Visualizing fMRI Data Using Volume Rendering in Virtual Reality

Joseph Holub, Eliot Winer

Iowa State University
Ames, Iowa

jholub@iastate.edu, ewiner@iastate.edu

ABSTRACT

Medical imaging technology has changed patient diagnosis since the first x-ray in 1895 (Rontgen, 1896). 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% (Raman et al., 2008). With the 2015 Defense Budget including $47.4 billion for the Military Health System (*Overview United States Department of Defense Fiscal Year 2015 Budget Request Office of the Under Secretary of Defense (Comptroller)/ Chief Financial Officer*, 2014), 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 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.

ABOUT THE AUTHORS

Joseph Holub is a graduate research assistant at the Virtual Reality Applications Center (VRAC) at Iowa State University where he is finishing his Ph.D. in human computer interaction (HCI) and computer engineering. He has worked on projects for path planning of unmanned aerial vehicles, live virtual constructive training of dismounted soldier, and visualization of large data using contextual self-organizing maps. Currently, he is working on building tools for augmented reality assembly in manufacturing as well as a tool for team training. His Ph.D. research is on visualizing functional imaging data on mobile, desktop, and virtual reality hardware platforms.

Eliot Winer, Ph.D., is an associate director of the Virtual Reality Applications Center (VRAC), associate professor of mechanical engineering, and a faculty affiliate of the human computer interaction (HCI) graduate program at Iowa State University. He recently led research developing and integrating four virtual and three live environments in a simultaneous capability demonstration for the Air Force Office of Scientific Research. He is currently leading an effort to develop a next-generation mixed-reality virtual and constructive training environment for the US Army. This environment will allow rapid reconfiguration (e.g., approximately 20 minutes) of a training system, including walls, streets, building textures, etc., to produce radically different scenarios for warfighter training. Dr. Winer has over 15 years experience working, virtual reality, computer graphics, and simulation technologies.

Visualizing fMRI Data Using Volume Rendering in Virtual Reality

Joseph Holub, Eliot Winer

Iowa State University
Ames, Iowa

jholub@iastate.edu, ewiner@iastate.edu

INTRODUCTION

Medical imaging was first discovered in 1895 with the first x-ray (Rontgen, 1896) 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% (Raman et al., 2008). 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 (*Overview United States Department of Defense Fiscal Year 2015 Budget Request Office of the Under Secretary of Defense (Comptroller)/ Chief Financial Officer*, 2014).

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, 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 (Martinez Escobar, Juhnke, Hisley, Eliot, & Winer, 2013). 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 (Solomon, 2014) 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 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.

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 1 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.

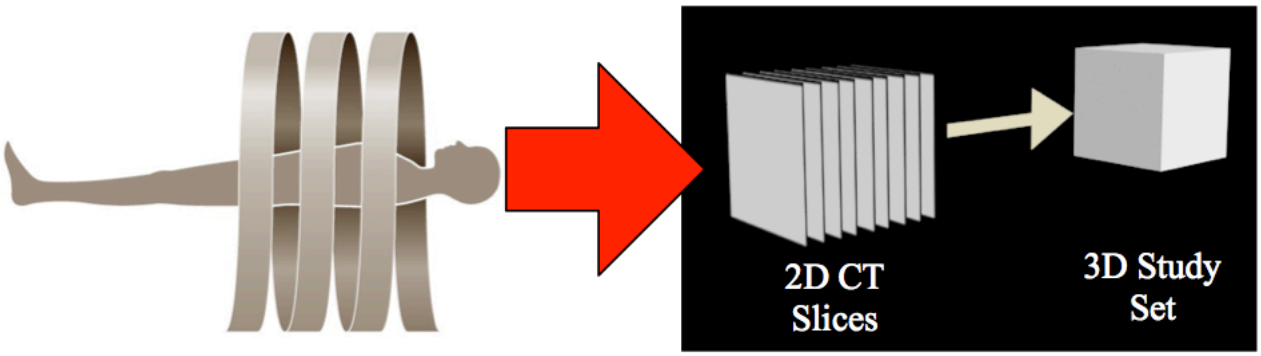

Figure 1. Process of obtaining a 3D volumetric dataset from 2D anatomical slices

MRI technology was first discovered in 1946 by Felix Bloch and Edward Purcell, but was not used for imaging purposes until the early 1970s (Bottomley, 1983). 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 (Shung, Smith, & Tsui, 1992).

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 (Di Salle et al., 1999). This technology has impacted research in areas such as science, clinical practice, cognitive neuroscience, mental illness (Rosen & Savoy, 2012).

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 (Kaufman & Mueller, 2005; Lorensen & Cline, 1987). Image Splatting is a direct volume rendering method using overlapping basis functions to represent the voxels (Botsch & Kobbelt, 2003; Chen, Ren, Zwicker, & Pfister, 2004; Neophytou & Mueller, 2005; Westover, 1990). 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 (Pfister, Hardenbergh, Knittel, Lauer, & Seiler, 1999; Wu, Bhatia, Lauer, & Seiler, 2003). Texture Slicing generates viewport-aligned slices parallel to the viewing plane and then composites them together for a final image (Engel, Kraus, & Ertl, 2001; Rezk-Salama, Engel, Bauer, Greiner, & Ertl, 2000).

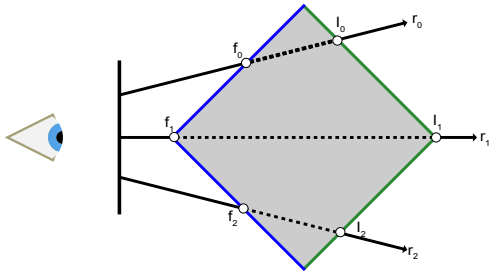

Figure 2. Example of raycasting with rays being cast from the viewing plan on the left through the volume on the right

Raycasting is a direct volume rendering method that casts rays in the viewing direction from each pixel of the screen. Figure 2 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. Of the five methods described, raycasting is generally considered to achieve the best visual representation. This research will utilize a raycasting implementation (Pfister, 2005; Weiskopf, 2006).

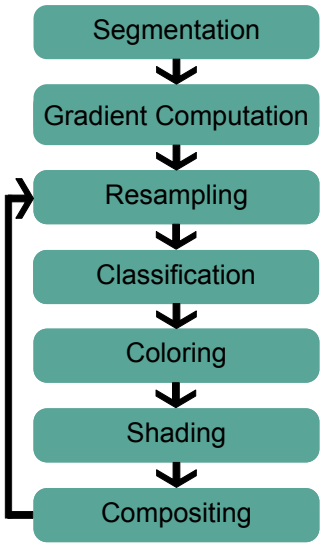

Figure 3. Volume rendering pipeline example

Volumetric raycasting typically goes through a rendering pipeline similar to that seen in Figure 3. 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 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.

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.

There are three commonly used open source 4D volumetric visualization tools, Osirix, MeVisLab (Hennemuth et al., 2011), 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 (Peng, Bria, Zhou, Iannello, & Long, 2014; Peng, Ruan, Long, Simpson, & Myers, 2010).

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.

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 (Serra et al., 1998) and the work done by Patel, et al., (Patel et al., 2007) 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 (Patel et al., 2007).

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 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.

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 4 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.

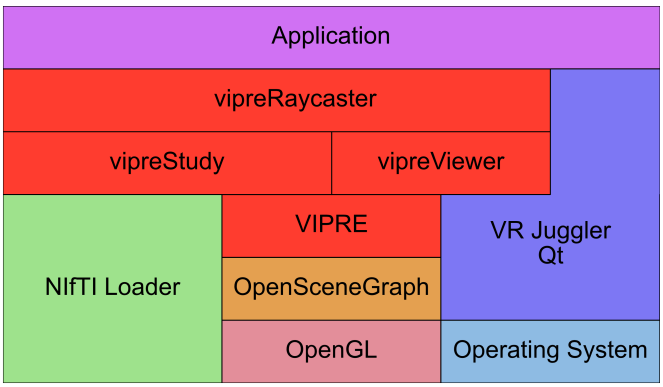

Figure 4. Simplified library architecture

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 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 ("DirectX," 2015) and OpenGL ("OpenGL," 2015). 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 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.

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.

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 ("OpenSceneGraph," 2015), 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.

VR Juggler

VR Juggler is a cross-platform, open-source virtual reality software development environment designed for executing immersive applications across multiple hardware platforms ("VR Juggler," 2012). 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.

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 (Staadt, Walker, Nuber, & Hamann, 2003). 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.

NIfTI File Reader

4D volumetric data can be stored in various file formats such as, Analyze/SPM, MINC, AFNI, and NIfTI ("Neuroimaging Informatics Technology Initiative (NIfTI)," 2011). 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.

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 (Noon, 2012) 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 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 Error! Reference source not found..

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.

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.

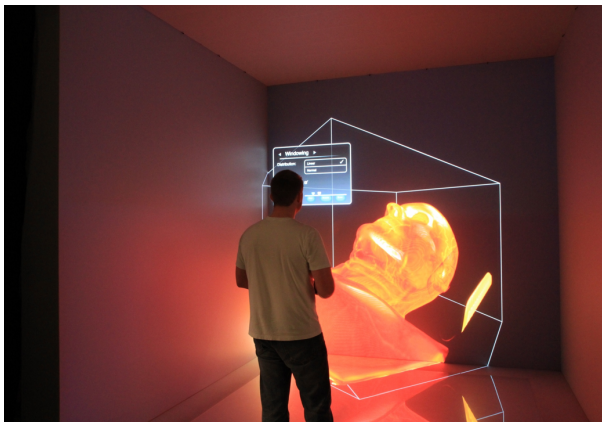

Figure 5. Immersive VR application in a 6-sided CAVE™ environment

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 5 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.

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 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 6.

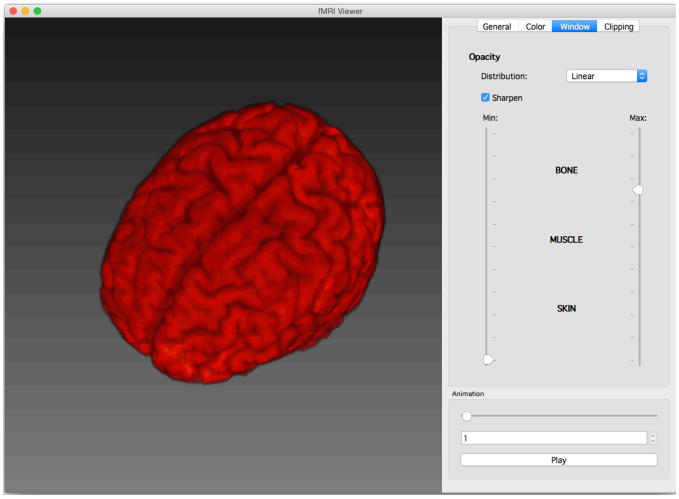

Figure 6. Desktop application visualizing a human brain

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 Intersense ultrasonic tracking system for tacking 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.

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

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 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 (*Overview United States Department of Defense Fiscal Year 2015 Budget Request Office of the Under Secretary of Defense (Comptroller)/ Chief Financial Officer*, 2014). 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.

REFERENCES

Botsch, M., & Kobbelt, L. (2003). High-quality point-based rendering on modern GPUs. *11th Pacific Conference onComputer Graphics and Applications, 2003. Proceedings.* doi:10.1109/PCCGA.2003.1238275

Bottomley, P. A. (1983). 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 Spectrum*, *20*(February), 32–38. doi:10.1109/MSPEC.1983.6369002

Chen, W. C. W., Ren, L. R. L., Zwicker, M., & Pfister, H. (2004). Hardware-accelerated adaptive EWA volume splatting. *IEEE Visualization 2004*. doi:10.1109/VISUAL.2004.38

Di Salle, F., Formisano, E., Linden, D. E., Goebel, R., Bonavita, S., Pepino, a, … Tedeschi, G. (1999). Exploring brain function with magnetic resonance imaging. *European Journal of Radiology*, *30*(2), 84–94. Retrieved from http://www.ncbi.nlm.nih.gov/pubmed/10401589

DirectX. (2015). Microsoft. Retrieved from https://support.microsoft.com/en-us/kb/179113

Engel, K., Kraus, M., & Ertl, T. (2001). High-quality pre-integrated volume rendering using hardware-accelerated pixel shading. In *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS Workshop on Graphics Hardware* (pp. 9–16). doi:10.1145/383507.383515

Hennemuth, A., Friman, O., Schumann, C., Bock, J., Drexl, J., Huellebrand, M., … Peitgen, H.-O. (2011). Fast Interactive Exploration of 4D MRI Flow Data, *7964*, 79640E–79640E–11. doi:10.1117/12.878202

Kaufman, A., & Mueller, K. (2005). Overview of volume rendering. In *The Visualization Handbook* (1st ed., pp. 127–174). Elsevier Inc.

Lorensen, W. E., & Cline, H. E. (1987). Marching cubes: A high resolution 3D surface construction algorithm. *ACM SIGGRAPH Computer Graphics*. doi:10.1145/37402.37422

Martinez Escobar, M., Juhnke, B., Hisley, K., Eliot, D., & Winer, E. (2013). Assessment of visual-spatial skills in medical context tasks when using monoscopic and stereoscopic visualization. In *SPIE Medical Imaging* (Vol. 8673, p. 86730N). doi:10.1117/12.2007087

Neophytou, N., & Mueller, K. (2005). GPU accelerated image aligned splatting. *Fourth International Workshop on Volume Graphics, 2005.*, 197–242. doi:10.1109/VG.2005.194115

Neuroimaging Informatics Technology Initiative (NIfTI). (2011). NIFTI Data Format Working Group. Retrieved from http://nifti.nimh.nih.gov

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

OpenGL. (2015). Khronos Group.

OpenSceneGraph. (2015).

Overview United States Department of Defense Fiscal Year 2015 Budget Request Office of the Under Secretary of Defense (Comptroller)/ Chief Financial Officer. (2014). Retrieved from http://comptroller.defense.gov/Portals/45/Documents/defbudget/fy2015/fy2015_Budget_Request_Overview_Book.pdf

Patel, D., Muren, L. P., Mehus, A., Kvinnsland, Y., Ulvang, D. M., & Villanger, K. P. (2007). A virtual reality solution for evaluation of radiotherapy plans. *Radiotherapy and Oncology*, *82*(2), 218–221. doi:10.1016/j.radonc.2006.11.024

Peng, H., Bria, A., Zhou, Z., Iannello, G., & Long, F. (2014). Extensible visualization and analysis for multidimensional images using Vaa3D. *Nature Protocols*, *9*(1), 193–208. doi:10.1038/nprot.2014.011

Peng, H., Ruan, Z., Long, F., Simpson, J. H., & Myers, E. W. (2010). V3D enables real-time 3D visualization and quantitative analysis of large-scale biological image data sets. *Nature Biotechnology*, *28*(4), 348–53. doi:10.1038/nbt.1612

Pfister, H. (2005). Hardware-Accelerated Volume Rendering. In *Visualization Handbook* (pp. 229–258). Elsevier. doi:10.1016/B978-012387582-2/50013-7

Pfister, H., Hardenbergh, J., Knittel, J., Lauer, H., & Seiler, L. (1999). The VolumePro Real-Time Ray-casting System. In *Siggraph 1999, Computer Graphics Proceedings,* (pp. 251–260). doi:10.1145/311535.311563

Raman, S. S., Osuagwu, F. C., Kadell, B., Cryer, H., Sayre, J., & Lu, D. S. K. (2008). Effect of CT on false positive diagnosis of appendicitis and perforation. *The New England Journal of Medicine*, *358*(9), 972–973. doi:10.1056/NEJMc0707000

Rezk-Salama, C., Engel, K., Bauer, M., Greiner, G., & Ertl, T. (2000). Interactive volume on standard PC graphics hardware using multi-textures and multi-stage rasterization. In *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS workshop on Graphics Hardware* (pp. 109–118). doi:10.1145/346876.348238

Rontgen, W. C. (1896). On a New Kind of Rays. *Science*, *3*(59), 227–231. doi:10.1126/science.3.59.227

Rosen, B. R., & Savoy, R. L. (2012). fMRI at 20: has it changed the world? *NeuroImage*, *62*(2), 1316–24. doi:10.1016/j.neuroimage.2012.03.004

Serra, L., Kockro, R. A., Guan, C. G., Hern, N., Lee, E. C. K., Lee, Y. H., … Nowinski, W. L. (1998). Multimodal volume-based tumor neurosurgery planning in the virtual workbench. *Medical Image Computing and Computer-Assisted Intervention — MICCAI'98*, *1496*, 1007–1015.

Shung, K. K., Smith, M. B., & Tsui, B. (1992). *Principles of Medical Imaging*. San Diego, CA: Academic Press, Inc.

Solomon, B. (2014). Facebook Buys Oculus, Virtual Reality Gaming Startup, For $2 Billion. Retrieved January 1, 2015, from http://www.forbes.com/sites/briansolomon/2014/03/25/facebook-buys-oculus-virtual-reality-gaming-startup-for-2-billion/

Staadt, O. G., Walker, J., Nuber, C., & Hamann, B. (2003). A survey and performance analysis of software platforms for interactive cluster-based multi-screen rendering. *EGVE '03: Proceedings of the Workshop on Virtual Environments 2003*, 261–270. doi:10.1145/769953.769984

VR Juggler. (2012). Retrieved from https://code.google.com/p/vrjuggler/wiki/WikiStart

Weiskopf, D. (2006). *GPU-Based Interactive Visualization Techniques*. *GPU-Based Interactive Visualization Techniques*. Springer Berlin Heidelberg. doi:10.1007/978-3-540-33263-3

Westover, L. (1990). Footprint evaluation for volume rendering. *ACM SIGGRAPH Computer Graphics*, *24*(4), 367–376. doi:10.1145/97880.97919

Wu, Y., Bhatia, V., Lauer, H., & Seiler, L. (2003). Shear-image order ray casting volume rendering. In *Proceedings of the 2003 symposium on Interactive 3D graphics - SI3D '03* (pp. 152–162). doi:10.1145/641508.641510