, S. Bruckner, I. Viola, A Multidirectional Occlusion Shading Model for Direct Volume Rendering, Comput. Graph. Forum. 29 (2010) 883–891. doi:10.1111/j.1467-8659.2009.01695.x.
[57] B.T. Phong, Illumination for computer generated pictures, Commun. ACM. 18 (1975) 311–317. doi:10.1145/360825.360839.
[58] H. Gouraud, Continuous Shading of Curved Surfaces, IEEE Trans. Comput. C-20 (1971) 623–629. doi:10.1109/T-C.1971.223313.
[59] J.F. Blinn, Light reflection functions for simulation of clouds and dusty surfaces, in: ACM SIGGRAPH Comput. Graph., 1982: pp. 21–29. doi:10.1145/965145.801255.
[60] T. Porter, T. Duff, Compositing digital images, ACM SIGGRAPH Comput. Graph. 18 (1984) 253–259. doi:10.1145/964965.808606.
[61] P. Sabella, A rendering algorithm for visualizing 3D scalar fields, in: ACM SIGGRAPH Comput. Graph., 1988: pp. 51–58. doi:10.1145/378456.378476.
[62] C. Upson, M. Keeler, V-buffer: Visible Volume Rendering, in: Proc. 15th Annu. Conf. Comput. Graph. Interact. Tech. - SIGGRAPH '88, 1988: pp. 59–64. doi:10.1145/54852.378482.
[63] M.S. Langer, H.H. Bülthoff, Depth discrimination from shading under diffuse lighting, Perception. 29 (2000) 649–660. doi:10.1068/p3060.
[64] M. Sattler, R. Sarlette, T. Mücken, R. Klein, Exploitation of human shadow perception for fast shadow rendering, in: Proc. 2nd Symp. Appied Percept. Graph. Vis. - APGV '05, 2005: pp. 131–134. doi:10.1145/1080402.1080426.
[65] M. Hadwiger, P. Ljung, C.R. Salama, T. Ropinski, Advanced illumination techniques for GPU volume raycasting, in: ACM SIGGRAPH ASIA 2008 Courses - SIGGRAPH Asia '08, 2008: pp. 1–166. doi:10.1145/1508044.1508045.

المنارة للاستشارات
www.manaraa.com

[66] F. Hernell, P. Ljung, A. Ynnerman, Interactive global light propagation in direct volume rendering using local piecewise integration, in: SPBG'08 Proc. Fifth Eurographics / IEEE VGTC Conf. Point-Based Graph., 2008: pp. 105–112. doi:10.2312/VG/VG-PBG08/105-112.

[67] I. Wald, H. Friedrich, G. Marmitt, P. Slusallek, H.P. Seidel, Faster isosurface ray tracing using implicit KD-trees, IEEE Trans. Vis. Comput. Graph. 11 (2005) 562–572. doi:10.1109/TVCG.2005.79.

[68] G. Marmitt, H. Friedrich, P. Slusallek, Interactive Volume Rendering with Ray Tracing, in: Eurographics State Art Reports, 2006: pp. 115–136.

[69] L. Williams, Casting curved shadows on curved surfaces, in: ACM SIGGRAPH Comput. Graph., 1978: pp. 270–274. doi:10.1145/965139.807402.

[70] W.T. Reeves, D.H. Salesin, R.L. Cook, Rendering antialiased shadows with depth maps, in: ACM SIGGRAPH Comput. Graph., 1987: pp. 283–291. doi:10.1145/37402.37435.

[71] T. Kim, U. Neumann, Opacity Shadow Maps, in: Render. Tech. 2001, Springer Vienna, 2001: pp. 177–182. doi:10.1007/978-3-7091-7484-5.

[72] T. Lokovic, E. Veach, Deep shadow maps, in: Proc. 27th Annu. Conf. Comput. Graph. Interact. Tech. SIGGRAPH 00, 2000: pp. 385–392. doi:10.1145/344779.344958.

[73] M. Hadwiger, A. Kratz, C. Sigg, K. Bühler, GPU-accelerated deep shadow maps for direct volume rendering, in: Proc. 21st ACM SIGGRAPH/EUROGRAPHICS Symp. Graph. Hardw. - GH '06, 2006: pp. 49–52. doi:10.1145/1283900.1283908.

[74] a. J. Stewart, Vicinity Shading for Enhanced Perception of Volumetric Data, in: Proc. IEEE Vis. Conf., 2003: pp. 355–362. doi:10.1109/VISUAL.2003.1250394.

[75] P. Desgranges, K. Engel, Fast ambient occlusion for direct volume rendering, US 11/455,627, 2006.

[76] F. Hernell, P. Ljung, A. Ynnerman, Efficient ambient and emissive tissue illumination using local occlusion in multiresolution volume rendering, in: Vol. Graph. 2007. Proc. Sixth Eurographics / IEEE VGTC Conf., 2007: pp. 1–8. doi:10.2312/VG/VG07/001-008.

[77] F. Hernell, P. Ljung, A. Ynnerman, Local ambient occlusion in direct volume rendering, IEEE Trans. Vis. Comput. Graph. 16 (2010) 548–559. doi:10.1109/TVCG.2009.45.

[78] G. Khanduja, B. Karki, A systematic approach to multiple datasets visualization of scalar volume data, Grapp 2006. (2006) 59–66. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.98.5016&rep=rep1&type=pdf.

[79] C. a. Dietrich, L.P. Nedel, S.D. Olabarriaga, J.L.D. Comba, D.J. Zanchet, A.M. Marques da Silva, et al., Real-time interactive visualization and manipulation of the volumetric data using GPU-based methods, in: Med. Imaging 2004 Vis. Image-Guided Proced. Disp., 2004: pp. 181–192. doi:10.1117/12.536578.

[80] T. McInerney, S. Broughton, HingeSlicer: interactive exploration of volume images using extended 3D slice plane widgets, in: Proc. 2006 Conf. Graph. Interface, 2006: pp. 171–178. http://portal.acm.org/citation.cfm?id=1143079.1143107.

[81] S.W. Wang, A.E. Kaufman, Volume sculpting, Symp. Interact. 3D Graph. (1995) 151–157. doi:10.1145/199404.199430.

[82] D. Weiskopf, K. Engel, T. Ertl, Interactive clipping techniques for texture-based volume visualization and volume shading, IEEE Trans. Vis. Comput. Graph. 9 (2003) 298–312. doi:10.1109/TVCG.2003.1207438.

[83] O. Konrad-Verse, B. Preim, A. Littmann, Virtual Resection with a Deformable Cutting Plane., in: SimVis, 2004: pp. 203–214.

[84] S. Stegmaier, M. Strengert, T. Klein, T. Ertl, A simple and flexible volume rendering framework for graphics-hardware-based raycasting, in: Fourth Int. Work. Vol. Graph. 2005., IEEE, 2005: pp. 187–195. doi:10.1109/VG.2005.194114.

للاستشارات المنارة

www.manaraa.com

[85] R. Brecheisen, A. Vilanova, B. Platel, H. Romeny, Flexible GPU-Based Multi-Volume Ray-Casting, in: Proc. Vision, Model. Vis., 2008: pp. 303–312. http://books.google.com/books?hl=en&lr=&id=WCVFWYIAeCMC&oi=fnd&pg=PA303&dq=Flexible+GPU-Based+Multi-Volume+Ray-Casting&ots=QzElVja2F6&sig=r-IRKzUvkOzYzBK689h3An3-Sr8.
[86] M. Chen, D. Silver, a S. Winter, V. Singh, N. Cornea, Spatial transfer functions: a unified approach to specifying deformation in volume modeling and animation, in: Proc. 2003 Eurographics/IEEE TVCG Work. Vol. Graph. - VG '03, 2003: pp. 35–44. doi:10.1145/827051.827056.
[87] S. Islam, S. Dipankar, D. Silver, M.C.M. Chen, Spatial and temporal splitting of scalar fields in volume graphics, in: 2004 IEEE Symp. Vol. Vis. Graph., 2004: pp. 87–94. doi:10.1109/SVVG.2004.11.
[88] M.J. McGuffin, L. Tancau, R. Balakrishnan, Using Deformations for Browsing Volumetric Data, in: Proc. IEEE Vis. Conf., 2003: pp. 401–408. doi:10.1109/VISUAL.2003.1250400.
[89] I. Viola, A. Kanitsar, M.E. Groller, Importance-driven volume rendering, in: IEEE Vis. 2004, 2004: pp. 139–145. doi:10.1109/VISUAL.2004.48.
[90] S. Bruckner, M.E. Gröller, Exploded views for volume data, IEEE Trans. Vis. Comput. Graph. 12 (2006) 1077–1084. doi:10.1109/TVCG.2006.140.
[91] J. Kruger, R. Westermann, Acceleration Techniques for GPU-based Volume Rendering, in: IEEE_VIS, Seattle, Washington, USA, 2003: pp. 38–44. doi:10.1109/VIS.2003.10001.
[92] T. Whitted, An improved illumination model for shaded display, ACM SIGGRAPH Comput. Graph. 13 (1979) 14. doi:10.1145/965103.807419.
[93] M. Levoy, Efficient ray tracing of volume data, ACM Trans. Graph. 9 (1990) 245–261. doi:10.1145/78964.78965.
[94] D. Meagher, Geometric modeling using octree encoding, Comput. Graph. Image Process. 19 (1982) 129–147. doi:10.1016/0146-664X(82)90128-9.
[95] S.Y.S. You, L.H.L. Hong, M.W.M. Wan, K. Junyaprasert, A. Kaufman, S. Muraki, et al., Interactive volume rendering for virtual colonoscopy, in: Proceedings. Vis. '97 (Cat. No. 97CB36155), 1997: pp. 433–436. doi:10.1109/VISUAL.1997.663915.
[96] M. Wan, Q. Tang, A. Kaufman, Z. Liang, M. Wax, Volume rendering based interactive navigation within the human colon, in: Proc. Vis. '99 (Cat. No.99CB37067), 1999: pp. 397–401. doi:10.1109/VISUAL.1999.809914.
[97] M. Wan, A. Sadiq, A. Kaufman, Fast and Reliable Leaping for Interactive Volume Rendering, in: IEEE Vis. 2002, 2002: pp. 195–202.
[98] T.H. Lee, J. Lee, H. Lee, H. Kye, Y.G. Shin, S.H. Kim, Fast perspective volume ray casting method using GPU-based acceleration techniques for translucency rendering in 3D endoluminal CT colonography, Comput. Biol. Med. 39 (2009) 657–666. doi:10.1016/j.compbiomed.2009.04.007.
[99] W.L.W. Li, K. Mueller, A. Kaufman, Empty space skipping and occlusion clipping for texture-based volume rendering, in: IEEE Vis. 2003. VIS 2003., 2003: pp. 317–324. doi:10.1109/VISUAL.2003.1250388.
[100] W. Li, A. Kaufman, Texture Partitioning and Packing for Accelerating Texture-based Volume Rendering, Graph. Interface. 3 (2003) 81–88.
[101] H. Fuchs, Z.M. Kedem, B.F. Naylor, On visible surface generation by a priori tree structures, in: ACM SIGGRAPH Comput. Graph., 1980: pp. 124–133. doi:10.1145/965105.807481.
[102] J. Wilhelms, A. Van Gelder, Octrees for faster isosurface generation, ACM Trans. Graph. 11 (1992) 201–227. doi:10.1145/130881.130882.
[103] P. Sutton, C.D. Hansen, Isosurface extraction in time-varying fields using a Temporal Branch-on-Need Tree (T-BON), in: Proc. Vis., Ieee, 1999: pp. 147–520. doi:10.1109/VISUAL.1999.809879.

المنارة للاستشارات
www.manaraa.com

[104] K. Ma, H. Shen, Compression and Accelerated Rendering of Time-Varying Volume Data, in: Proc. 2000 Int. Comput. Symp. Comput. Graph. Virtual Real., 2000: pp. 82–89. http://www.cse.ohio-state.edu/~hwshen/papers/Ma2000.pdf (accessed October 23, 2014).

[105] I. Boada, I. Navazo, R. Scopigno, Multiresolution volume visualization with a texture-based octree, Vis. Comput. 17 (2001) 185–197. doi:10.1007/PL00013406.

[106] J. Plate, M. Tirtasana, R. Carmona, B. Frohlich, Octreemizer: A Hierarchical Approach for Interactive Roaming Through Very Large Volumes, Eurographics/IEEE TCVG Symp. Vis. (2002) 53–60.

[107] E. LaMar, B. Hamann, K.I. Joy, Multiresolution Techniques for Interactive Texture-Based Volume Visualization, in: R.F. Erbacher, P.C. Chen, J.C. Roberts, C.M. Wittenbrink (Eds.), SPIE 3960, Vis. Data Explor. Anal. VII, San Jose, CA, 2000: pp. 365–374. doi:10.1117/12.378913.

[108] J.L. Bentley, Multidimensional binary search trees used for associative searching, Commun. ACM. 18 (1975) 509–517. doi:10.1145/361002.361007.

[109] G. Marmitt, R. Brauchle, H. Friedrich, P. Slusallek, Accelerated and Extended Building of Implicit kd-Trees for Volume Ray Tracing, in: Proc. 11th Int. Fall Work. Model. Vis., 2006: pp. 317–324. http://books.google.com/books?hl=en&lr=&id=zndnSzkfkXwC&oi=fnd&pg=PA317&dq=Accelerated+and+Extended+Building+of+Implicit+kd-Trees+for+Volume+Ray+Tracing&ots=0Y9_bDUJh6&sig=BYJmJC3nSGEh5ZpU_IE0SvbmrY8 (accessed October 16, 2014).

[110] H. Shen, L.-J. Chiang, K.-L. Ma, A Fast Volume Rendering Algorithm for Time-Varying Fields Using a Time-Space Partitioning (TSP) Tree, in: Proc. Conf. Vis., 1999: pp. 371–377.

[111] D. Ellsworth, L.-J. Chiang, H.-W. Shen, Accelerating time-varying hardware volume rendering using TSP trees and color-based error metrics, in: Proc. 2000 IEEE Symp. Vol. Vis. - VVS '00, ACM Press, New York, New York, USA, 2000: pp. 119–128. doi:10.1145/353888.353908.

[112] N. Neophytou, K. Mueller, Space-Time Points: 4D Splatting on Efficient Grids, in: Proc. 2002 IEEE Symp. Vol. Vis. Graph., 2002: pp. 97–106.

[113] R.C. Gonzalez, Digital image processing, Pearson Education India., 2009.

[114] K. Sayood, Introduction to data compression, Newnes, 2012.

[115] E.B. Lum, K. Ma, J. Clyne, Texture Hardware Assisted Rendering of Time-Varying Volume Data, in: Proc. Conf. Vis., 2001: pp. 263–270.

[116] I. Daubechies, Orthonormal Bases of Compactly Supported Wavelets II. Variations on a Theme, SIAM J. Math. Anal. 24 (1993) 499–519. doi:10.1137/0524031.

[117] A. Cohen, I. Daubechies, J.C. Feauveau, Biorthogonal bases of compactly supported wavelets, in: Commun. Pure Appl. Math., 1992: pp. 485–560. doi:10.1002/cpa.3160450502.

[118] S. Guthe, W. Strasser, Real-time decompression and visualization of animated volume data, in: Proc. Vis. 2001. VIS '01., IEEE, 2001: pp. 349–372. doi:10.1109/VISUAL.2001.964531.

[119] S. Guthe, M. Wand, J. Gonser, W. Strasser, Interactive rendering of large volume data sets, in: IEEE Vis. 2002, IEEE, Boston, Massachusetts, 2002: pp. 53–60. doi:10.1109/VISUAL.2002.1183757.

[120] K.G. Nguyen, D. Saupe, Rapid High Quality Compression of Volume Data for Visualization, Comput. Graph. Forum. 20 (2001) 49–57. doi:10.1111/1467-8659.00497.

[121] M. Strengert, M. Magallón, D. Weiskopf, S. Guthe, T. Ertl, Hierarchical visualization and compression of large volume datasets using GPU clusters, in: Eurographics Conf. Parallel Graph. Vis., 2004: pp. 41–48. doi:10.2312/EGPGV/EGPGV04/041-048.

[122] M. Kraus, T. Ertl, Adaptive Texture Maps, in: Proc. ACM SIGGRAPH/EUROGRAPHICS Conf. Graph. Hardw., 2002: pp. 7–15.

www.manaraa.com

[123] A.P.D. Binotto, J.L.D. Comba, C.M.D. Freitas, Real-time volume rendering of time-varying data using a fragment-shader compression approach, in: Proc. 2003 IEEE Symp. Parallel Large-Data Vis. Graph., IEEE, Seattle, Washington, USA, 2003: pp. 69–75. doi:10.1109/PVGS.2003.1249044.

[124] J. Schneider, R. Westermann, Compression domain volume rendering, in: IEEE Vis., IEEE, Seattle, Washington, USA, 2003: pp. 293–300. doi:10.1109/VISUAL.2003.1250385.

[125] M. Ferre, A. Puig, D. Tost, A framework for fusion methods and rendering techniques of multimodal volume data, Comput. Animat. Virtual Worlds. 15 (2004) 63–77. doi:10.1002/cav.1.

[126] M. Ferré, A. Puig, D. Tost, Rendering Techniques for Multimodal Data, in: Proc. SIACG 2002 1st Ibero-American Symp. Comput. Graph., 2002: pp. 305–313.

[127] L. Serra, R.A. Kockro, C.G. Guan, N. Hern, E.C.K. Lee, Y.H. Lee, et al., Multimodal volume-based tumor neurosurgery planning in the virtual workbench, Med. Image Comput. Comput. Interv. — MICCAI'98. 1496 (1998) 1007–1015.

[128] D. Patel, L.P. Muren, A. Mehus, Y. Kvinnsland, D.M. Ulvang, K.P. Villanger, A virtual reality solution for evaluation of radiotherapy plans, Radiother. Oncol. 82 (2007) 218–221. doi:10.1016/j.radonc.2006.11.024.

[129] F. Robler, E. Tejada, T. Fangmeier, T. Ertl, M. Knauff, GPU-based Multi-Volume Rendering for the Visualization of Functional Brain Images, SimVis. (2006) 305–318.

[130] T. Schafhitzel, F. Robler, D. Weiskopf, T. Ertl, Simultaneous visualization of anatomical and functional 3D data by combining volume rendering and flow visualization, in: Med. Imaging 2007 Vis. Image-Guided Proced., 2007: pp. 1–9. doi:10.1117/12.708799.

[131] B. Wilson, E.B. Lum, K. Ma, Interactive Multi-volume Visualization, in: Comput. Sci. — ICCS 2002, Springer Berlin Heidelberg, 2002: pp. 102–110. doi:10.1007/3-540-46080-2_11.

[132] M. Levoy, A hybrid ray tracer for rendering polygon and volume data, IEEE Comput. Graph. Appl. 10 (1990) 33–40. doi:10.1109/38.50671.

[133] K. Kreeger, A. Kaufman, Mixing Translucent Polygons with Volumes, in: IEEE Vis., 1999: pp. 191–198.

[134] C. Everitt, Interactive Order-Independent Transparency, 2001. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.18.9286&rep=rep1&type=pdf.

[135] J. Plate, T. Holtkaemper, B. Froehlich, A flexible multi-volume shader framework for arbitrarily intersecting multi-resolution datasets, IEEE Trans. Vis. Comput. Graph. 13 (2007) 1584–1591. doi:10.1109/TVCG.2007.70534.

[136] B. Kainz, M. Grabner, A. Bornik, S. Hauswiesner, J. Muehl, D. Schmalstieg, Ray casting of multiple volumetric datasets with polyhedral boundaries on manycore GPUs, ACM Trans. Graph. 28 (2009) 1–9. doi:10.1145/1618452.1618498.

[137] W. Cai, G. Sakas, Data Intermixing and Multi-volume Rendering, Comput. Graph. Forum. 18 (1999) 359–368. doi:10.1111/1467-8659.00356.

[138] I.H. Manssour, S.S. Furuie, L.P. Nedel, C.M.D.S. Freitas, A Framework to Visualize and Interact with Multimodal Medical Images, in: Vol. Graph., 2001: pp. 385–398. doi:10.1007/978-3-7091-6756-4_27.

[139] J. Beyer, M. Hadwiger, S. Wolfsberger, K. Bühler, High-quality multimodal volume rendering for preoperative planning of neurosurgical interventions, IEEE Trans. Vis. Comput. Graph. 13 (2007) 1696–1703. doi:10.1109/TVCG.2007.70560.

[140] I.H. Manssour, S.S. Furuie, S.D. Olabarriaga, C.M.D.S. Freitas, Visualizing inner structures in multimodal volume data, in: Proceedings. XV Brazilian Symp. Comput. Graph. Image Process., 2002: pp. 51–58. doi:10.1109/SIBGRA.2002.1167123.

[141] S. Grimm, S. Bruckner, A. Kanitsar, E. Groller, Flexible Direct Multi-Volume Rendering in Dynamic Scenes, in: Proc. Int. Fall Work. Vision, Model. Vis., 2004: pp. 379–386.

المنارة للاستشارات
www.manaraa.com

[142] D.R. Nadeau, Volume scene graphs, in: Proc. 2000 IEEE Symp. Vol. Vis. - VVS ’00, ACM Press, New York, New York, USA, 2000: pp. 49–56. doi:10.1145/353888.353898.

[143] F. Rößler, R.P. Botchen, T. Ertl, Dynamic shader generation for GPU-based multi-volume ray casting, IEEE Comput. Graph. Appl. 28 (2008) 66–77. doi:10.1109/MCG.2008.96.

[144] A. Leu, M. Chen, Modelling and Rendering Graphics Scenes Composed of Multiple Volumetric Datasets, Comput. Graph. Forum. 18 (1999) 159–171. doi:10.1111/1467-8659.00366.

[145] R.A. Robb, Visualization in biomedical computing, Parallel Comput. 25 (1999) 2067–2110. doi:10.1016/S0167-8191(99)00076-9.

[146] W.L. Nowinski, B.C. Chua, G.Y. Qian, N.G. Nowinska, The human brain in 1700 pieces: design and development of a three-dimensional, interactive and reference atlas., J. Neurosci. Methods. 204 (2012) 44–60. doi:10.1016/j.jneumeth.2011.10.021.

[147] K.J. Zuiderveld, M.A. Viergever, Multi-modal volume visualization using object-oriented methods, in: Proc. 1994 Symp. Vol. Vis. - VVS ’94, ACM Press, New York, New York, USA, 1994: pp. 59–66. doi:10.1145/197938.197966.

[148] Q. Zhang, R. Eagleson, T.M. Peters, Dynamic real-time 4D cardiac MDCT image display using GPU-accelerated volume rendering., Comput. Med. Imaging Graph. 33 (2009) 461–76. doi:10.1016/j.compmedimag.2009.04.002.

[149] H. Hauser, L. Mroz, M.E. Gr, Two-level volume rendering – fusing MIP and DVR, in: Proc. Conf. Vis., IEEE, 2000: pp. 211–218.

[150] H. Hauser, L. Mroz, G.I. Bischi, M.E. Gro, Two-Level Volume Rendering, IEEE Trans. Vis. Comput. Graph. 7 (2001) 242–252.

[151] F. Enders, M. Strengert, S. Iserhardt-Bauer, U.E. Aladl, P.J. Slomka, Interactive Volume Rendering of Multimodality 4D Cardiac Data with the Use of Consumer Graphics Hardware, in: R.L. Galloway (Ed.), Vis. Image-Guided Proced. Disp., 2003: pp. 119–128. doi:10.1117/12.480377.

[152] A.J. Hanson, P.A. Heng, Four-dimensional views of 3D scalar fields, in: Proc. Vis. ’92, IEEE Comput. Soc. Press, 1992: pp. 84–91. doi:10.1109/VISUAL.1992.235222.

[153] J. Woodring, C. Wang, H.-W. Shen, High dimensional direct rendering of time-varying volumetric data, in: IEEE Trans. Vis., IEEE, Seattle, Washington, USA, 2003: pp. 417–424. doi:10.1109/VISUAL.2003.1250402.

[154] J. Woodring, H. Shen, Chronovolumes: a direct rendering technique for visualizing time-varying data, in: Proc. 2003 Eurographics/IEEE TVCG Work. Vol. Graph. - VG ’03, ACM Press, New York, New York, USA, 2003: p. 27. doi:10.1145/827051.827054.

[155] J. Woodring, H.W. Shen, Multi-variate, time-varying, and comparative visualization with contextual cues, IEEE Trans. Vis. Comput. Graph. 12 (2006) 909–916. doi:10.1109/TVCG.2006.164.

[156] J. Meyer-Spradow, T. Ropinski, J. Mensmann, K. Hinrichs, Voreen: a rapid-prototyping environment for ray-casting-based volume visualizations., IEEE Comput. Graph. Appl. 29 (2009) 6–13. doi:10.1109/MCG.2009.130.

[157] P. Bhaniramka, R. Wenger, R. Crawfis, Isosurfacing in higher dimensions, in: Proc. Vis. 2000. VIS 2000, IEEE, Salt Lake City, UT, 2000: pp. 267–273. doi:10.1109/VISUAL.2000.885704.

[158] R. Chaoui, J. Hoffmann, K.S. Heling, Three-dimensional (3D) and 4D color Doppler fetal echocardiography using spatio-temporal image correlation (STIC)., Ultrasound Obstet. Gynecol. 23 (2004) 535–45. doi:10.1002/uog.1075.

[159] J. Espinoza, L.F. Gonçalves, W. Lee, M. Mazor, R. Romero, A novel method to improve prenatal diagnosis of abnormal systemic venous connections using three- and four-dimensional ultrasonography and “inversion mode”., Ultrasound Obstet. Gynecol. 25 (2005) 428–34. doi:10.1002/uog.1877.

[160] M. Tory, N. Röber, T. Möller, A. Celler, M.S. Atkins, 4D Space-Time Techniques : A Medical Imaging Case Study, in: IEEE Vis., San Diego, 2001: pp. 473–477.

المنارة للاستشارات

www.manaraa.com

[161] C.R. Johnson, Biomedical Visual Computing: Case Studies and Challenges., Comput. Sci. Eng. 14 (2012) 12–21. doi:10.1109/MCSE.2011.92.
[162] A. Krüeger, C. Kubisch, G. Straub, B. Preim, Sinus endoscopy--application of advanced GPU volume rendering for virtual endoscopy., IEEE Trans. Vis. Comput. Graph. 14 (2008) 1491–1498. doi:10.1109/TVCG.2008.161.
[163] C. Noon, J. Holub, E. Winer, Real-time volume rendering of digital medical images on an iOS device, 8667 (2013) 86670U. doi:10.1117/12.2005335.
[164] C. Noon, E. Foo, E. Winer, Interactive GPU volume raycasting in a clustered graphics environment, Med. Imaging 2012 Image-Guided Proced. Robot. Interv. Model. 8316 (2012) 1–9. doi:10.1117/12.911646.
[165] A. Hennemuth, O. Friman, C. Schumann, J. Bock, J. Drexl, M. Huellebrand, et al., Fast Interactive Exploration of 4D MRI Flow Data, 7964 (2011) 79640E–79640E–11. doi:10.1117/12.878202.
[166] H. Peng, A. Bria, Z. Zhou, G. Iannello, F. Long, Extensible visualization and analysis for multidimensional images using Vaa3D., Nat. Protoc. 9 (2014) 193–208. doi:10.1038/nprot.2014.011.
[167] H. Peng, Z. Ruan, F. Long, J.H. Simpson, E.W. Myers, V3D enables real-time 3D visualization and quantitative analysis of large-scale biological image data sets., Nat. Biotechnol. 28 (2010) 348–53. doi:10.1038/nbt.1612.
[168] S.S. Raman, F.C. Osuagwu, B. Kadell, H. Cryer, J. Sayre, D.S.K. Lu, Effect of CT on false positive diagnosis of appendicitis and perforation., N. Engl. J. Med. 358 (2008) 972–973. doi:10.1056/NEJMc0707000.
[169] Overview United States Department of Defense Fiscal Year 2015 Budget Request Office of the Under Secretary of Defense (Comptroller)/ Chief Financial Officer, 2014. http://comptroller.defense.gov/Portals/45/Documents/defbudget/fy2015/fy2015_Budget_Request_Overview_Book.pdf.
[170] M. Martinez Escobar, B. Juhnke, K. Hisley, D. Eliot, E. Winer, Assessment of visual-spatial skills in medical context tasks when using monoscopic and stereoscopic visualization, in: SPIE Med. Imaging, 2013: p. 86730N. doi:10.1117/12.2007087.
[171] W.C.W. Chen, L.R.L. Ren, M. Zwicker, H. Pfister, Hardware-accelerated adaptive EWA volume splatting, IEEE Vis. 2004. (2004). doi:10.1109/VISUAL.2004.38.
[172] DirectX, (2015). https://support.microsoft.com/en-us/kb/179113.
[173] OpenGL, (2015).
[174] OpenSceneGraph, (2015).
[175] VR Juggler, (2012). https://code.google.com/p/vrjuggler/wiki/WikiStart.
[176] O.G. Staadt, J. Walker, C. Nuber, B. Hamann, A survey and performance analysis of software platforms for interactive cluster-based multi-screen rendering, EGVE '03 Proc. Work. Virtual Environ. 2003. (2003) 261–270. doi:10.1145/769953.769984.
[177] INMOS, Graphics Databook, 2nd ed., Redwood Press Limited, Melksham,Wiltshire, 1990.
[178] A. Van Gelder, K. Kim, Direct volume rendering with shading via three-dimensional textures, in: Proc. 1996 Symp. Vol. Vis., Acm, San Fransisco, CA, 1996: p. 23–30,. doi:10.1109/SVV.1996.558039.
[179] Metal, (20015).
[180] Khronos Group, Vulkan, (2016).
[181] Khronos Group, OpenGL ES, (2016). https://www.khronos.org/opengles/.
[182] L. Forsberg, NeuroPub, (2013).
[183] J. Holub, E. Winer, Visualizing fMRI Data Using Volume Rendering in Virtual Reality Joseph, in: Interservice/Industry Training, Simulation, Educ. Conf., Orlando, FL, 2015: pp. 1–11.
[184] J. Gaudiosi, How this med school is using virtual reality to teach students, Fortune. (2015). http://fortune.com/2015/10/16/western-university-is-using-virtual-reality-to-teach/ (accessed January 1, 2016).

المنارة للاستشارات
www.manaraa.com

[185] W.D. Bidgood, S.C. Horii, F.W. Prior, D.E. Van Syckle, Understanding and Using DICOM, the Data Interchange Standard for Biomedical Imaging, J. Am. Med. Informatics Assoc. 4 (1997) 199–212. doi:10.1136/jamia.1997.0040199.
[186] Siemens, Syngo, (2016). https://www.healthcare.siemens.com/medical-imaging-it/imaging-it-radiology-image-management-pacs/syngoplaza/features.
[187] FEI, Amira, (2016). https://www.fei.com/software/amira-avizo/.
[188] A. Rosset, L. Spadola, O. Ratib, OsiriX: An open-source software for navigating in multidimensional DICOM images, J. Digit. Imaging. 17 (2004) 205–216. doi:10.1007/s10278-004-1014-6.
[189] Pixmeo, Osirix Viewer, (2016). http://www.osirix-viewer.com.
[190] MeVis Medical Solutions AG, MeVisLab, (2016). http://www.mevislab.de.
[191] M. St John, M.B. Cowen, H.S. Smallman, H.M. Oonk, The use of 2D and 3D displays for shape-understanding versus relative-position tasks., Hum. Factors. 43 (2001) 79–98. doi:10.1518/001872001775992534.
[192] M. Martinez Escobar, B. Junke, J. Holub, K. Hisley, D. Eliot, E. Winer, Evaluation of monoscopic and stereoscopic displays for visual–spatial tasks in medical contexts, Comput. Biol. Med. 61 (2015) 138–143. doi:10.1016/j.compbiomed.2015.03.026.
[193] D.J. Getty, R.M. Pickett, C.J. D'Orsi, Stereoscopic digital mammography: improving detection and diagnosis of breast cancer, Int. Congr. Ser. 1230 (2001) 538–544. doi:10.1016/S0531-5131(01)00084-X.
[194] Qt, (2015). http://www.qt.io/developers/.

المنارة للاستشارات
www.manaraa.com