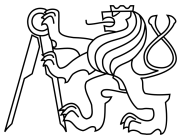
CZECH TECHNICAL UNIVERSITY IN PRAGUE
FACULTY OF INFORMATION TECHNOLOGY



ASSIGNMENT OF BACHELOR'S THESIS

Title:	3D Multimodal Visualization for Neurology Screenings
Student:	Tomáš Michalík
Supervisor:	Ing. Petr Ježdík, Ph.D.
Study Programme:	Informatics
Study Branch:	Software Engineering
Department:	Department of Software Engineering
Validity:	Until the end of summer semester 2016/17

Instructions

Design and implement an application for multimodal visualization of neurological examinations in 3D space. Minimal functional requirements are an ability to present CT, MRI and iEEG modalities in one 3D workspace as a model and standard planar presentation in coronal, axial, and transversal planes. The most important output is robustness of the application and its intuitive use in clinical praxis.

References

Will be provided by the supervisor.

L.S.

Ing. Michal Valenta, Ph.D.
Head of Department

prof. Ing. Pavel Tvrdík, CSc.
Dean

Prague November 24, 2015

Czech Technical University in Prague
Faculty of Information Technology
Department of Software Engineering

Bachelor's thesis

3D Multimodal Visualization for Neurology Screenings

Tomáš Michalík

Supervisor: Ing. Petr Ježdík, Ph.D

17th May 2016

Acknowledgements

I would like to thank to my supervisor Ing. Petr Ježdík, Ph.D for introducing me to this topic and also for his time.

Declaration

I hereby declare that the presented thesis is my own work and that I have cited all sources of information in accordance with the Guideline for adhering to ethical principles when elaborating an academic final thesis.

I acknowledge that my thesis is subject to the rights and obligations stipulated by the Act No. 121/2000 Coll., the Copyright Act, as amended. In accordance with Article 46(6) of the Act, I hereby grant a nonexclusive authorization (license) to utilize this thesis, including any and all computer programs incorporated therein or attached thereto and all corresponding documentation (hereinafter collectively referred to as the "Work"), to any and all persons that wish to utilize the Work. Such persons are entitled to use the Work in any way (including for-profit purposes) that does not detract from its value. This authorization is not limited in terms of time, location and quantity. However, all persons that makes use of the above license shall be obliged to grant a license at least in the same scope as defined above with respect to each and every work that is created (wholly or in part) based on the Work, by modifying the Work, by combining the Work with another work, by including the Work in a collection of works or by adapting the Work (including translation), and at the same time make available the source code of such work at least in a way and scope that are comparable to the way and scope in which the source code of the Work is made available.

In V Praze on 17th May 2016 .....

Czech Technical University in Prague
Faculty of Information Technology

Citation of this thesis

Michalík, Tomáš. *3D Multimodal Visualization for Neurology Screenings.* Bachelor's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2016.

Abstrakt

Tato práce se zabývá návrhem a implementací prohlížeče lékařských zobrazovacích technik pro virtuální realitu. Aplikace je implementována ve frameworku Unity3D s použitím Google Cardboard API. Externí část této aplikace je implementována jako doplněk do aplikace 3D Slicer. Hlavními prvky navržené aplikace jsou: vykreslování volumetrických dat, vykreslování více volumetrických dat zároveň, vizualizace iEEG, zobrazení standardních řezných rovin volumetrických modalit, základní manipulace s 3D modelem, načítání dat z aplikace 3D Slicer a základní grafické rozhraní pro virtuální realitu. Aplikace byla testována na telefonu se systémem Android a dále v editoru aplikace Unity3D v simulačním módu Google Cardboard.

Klíčová slova Vykreslování volumetrickách dat, vykreslování více volumetrických dat zároveň, lékařské zobrazovací techniky, virtuální realita

Abstract

This thesis is devoted to design and implementation of medical imaging viewer for virtual reality. The application is implemented in Unity3D framework with Google Cardboard API and its external part is implemented as a scripted extension for 3D Slicer. Key features are volume rendering, multi-volume rendering (e.g. MRI, CT), iEEG data visualization, multi-planar reconstruction of volumetric modalities, basic 3D model manipulations, data retrieval from 3D Slicer and basic GUI for virtual reality. The application was tested on Android phone and inside of Unity3D Editor with Google Cardboard simulation mode.

Keywords Volume rendering, multi-volume rendering, medical imaging, virtual reality

Contents

List of Figures

List of Tables

Introduction

The ultimate goal of this thesis is a design of medical imaging viewer for virtual reality which would support diagnosis and surgical operation planning in epilepsy surgery. The thesis is divided into two successive parts: the theoretical section and the practical section.

In the theoretical part, there are three topics related to this task. The first one is volume rendering which is the most crucial feature of the final application. The next one is an overview of medical visualization software. The last part is devoted to survey of virtual reality devices.

The practical section then consists of analysis and design, implementation and at the end the testing. The key features of the application are multi-volume rendering (multi-modalities visualization such as CT and MRI), invasive EEG data visualization, standard planar presentation of these modalities, basic 3D model manipulations, data retrieval from 3D Slicer and basic GUI for virtual reality. The last requirement is an application robustness and its intuitive interface suitable for clinical praxis.

The main application is designed in Unity3D framework with usage of Google Cardboard API. A data retrieval is delegated to 3D Slicer. This external part is implemented as a scripted module in 3D Slicer and it provides data content over HTTP connection.

The application is tested on Android phone and in Unity3D Editor with Google Cardboard simulator.

Motivation

Epilepsy is the most frequent neurological disorder. In population there is approximately 0.5 – 1 % of people who have had at least one epileptic seizure during the last five years. Standard medication treatment with antiepileptic drugs has positive results on majority of patients and often suspends seizures completely. The rest of affected people (20 – 30 %) are pharmacoresistant. A part of this group containing approximately 10 % of patients with pharmacoresistant (intractable, refractory) epilepsy can be treated with surgical intervention. This particular case needs supporting processes for diagnosis and surgical operation planning.

The techniques used for the purpose of epilepsy diagnosis and surgery planning are brain imaging and electroencephalography (EEG). There are following techniques of neuroimaging: computed tomography (CT), magnetic resonance imaging (MRI), functional magnetic resonance imaging (fMRI), positron emission tomography (PET) and single-photon emission computed tomography (SPECT). EEG is a monitoring method that detects electrical activity of brain using electrodes placed on a scalp. The invasive EEG (iEEG) is a variant of EEG with surgically implanted electrodes. All these methods are used for recognizing and precise localization of seizure onset zone.

Medical doctors traditionally use multi planar reconstruction (slices view) of volumetric data sets which are an output from neuroimaging methods. Another option is a 3D reconstruction - volume rendering - that improves spatial navigation inside of volume model. The next step in 3D graphic visualization for this purpose may be virtual reality (VR) which is now applicable thanks to high performance graphic hardware. A goal of this thesis is to explore and try to use advantages of virtual reality for medical volume rendering.

Treatment of epilepsy is a complex process that includes a broad spectrum of methods and techniques. Numerous specialists from various fields contribute to this and bring views from different perspectives. Visualization in VR brings a new piece into this puzzle with target goal to improve diagnosis, surgery planning and also communication between participant medical doctors and researchers.

Purpose of this work is a design of application that will support diagnosis and surgery planning in epilepsy surgery. Target users will be researchers and medical doctors from Intracranial Signal Analysis Research Group (ISARG). The greatest benefit of this research group is in synergy of medical and technical experts that brings immediate feedback from application on real data. With this profit, it is possible to push development prototypes quickly into production environment.

Chapter 1

Goals

The main goal is to design and implement an application primarily focused on volume rendering with adaptation into virtual reality. Key requirements are multi-volume support, visualization of vertex iEEG data (data from invasive electroencephalography) with projection to volume data and standard planar presentation of volumes. Graphic user interface should be intuitive and suitable for usage in clinical praxis.

Chapter 2

The Theoretical Section

This section deals with the following topics: Volume rendering, Medical visualizations and Virtual reality. The first part contains a summary of volume rendering techniques especially focused on direct volume rendering and multi-volume rendering. The second part provides an overview of medical imaging software commonly used in medical praxis; it also describes applications for virtual reality. Finally, the last part includes survey of virtual reality devices.

2.1 Volume rendering

2.1.1 General info

Volume visualization is a method of extracting desired information from volumetric data. Volume data set is three-dimensional, possibly time-varying, data are typically produced by sampling, simulations or modelling techniques. In general, volume visualization is a capture of 3D data into a single 2D image [2].

The sample of data is called voxel which is a quartet (x, y, z, v). Coordinates x, y and z keep the position of the voxel and v represent a value. The value could be binary data, multi-value or vector. Binary data type is a single integer – value 0 signifies background and Oi indicates presence of an object. In case of multi-value data, the value is some measurable quantity, for example color, density or heat. A value represented by vector holds information from more sources such as multiple modalities (CT, MRI, etc.). Time-varying data extend the basic quartet to quintet (x, y, z, t, v) where t adds time information [2].

Volume data are typically placed into regular grid of voxels. Isotropic layout consists of regular intervals along orthogonal axes. On the other side, anisotropic spacing between samples is also constant but each axis might have a different constant. From the implementation point of view, the data are stored as 3D array (also called volume buffer or 3D raster). An element

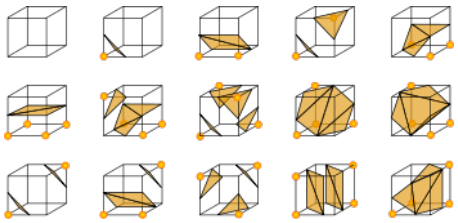


Figure 2.1: Marching Cubes algorithm cases [1]

location within the array corresponds with the coordinates. Mapping from the array to grid can be two types. The rectilinear grid is used when the cells are aligned to axes. If the transformation is non-linear then curvilinear grid is used. The logical organization, called computational space, is typically expressed by rectilinear grid and curvilinear grid matches with physical space. In general, the distribution of voxels can be also unstructured or irregular; in that case the connections between samples have to be explicitly defined [2].

2.1.2 Indirect volume rendering

Indirect volume techniques, volume surface rendering, transform volume data into polygonal mesh. Binary classification of whetheher the voxel belongs inside or outside of the object is based on threshold value. An example of this type of algorithm are Marching Cubes [3] which were designed for application on medical data sets. Thanks to transformation into geometric primitives, the final surface can be rendered using conventional graphic rendering algorithms.

The surface reconstruction with Cube Marching uses logical cubes where each of the eight vertexes represents a neighboring voxel value from source volume. The final surface is reconstructed according to the values of voxels. Each vertex of cube is possible inside or outside of the final object which implies 256 cases of different topologies how surface of the data from single cube can be shaped. The algorithm simplifies, due to symmetries, these cases to 14 patterns. See fig. 2.1. The advantage of isosurface visualization is definitely in rendering speed and this type of imaging is useful in many applications. On the other hand, surface based on geometric primitives is always only approximation of original data. More precise surface reconstruction may involve creation of excessive amount of primitives. Second disadvantage is the loss of information from interior of object [2].

Surface rendering is useful especially when we are interested in the shape of the data and there is not much noise similar to threshold value.

2.1.3 Direct volume rendering

Direct volume rendering methods visualize volume data without intermediate state of geometric representation. In surface rendering, the whole mesh has to be recalculated each time when the threshold value is changed. Direct volume rendering also consider semi-transparent data so the output image may represent whole depth of volume.

Direct rendering algorithms are divided in two groups – object-order and image-order. The first one, also called forward mapping, is performed voxel-by-voxel and volume data are mapped onto image plane. Examples of these methods are splatting, Shear-Warp [4] and slicing [5].

2.1.3.1 Object-order algorithms

Splatting takes each voxel and maps it onto projection plane in back-to-front order. Samples are splatted on each other to produce final image. The second algorithm Shear-Warp consists of three steps. At first, the viewing projection is transformed in a way that the voxels of volume are aligned with rows of pixels. In the second stage, distorted intermediate image is created and finally is executed 2D warp to produce output result. The last algorithm, the slicing, or also called texture based rendering, computes slices of volume which are then mapped as 2D textures onto planes parallel with projection plane. The final image is composed by blending possibly transparent slices together.

2.1.3.2 Image-order algorithms

On the other side, the image-order algorithms, called backward mapping, are using virtual scheme of casting rays in direction from camera through each pixel and the whole volume model to calculate final pixel color.

A representative of image-order algorithms is ray-casting, sometimes also called ray-marching. GPU-based ray-casting stores volume as single 3D texture. A ray through volume for each pixel is generated in fragment shader. The ray starts at the front face of bounding box and then continue until back face is reached [6].

Computational complexity is the main disadvantage of ray-casting but thanks to rapid development of graphic hardware during last years, is it possible to implement ray-casting on a common consumer GPU [5]. In comparison to other mentioned algorithms it provides the best image quality. Technological progress for example demonstrates the ability of interactive ray-casting implementation for web browsers using WebGL [5].

2.1.3.3 Rendering techniques

The three most common techniques of voxel values are maximum intensity projection (MIP), isosurface ray-casting and compositing. MIP technique selects

only the highest value of the voxel across single ray. The second mentioned, isosurface ray-casting, is similar but it takes the value of the first voxel which is above set threshold value. The last one accumulates values across ray from whole volume [5].

The final output image is typically customized by transfer function which is applied on value of a voxel. Function returns output color and opacity for each value. This allows the user to set transparent and semi-transparent areas which are just not in focus of interest. The transfer function often uses value of a voxel as an index into 1D array, a lookup table, which contents output color and opacity. This can for example distinguish organs or tissues in medical imaging.

Direct and indirect rendering are used in different applications. Indirect methods need to preprocess input volume into geometric primitives and extract only surface. Every time when the threshold value is changed than surface mesh must be recalculated. On the other hand, direct rendering is more complex itself but the output can be customized easily by changing transfer function; internal structure of volume may also be displayed.

2.1.4 Multi-volume data

An output information may be extended by adding multi-volume data. Final image is then composed from two or more volumes. As mentioned earlier in this section, voxel can beside single and multi-value data also hold composite data where each component of vector belongs to different source. Such as example can be more medical modalities, e. g., CT and MRI.

There are three different methods of mixing the separate data together. *Image-level intermixing* is the simplest one. Each component / volume is rendered individually as a scalar dataset and finally all the images are blended together. It could be done by weighting function depending, for example, on opacity channel. This method is not much valuable most of the times because it does not respect depth ordering of samples between components. This problem is solved by the second method called *accumulation level intermixing*. During every ray step, color and opacity are computed first and then are these values mixed together. The third method, *illumination model level intermixing*, varies from the previous one in the phase of mixing. Samples are combined first and color and opacity are computed afterwards [2].

When using the second method, mentioned in previous paragraph, volume mixing can be computed by using CPU or GPU. Merged volume model can be prepared on CPU and then rendered as single volume. Fast rendering is at the expense of flexibility and the whole volume must be recalculated when the mixing function is changed. On the other hand, if we use mixing on GPU, multi-volumes must be loaded separately which significantly increases memory requirements. During rendering the computational complexity of

fragment shader then increases accordingly. The choice of the type depends on the particular situation and on whether we prefer speed or flexibility.

Multi-volume rendering is generally used for studying connections and content dependencies across different volumetric data sets. Due to the ability of volume rendering to visualize internal structure, we can compare the relations of data from various sources and retrieve supplementary information.

2.1.5 Geometry intersection

In some cases, it may be required to display volumetric and polygonal data in a single scene. A final composed result must then respect correct visibility (depth) order. This effect can be achieved by a ray traversing through volume, which terminates on a surface of a polygonal object and which incorporates surface color into final color computation. When the traversing ray hits a non-transparent polygonal object, it terminates immediately and the surface color is added to the accumulated color; if the object is transparent, the ray continues [7].

Another indirect option is to convert polygonal data into volume data. This method requires preprocessing phase and memory complexity is typically increased; on the other hand, after this transformation, traditional multi-volume rendering techniques may be applied.

Geometry intersection techniques allow combining volume data with more common polygonal objects. And again, similarly to what was mentioned in connection with multi-volume rendering, the addition of geometric data can enrich an output information value.

2.1.6 Acceleration techniques of ray-casting

There are two basic types of ray-casting algorithm acceleration. The first type, early ray termination, monitors alpha value of merged samples. If the ray accumulating samples across volume reaches 100 % of alpha channel value, no other samples are visible from the current view angle. In that case traversing is terminated. Early ray termination accelerates rendering assuming low transparency of volume so that rays are shortened.

The second type, empty space skipping, is a more general topic but a principle is also straightforward. Fitness for the particular kind of data always depends on a data distribution. Examples include MRI or CT scan of head. The volume data are aligned with head size with bigger or smaller space overhead. Even if it fits the shape of the head perfectly, there is empty space in the corners of the bounding box. These areas will never be in the focus of interest in any transfer function configuration. There are several strategies how to skip these uninteresting voxels.

The first method requires additional volume containing information about data distribution. At the beginning, threshold value, which splits original

volume to groups of useful and useless voxels, is set. In next step, new volume containing value of distance to nearest surface (voxel with value above threshold) is generated for each cell. At coordinates containing useful value the distance is set to zero. With this improvement, the ray traversing is accelerated according to values from supplementary volume. The distance represents radius of sphere which does not contain any useful data. Ray-casting step size is then set up to this distance and no information above threshold is lost.

The second method reduces processed voxels by setting start and end point of each ray analytically during rendering. The key data of volume are descried with a simple shape; it can be for example a sphere or an oriented bounding box (OBB). In this way, the fragment shader can calculate the precise position of front and back face of interesting parts of volume and do not spend time processing empty space [8].

The third method works with mesh data that characterize interesting parts. To generate polygonal mesh, we can for example use the Cube Marching algorithm. The front and the back face of 3D shape are rendered into two z-buffers and then these depth values are used for the rendering of volumetric scene. Depths from z-buffers are mapped to world coordinates and then used as ray start and end positions [8].

The last method mentioned is the adaptive sampling. It is not only used for rendering acceleration, but also for quality improvement. The method also works with threshold value. The ray step length changes during traversing depending on voxel values. The basic model can contain two ray distances for useful and useless areas which are switched when the threshold value is overstepped. Other more advanced models can be designed to primarily improve quality [7].

Volume rendering acceleration techniques are used to reduce computational complexity which is typically high in volume rendering. The efficiency of acceleration mainly depends on the characteristics of volume data and its distribution. A particular technique used on unsuitable data set may even cause deceleration due to overhead of applied algorithm. This reason makes the choice of acceleration technique crucial for the final rendering performance.

2.1.7 Volume data illumination

Illumination models add simulation of real-world light with its physical properties to the 3D graphic rendering. For the volume rendering, the light interaction is computed for each voxel.

Phong illumination is the basic model often used in volume rendering due to its reduced complexity. It considers only direct illumination so the light interaction is not affected by other voxels in the scene. The computation of final voxel color depends on voxel position, gradient at current voxel position, voxel color assigned by transfer function and position of the light source.

The Phong model uses three following illumination types: diffuse reflection, specular reflection and ambient lighting.

Diffuse reflection simulates light reflection according to angle of incidence. The angle depends on the normalized voxel gradient that is computed on the basis of neighboring voxel values.

Specular reflection is also based on viewing angle but the specular color of voxel has to be also computed. The final result depends on an input parameter which influences light reflection on surface-like structures. With this we can setup small sharp or bigger smoother highlight.

In comparison with previous types, the ambient light is the easiest one to compute. Because the Phong illumination considers only direct lighting voxels with gradient facing away from light source may disappear from scene completely. This issue fixes ambient light, which does not depend on any spatial and gradient information. Final voxel color is influenced only by fixed ambient color common for all voxels.

More advanced lighting models also provide simulations of shadowing, reflections or light refraction. These effects are achieved by extending ray-caster with ray-tracing. Indirect lighting models significantly increase rendering complexity and because of the high complexity of direct volume rendering itself, it is usually not applicable in interactive software [7].

The illumination techniques are used for simulating of more or less realistic light conditions. A choice of concrete illumination model depends on target application and must primarily respect final rendering complexity which is significantly increased by applied lighting.

2.2 Medical visualizations

This section will at first present short overview of software used in clinical praxes and related research; then a brief description of application 3D Slicer will follow and in the last part will discuss the possibilities of virtual reality application in medical environment.

2.2.1 Medical imaging software

For this purpose, there the software solutions will be divided into two groups: commercial and open source applications.

Commercial solutions usually cover whole sets of features needed for particular tasks or provide complex workstations. Examples of these products are: AW VolumeShare (GE Healthcare), syngo.via (Siemens), PMOD (PMOD Technologies Ltd), Definiens (Definiens Inc.) and MimVista (MIM Software Inc.). Main advantages of this type of software are coverage of wide spectrum of tasks, professional customer support provided by their vendors and for some of them approval by the FDA (Food and Drug Administration) for

certain clinical tasks. On the other hand, this software type is not always affordable for academic research and usually there are no options for end users to extend these solutions with custom features.

The opposite side occupies a wide spectrum of open source software. These tools are usually built on top of libraries such as ITK (Insight Segmentation and Registration Toolkit) and VTK (Visualization Toolkit) which provide tools for image processing and visualizations. Examples of free software are: 3D Slicer, ClearCanvas, OstriX, BioImage Suit, VolView, MeVisLab and SCIRun. They typically provide API for developing additional extensions and plugins that enables utilization for a specific function [9] [5].

2.2.2 3D Slicer

One of the most used open software is 3D Slicer. It is a multi-platform, free and open source software, which provides tools for medical image computing and visualization.

This application merged previously separate projects focused on image visualization, surgical navigation and graphical user interface. The tool with the initially specific purpose for neurosurgery and analysis has been transformed during years of development into an integrated platform used in various fields.

The Slicer contains plenty of modules for certain tasks from registration, segmentation and annotating to volume rendering. Additional features may be developed with three different types of extension – command line interface (CLI), loadable module or scripted module. For the extensions including heavy computations, a loadable module type which is written in C++ is available. On the other hand, for prototyping and custom workflow development there is convenient Python scripted module.

3D Slicer with all its features and extensibility possibilities became useful tool for clinical research. A stability and future development is ensured by companies Isomics Inc., Kitware Inc. and GE Global Research and also still growing broad Slicer community [9].

2.2.3 Virtual reality in medical environment

Virtual reality (VR) is rapidly growing field that is mainly accelerated by progress in graphic hardware performance during last years. Although VR devices and software development are mainly powered by entertainment industry, there are many opportunities of adapting VR for serious tasks for various fields. More information about virtual reality will be presented in the next section.

In medical environment, there are several fields where VR has already been used for years. The following list provides overview of such usages: exposure therapy [10], treatment for post-traumatic stress disorder (PTSD)

[11], brain damage assessment and rehabilitation [12], social cognition training for patients with autism [13] and generally surgery. In a surgery there are three main approaches, it is surgical training, diagnosis and surgical planning. All above mentioned examples give positives results and they contribute to enhancement of the quality of healthcare.

In neurosurgery, traditional multi-planar views of modalities are typically used besides the 3D model which does not increase information value a lot. This state of art may change with virtual reality where the user is directly inside of scene. Software for this particular application is for example developed by Surgical Theater [14], a company that which integrates VR with the 3D surgery navigation device called SNAP. According to University of California Los Angeles, which uses their software in neurosurgery for diagnosis of brain tumor, it significantly speeds up tumor localization and overall diagnosis process [9].

Applications like one from Surgical Theater bring new possibilities into medical environment and on the other hand it also opens new challenges on the technological side. The positive results of existing solutions show its usefulness which should be increased with a future development.

2.3 Virtual reality

"Virtual Reality is an alternate world filled with computer-generated images that respond to human movements. These simulated environments are usually visited with the aid of an expensive data suit which features stereophonic video goggles and fiber-optic data gloves." (Coates, 1992) [9]

VR is the next step in consuming of 3D graphic. It places the user in the middle of a virtual environment and with specialized controllers provides believable simulation. The VR device consists of a head-mounted display (HMD), which provides stereoscopic view of the virtual scene. It is one display split in half or two separate displays. The correct geometry of scene is calculated for both eyes in order to provide 3D illusion. The device traces the position of the user's head and projects the head rotations (and sometimes also spatial position in reality) into the virtual scene. The movements are tracked by built-in gyroscope or by external motion tracking cameras.

Interaction with the virtual scene is controlled by classical gamepads, specialized controllers or for example by the Leap Motion [15]. Specialized controllers similarly to HMD include rotation and position trackers and some of them also provide haptic feedback. Leap Motion is a contactless controller, which traces area up to one meter, typically user's hands, and recognizes hands and fingers inputs.

VR devices will be presented in two groups for this purpose. The first one covers professional HMD devices and the second one includes Google Cardboard and Cardboard-like products.

2.3.1 VR devices

2.3.1.1 Google Cardboard

Google Cardboard is a virtual reality (VR) platform for mobile devices. It was introduced at the Google I/O 2014 with its open software toolkit. A simply shaped cardboard transforms phone into a VR headset. The cardboard cover includes two lenses which allows the user to focus on the smart phone display. The built-in gyroscope in phone traces user's head orientation and projects these rotations into the virtual scene [16].

The Cardboard SDK now supports Android 4.4+, iOS 8+ and modern web browsers. SDK is available for Android, iOS and Unity3D. The first version of Google Cardboard from 2014 is compatible with phones up to 5.7 inches and it is controlled by a magnetic button. The second updated design was released is 2015. It supports phones up to 6 inches and the magnetic button is replaced by a conductive lever which ensures the touch input with display.

This category also includes other devices based on the same principle. An example is Samsung Gear VR which is a device developed by Oculus and it which works with some Samsung smartphones. In comparison with Google Cardboard, it provides more advanced optical system and input is handled with a touch pad placed on the side of the device.

2.3.1.2 Professional head mount display (HMD) devices

More advanced devices providing virtual reality in higher resolution and with better response work in connection to computer. The currently available devices are Oculus Rift and HTC Vive.

Oculus Rift Oculus Rift is a device developed by Oculus VR, which was released in March 2016. It has two OLED displays with 1080x1200 resolution per eye, 90 Hz refresh rate and approximately 110° field of view. The spatial motion tracking is solved by wireless sensor which tracks infrared LEDs placed on headset [17].

HTC Vive HTC Vive was released in April 2016 and has the same parameters in resolution, refresh rate and field of view. The motion detection in based on laser position sensor. Compared to Oculus Rift, HTC Vive includes an additional front-facing camera that makes the real world accessible from the virtual one [18].

2.3.1.3 Comparison

The two mentioned groups represent different target usage. Google Cardboard platform makes virtual reality accessible for broad spectrum of users thanks to low acquisitions and minimal requirements for compatible phones. On

the opposite side stand Oculus Rift and HTC Vive which offer immersive experience thanks to high resolution and refresh rate. High requirements for computer performance also enable visualization of graphic scenes based on heavy computations. Apart from the platform and performance differences, it has considerably divergent interface for controlling virtual scene. The Google Cardboard and similar gadgets are controlled by a single button or a touchpad so the interface design is quite a specific task. Another difference lies in users tracking. The Google Cardboard offers only head rotation tracking while Oculus Rift and HTC Vive also support real-world spatial position tracking which allows users to move in the virtual scene with their own movements.

Chapter 3

The Practical Section

In this section is presented analysis, design, implementation and testing of a designed application.

3.1 Analysis and design

3.1.1 Requirements

Functional requirements

- Single modality visualization
- Multi modalities visualization into one 3D workspace
- Vertex iEEG data projection into volume modalities
- Standard planar presentation of volume modalities
- Rendering output customization using linear transfer function
- 3D model manipulations

Non-functional requirements

- Implementation in framework Unity3D
- NIFTI raw data file format support
- Support of VR interface
- Intuitive GUI for medical doctors

Single modality visualization

Modalities MRI and CT are provided in the form of volumetric data set. Volume data consist of numerical floating point scalar values.

Multi modalities visualization into one 3D workspace

The key requirement is to increase information value of output image by combining different modalities together and displaying them into a common 3D workspace. Different modalities, which belong to one patient, are registered to the same coordinates system and have the same spacing and rotation.

Vertex iEEG data projection into volume modalities

Data from the invasive electroencephalography (iEEG) are produced by 120 or more electrodes. Data sets contain coordinates and array of values for each electrode. The coordinates are registered with volume data coordinates.

Standard planar presentation of volume modalities

Medical volume data are typically presented as three planar slices, called multiplanar reconstruction (MPR). Coronal, axial and transversal planes are mutually orthogonal slices.

Rendering output customization using linear transfer function

Volume rendering must be customizable with the linear transfer function which sets brightness and contrast of data.

3D model manipulations

Volume rendering must support basic manipulating functions – rotation, zoom and volume crop.

Implementation in framework Unity3D

Application must be implemented in framework Unity3D [19].

NIFTI raw data file format support

Volume data are saved in unified NIFTI-1 format. Each volume is stored with metadata in a single file.

Support of VR interface

Application should support virtual reality GUI for displaying volume data as well as multi planar reconstruction.

Intuitive GUI for medical doctors

Target users of the application are technical researchers and medical doctors. The graphic user interface must satisfy a following requirements: intuitive user controls, corresponds with classical arrangement of MPR and flexible layout.

3.1.2 Domain model

This section is identified the main domain entities related to the topic of this thesis.

Volume data

Volume data represent neurological examinations such as CT, MRI, etc. Raw data are provided as array of scalars. Voxel is represented as a single floating point number. Multi-volumes data from one patient are resampled and share the same slice spacing, spatial dimensions and rotation.

Vertex data

For this purpose, there are iEEG data provided in form of 3D coordinates and 1D array of values. The array represents the time progress of the value measured by a single electrode.

Scene

The scene consists of several objects representing modalities. The first one is a 3D model of single or multi volume data.

Planar views

Planar view is a single slice of volume data. Medical software standardly work with three orthogonal planes - coronal, axial and transversal.

3.1.3 Possible solutions

The following section presents the possible solutions of implementation; for each there is described one which was chosen.

3.1.3.1 Volume rendering

Volume rendering can be implemented by using image-order or image-order algorithms. For the purpose of medical imaging, we chose ray-casting due to its high quality and flexibility of output customization.

3.1.3.2 Multi-volume rendering

In multi-volume rendering, there are also few types of volumes composition: image-level intermixing, accumulation level intermixing and illumination model level intermixing. We considered only the last two types because they respect depth ordering. Finally, we chose the illumination model level intermixing because of simplification, this method performs color and opacity computation only once.

Volume samples mixing during multi-volume rendering can be executed using CPU or GPU. The composited volume on CPU is rendered fast but interactivity is low. This is a good choice when we already know an ideal rendering configuration. On the other hand, GPU based mixing significantly increases the rendering complexity of fragment shader, where the mixing is performed, but it keeps high flexibility of mixing function. This type is suitable for situations where the rendering customization is a key feature.

3.1.3.3 EEG data visualization

EEG data can be represented as geometric objects or as a volume data set. A generation of geometric object would be simple and fast, but it would require front and back face depth test during rendering. Based on this motive, we chose generation of volumetric data and for rendering, we used common multi-volume rendering technique.

3.1.3.4 Data loading

Data loading can be handled in several ways. The key requirement is the NIFTI file format support which can be loaded directly or with assistance of other application. Delegation to other software was on the one hand a complication, in the way of other framework and programming language usage, but on the other hand it simplified the VR application itself. This solution excludes file system structure exploration from the VR environment and it also provides better integration with the target environment where is this application (Slicer) used.

3.2 Implementation

3.2.1 The big picture

The entire application is divided into two components. The bigger one is implemented in C# using Unity3D framework [19], hereafter referred to as the *UnityApplication*. The smaller one is implemented in Python as a 3D Slicer [20] scripted module, referred to as the *UnityConnector*. Applications are linked together with HTTP connection in form of the client-server topology. Overall structure is in figure 3.1.

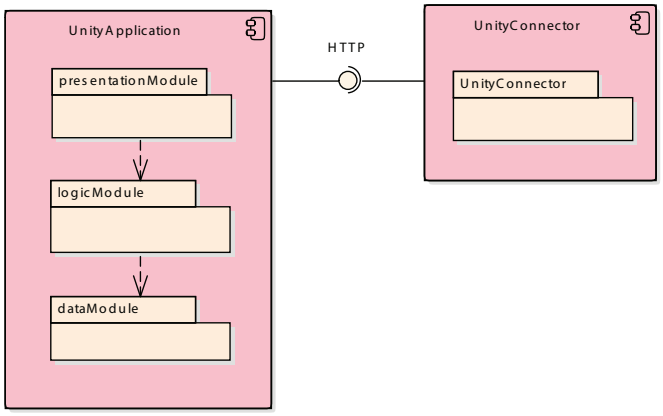


Figure 3.1: Component diagram of UnityApplication and UnityConnector

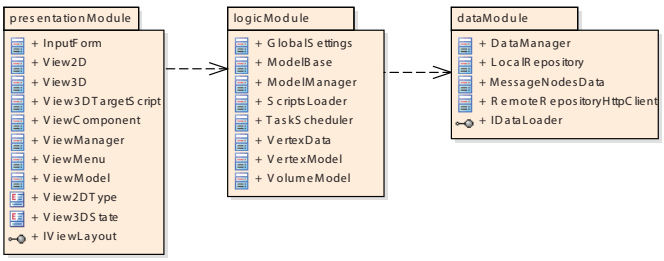


Figure 3.2: Class diagram of packages

This distribution takes advantage of the existing Slicer application and its various file formats support. This software is also a tool commonly used in clinical praxis and medical research. The researchers from ISARG use Slicer as one of their tools for analysis and surgery planning that increases integration ability of this application.

3.2.2 Unity3D application

3.2.2.1 Architecture model

Unity application is split into three logical modules 3.2. Each module is mainly controlled by its manager class which is *ViewManager* for the presentation module, *ModelManager* for the logic module and *DataManager* for the data module.

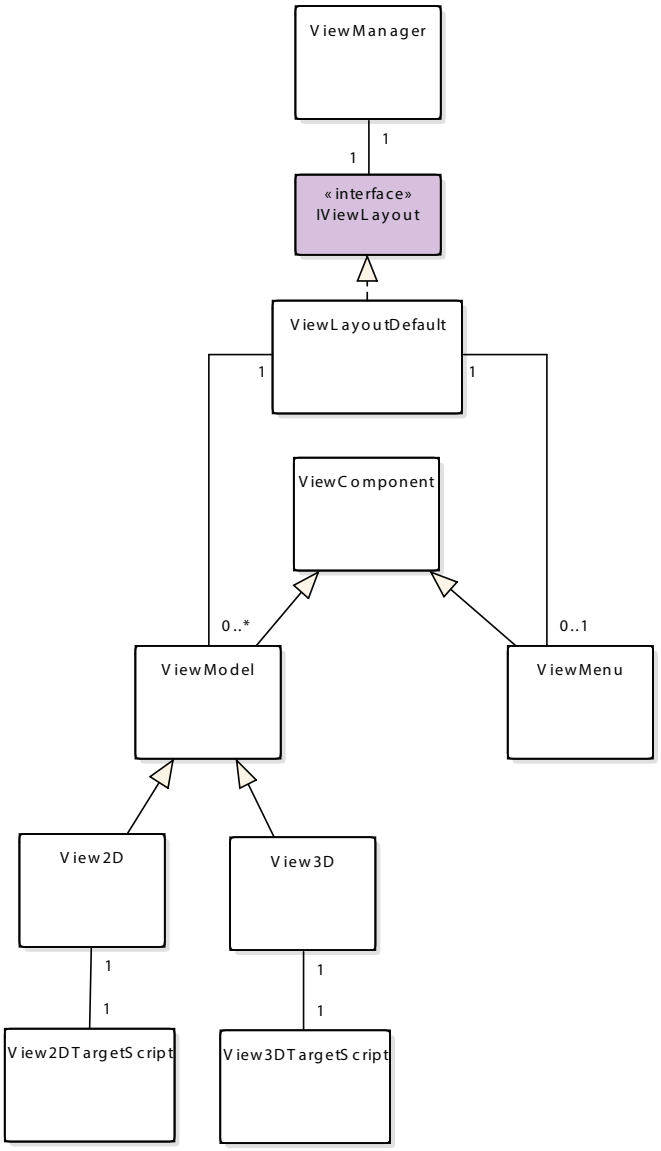


Figure 3.3: Class diagram of presentation module

Presentation module The presentation module covers everything connected with GUI. The structure of the module is captured in figure 3.3. *ViewManager* class is implemented as a singleton. It holds current layout structure and because Unity uses a single rendering thread, there is no need of more instances per application.

Individual elements are arranged in hierarchical structure implemented

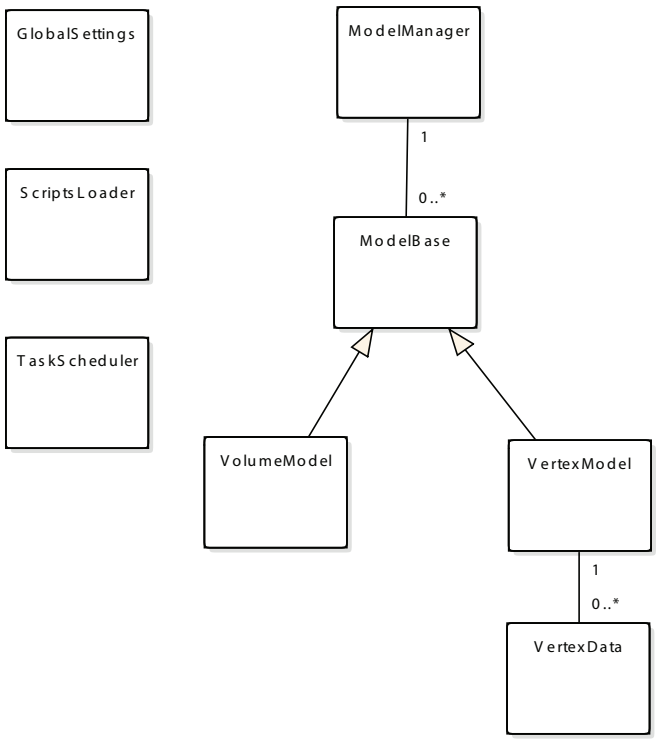


Figure 3.4: Class diagram of logic module

by using inheritance. To each class of graphic component belongs another class named "class name + TargetScript" which inherits from Unity class *MonoBehaviour*. This allows the script to execute per frame updates.

Logic module Just as in the presentation module, there is also a main manager class, see fig. 3.4. *ModuleManager* is implemented in the same way as singleton and it keeps instances of all the models in the application.

VolumeModel class represents single volume data. It loads array of scalars in binary format and then it generates 3D texture.

VertexModel class represents data set produced by iEEG. Data from single electrode are stored as *VertexData* types. Vertex data are transformed into volume data by generating 3D texture where sphere of constant radius is placed around coordinates of each electrode. The texture then contains zero value at coordinates where no sphere is present or ordinal number of electrode. The measured values of the electrodes are mapped to colors and used in form of a 2D texture. This class accepts the already encoded raw data in color format. Rows of this texture represent data from single electrode and columns stand for value progress in time. The first whole row contains fully transparent color

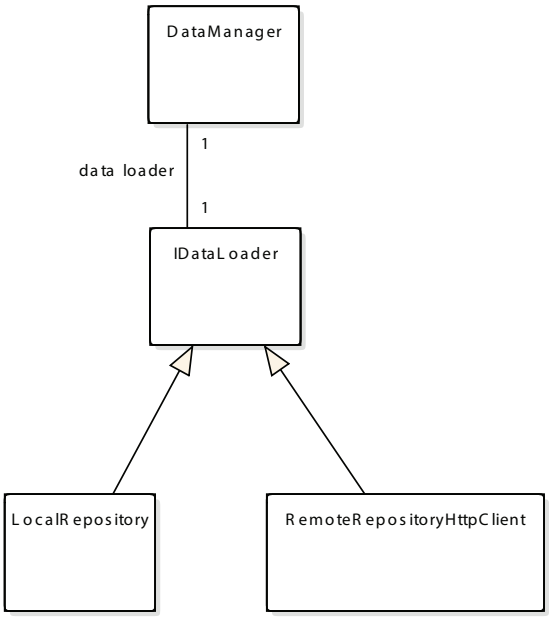


Figure 3.5: Class diagram of data module

to indicate absence of data from electrodes.

GlobalSettings static class holds default configuration and it also keeps parameters between switching of scenes.

TaskScheduler is a class handling scheduled tasks that were added from asynchronous callbacks. Classes from Unity framework are not thread safe and it is not allowed to call methods on those objects from non-main thread. This class provides functional connection between main and non-main threads. Callbacks of asynchronous methods insert required commands in a way of delegate method for later execution. In periodic intervals the class checks whether there are any waiting tasks. This mechanism ensures execution of methods requiring to be run from the main thread.

Data module Data module 3.5 has also its *DataManager* class implemented as singleton. It is a connection point to the data tier of the application. DataManager always binds two separate connections for data loading and data writing. The current setup is loaded from *GlobalSettings* class from logic module at the start of the application.

Data are loaded from the local repository or from the Slicer application. The local repository stores serialized examples saved as binary files.

In a mode of remote repository, the connection to Slicer is made with HTTP connection.

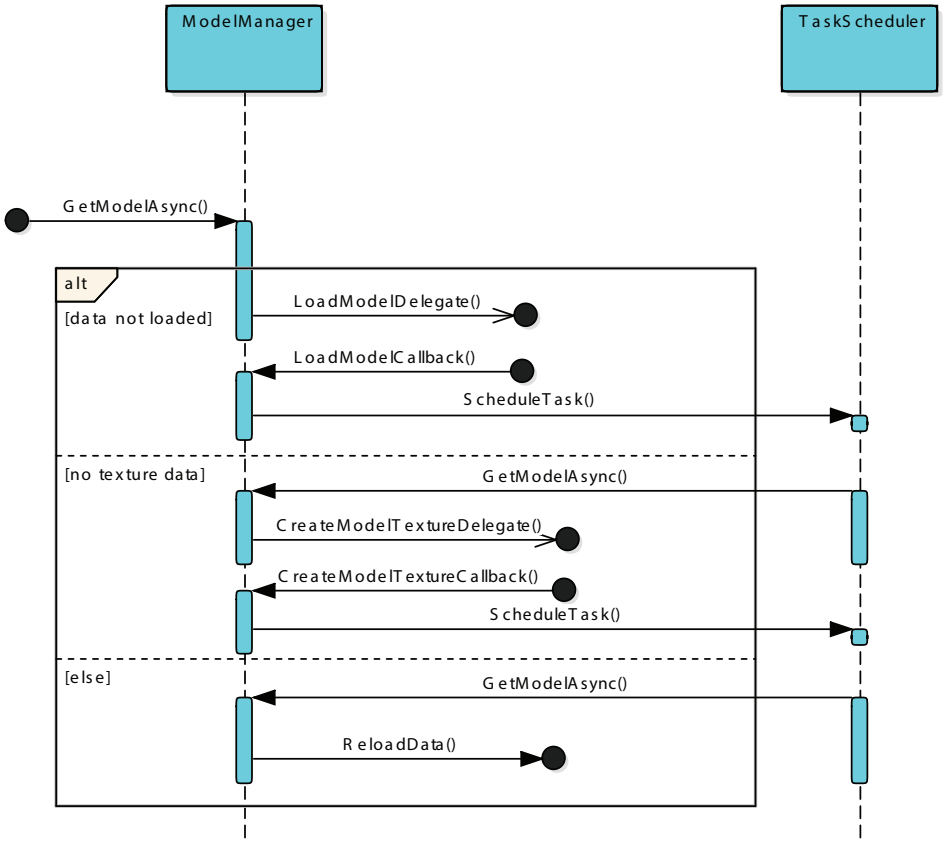


Figure 3.6: Sequence diagram of asynchronous data loading

3.2.2.2 Asynchronous data loading

Data for model are loaded asynchronously in three phases, see fig. 3.6. In the first phase, the raw data are loaded from data module. Before the second and the third phase, some methods are delegated to *TaskScheduler*, which ensures their execution in the main thread. The second phase generates texture from raw data and finally in the third phase, a method from view module is called to reload data.

3.2.2.3 Project structure

The folder structure in the Unity project at top level corresponds with a default one. In the Assets folder there are folders named App, Cardboard, Editor, Scenes and also Materials and Plugins. All classes and scripts developed for the purpose of this application are placed in the App folder and the folders are further subdivided accordingly to the names of the modules. The Cardboard folder contains Google Cardboard SDK; the Editor folder contains all

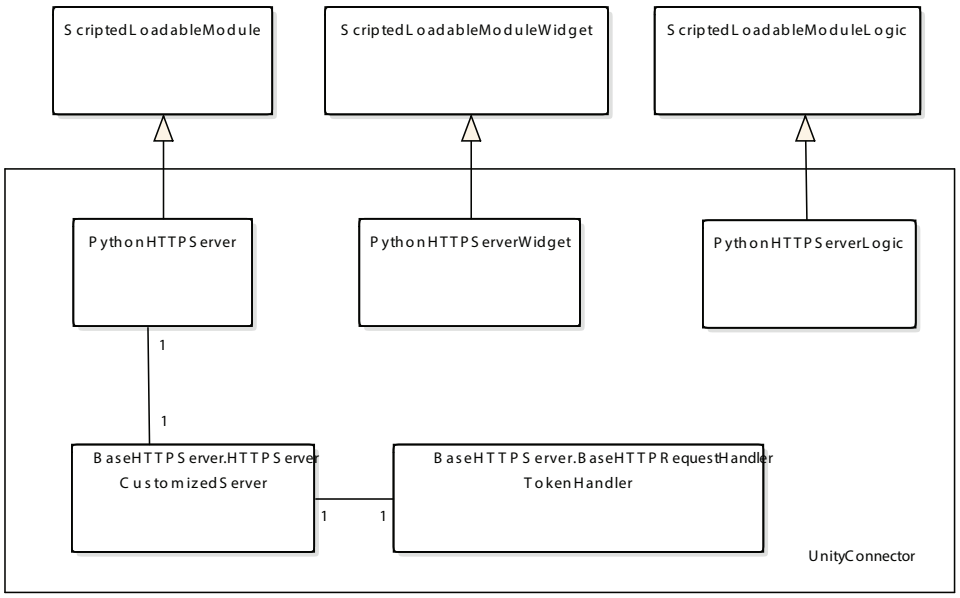


Figure 3.7: Class diagram of 3D Slicer extension

automatic tests.

3.2.3 Slicer extension

Slicer extension is developed as a scripted module for Slicer 4.5. It follows a predefined structure of classes *ScriptedLoadableModule*, *ScriptedLoadableModuleWidget* and *ScriptedLoadableModuleLogic*, see figure 3.7. The class *ScriptedLoadableModule* ensures correct module loading in the Slicer application. The elements of GUI are set in the *ScriptedLoadableModuleWidget* and there are also established connections to *ScriptedLoadableModuleLogic* where function calls are realized.

The HTTP server itself is implemented in class named *CustomizedServer* and the request handler in class *TokenHandler*. Due to internal architecture of Slicer, it is not allowed to create new threads in scripted modules directly. Server instance is initialized synchronously and in the start method, server notify method is bound to *qt.QSocketNotifier*. This connection ensures run of the server realized by Qt library and notification on each network request.

3.2.3.1 Function

The extension works in two states – server stopped and server running. It is controlled with a simple form which consists of two input fields for server bind IP address and port. Server is then stopped and started with a single button.

3.2.4 Connection

Connection between the applications *UnityApplication* and *UnityConnector* is realized with a HTTP client-server architecture. There are two types of HTTP GET requests. The first one requests list of volume data currently loaded in the Slicer application. A response is send in a JSON format and for each scalar node (internal representation in Slicer) includes its name, model dimension and minimum and maximum scalar value. The second request applies for a single model data. Volume data are encoded to base64 string and vertex data (data from iEEG) are sent in JSON format.

3.2.5 Final solution

The solution consists of a UnityApplication developed in Unity3D and a Unity-Connector which is scripted extension for the 3D Slicer. The connection between applications is dealt with the HTTP connection. Data are transferred in a JSON format and binary.

The UnityApplication is an implementation of medical imaging viewer in virtual reality. The implementation uses Google Cardboard API for Unity3D framework. The main feature of the application is a multi-volume renderer which can display one or two modalities in a common 3D space. Besides multi-volume rendering, it supports dynamic iEEG data visualization with projection into volume modalities. Another feature is a multi-planar reconstruction (slice view) of volume data.

The virtual scene supports following general actions: - loading of available models list - loading of volume model - loading of iEEG data.

Supported actions for 3D model are: - model rotation - model crop - model opacity setup - settings of low or high rendering quality.

Supported actions for 2D model views: - slice position setup within model

Local and remote mode The application includes two modes: the local and the remote. A switch between those modes is provided in an input form which is displayed on the application startup. The local mode uses data serialized binary files stored in a resources folder. The remote mode than uses data loaded from Slicer application.

3.2.5.1 Interface design

The graphic user interface was in this phase implemented only in schematic version. Because the target platform will not be Google Cardboard, the non-user friendly controlling was omitted. A future GUI design will be related with some specific controller. See fig. 3.8.

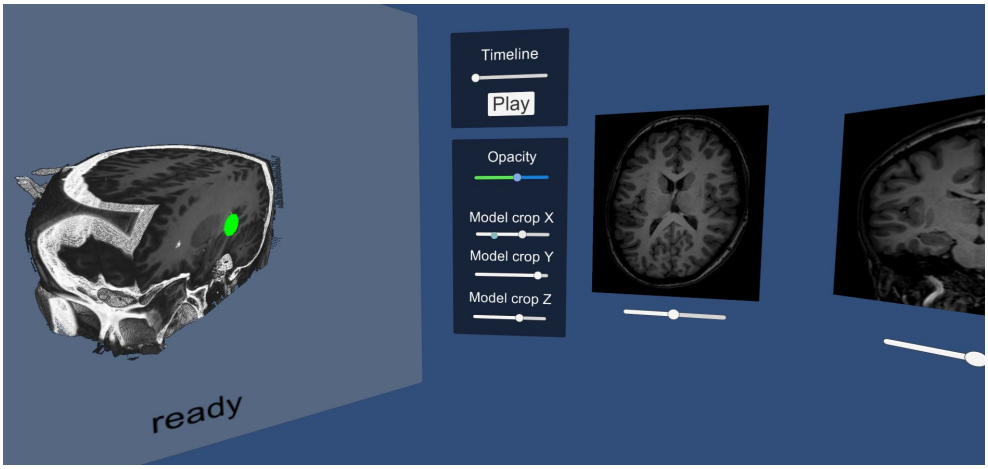


Figure 3.8: UnityApplication GUI

3.2.5.2 Known problems in the implementation

Both applications were developed in Windows OS environment but during development of Slicer extension it has appeared an error which was not successfully fixed. It is an error in socket notification which raises when handling incoming connection from client. This error has not raised in second testing environment on Ubuntu 15.04.

3.2.5.3 Not implemented requirements

In this phase, there are yet not implemented following requirements which were set at the beginning: linear transfer function and 3D model zoom. Can be run on android device or inside of Unity3D Editor with Google Cardboard simulator.

3.2.5.4 Deployment

The *UnityApplication* is built for Android 6.0 (API level 23) as a single APK file. The installation consists of file transfer into phone and simply installation. The *UnityConnector* is deployed in a folder containing all files of extension. In a Slicer it must be then added additional path to this module. Detailed installation tutorial is described in an appendix. See fig. 3.9.

3.2.5.5 Input data requirements

Multi-volume rendering in this application requires the same dimension, slice spacing and rotation for all models. Alternatively, it can render only one model in the scene.

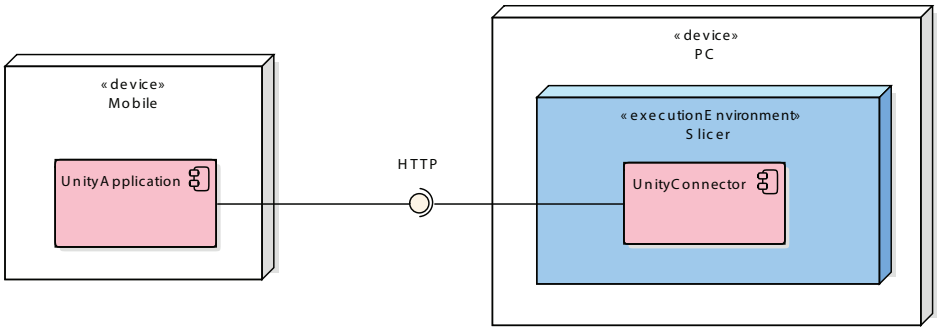


Figure 3.9: Deployment diagram

3.3 Testing

The designed applications were tested in two ways. The first one was automatic testing and the second were usability tests.

3.3.1 Testing environment

- Ubuntu 15.04, Slicer 4.5 (Slicer extension testing)
- Windows 8.1, Unity3D 5.3.4f1 (UnityApplication testing in Unity3D Editor)
- LG Nexus 5X, Android 6.0.1, Google Cardboard V2

3.3.2 UnityApplication automatic tests

Automatic testing of UnityApplication was made with NUnity, which is a unity testing framework. Unity3D enables running these tests inside of the Unity Editor, so all features from Unity are available. For each class in application was created a test class with appropriate tests.

3.3.3 Usability testing

Both applications together were tested by five persons. They were chosen on the following criteria: they are not IT specialists and they do not have any experience with virtual reality devices. This sample of users should sufficiently simulate target users.

There were tests with sequential instructions. The first one was focused only on testing of UnityApplication and its local repository mode. The second one then tested integration of both UnityApplication and UnityConnector.

Test users got a prepared testing environment with installed application, orally presented instructions about controlling UnityApplication in VR and lists with following scenarios.

3.3.3.1 Test 1

- Run the UnityApplication on the phone.
- In an input form, select option "local repository".
- Put the phone into Google Cardboard.
- In the left part of virtual scene select "Load list".
- In the panel with the title "Models", list of models will be displayed.
- S select "volumeTest1" in the model list.
- Wait until the model is loaded; it will be displayed in front of you.
- Try rotation of the model.
- On the right of the model there are sliders for cropping. Try to customize the one labeled "Model crop x" and set slider approximately to the half of its range.
- Above the crop tools, there is another slider with the label "opacity".
- Look again at the model and check whether the model has been changed after last two steps. Only a half of model should be displayed and internal structure of test data should also be visible.
- In the model list select "volumeTest2" and also "vertexSample". Wait until the data will be loaded. In the right menu click on the play button and watch a color animation in 3D model.
- On the right side of cropping tools, there are three monitors of model planar view. Below each of them there are sliders. Try to move them and test the whole range. Visualization on the monitors should be changing accordingly to positions of the sliders.
- Close the application with a button on the virtual ground.

3.3.3.2 Test 2

- On a computer in the Slicer application, run the server with the "Start server" button.
- Run the UnityApplication on phone.

Table 3.1: Performance test with 128x128x128 volume data sets

	empty scene	single volume	two volumes	two volumes + color animation
PC	80 fps	75 fps	72 fps	70 fps
phone	60 fps	25 fps	20 fps	8 fps

- In the input form, select the "remote repository" option and fill server address and server port according to instructions.
- Put phone into Google Cardboard.
- Select "Load list" in the left part of the virtual scene
- Select "rCT" in the model list.
- Wait until the model is loaded; it will be displayed in front of you.
- Try rotatingthe model.
- Close the application with the button on the virtual ground.

The testing discovered several bugs and recommendations. It also proved that the steps of testing scenarios were not enough described and this drawback involved necessity of detailed description for nearly each step. Some of those misunderstandings were caused by the schematic nature of the GUI which was not the main focus of interest of this thesis.

Almost all testers recommended highlighting all buttons in the scene to accentuate its interactivity. Another reported issue was the unclear status displaying. The status, showing whether the application is loading something or whether it is ready, is displayed below the 3D model and is not visible after the model selection. Other bugs were small error message in the input form and also the not function into "quit" button.

All reported bugs were fixed in the implementation and status label redesign was scheduled for the next development phase.

3.3.4 Performance testing

UnityApplication was tested on phone and in Unity3D Editor with Google Cardboard simulator. Both tests were aimed at refresh rate, which was displayed in the middle of the view. The refresh rate during testing was monitored in following situations: empty scene, single volume rendering, multi-volume rendering (two volumes), multi-volume rendering (two volumes) with displaying vertex data. Results from this test are in a table 3.1.

Conclusion

The main purpose of this thesis was to design a medical imaging viewer for virtual reality and it was successfully fulfilled. The current implementation of the application is a rough development preview but it has met all the main requirements which were set in the beginning. Despite broad spectrum of topics which are related with this thesis it was successfully brought to the objectives.

The application is implemented in an environment of the Unity3D framework with usage of Google Cardboard SDK. The key features which were successfully adapted into implementation are:

- volume rendering,
- multi-volume rendering,
- dynamic visualization of vertex iEEG data and its projection into other volume modalities,
- the visualization multi-planar reconstruction,
- basic 3D model manipulations,
- data retrieval from 3D Slicer which ensures broad file format support,
- basic GUI for virtual reality.

Only the last requirement, intuitive graphic user interface suitable for clinical praxis, was not entirely fulfilled. The GUI was implemented only in a very schematic form because the main emphasis was placed on volume rendering.

Beside the main application designed in Unity3D framework, an extension for Slicer 3D application was developed. It manages data content loaded in Slicer 3D and make it accessible over HTTP requests.

The current implementation was tested on an Android phone and inside of Unity3D Editor using Google Cardboard simulator mode. Due to high

computational complexity of multi-volume rendering it was not possible to test all features on a mobile device with test data in full resolution.

Future plan

The main goal of the application is a future adaptation for any virtual reality platform such as Oculus Rift or HTC Vive and its deployment to clinical research environment. The main improvement requirement is a redesign of current graphic user interface. Other features to be developed are: any acceleration technique of volume rendering, separate mode for technical researchers and medical doctors, volume rendering transformation functions and volume model color mapping.

Bibliography

[1] Dostupné z: https://graphics.stanford.edu/~mdfisher/MarchingCubes.html

[2] Charles D. Hansen, C. R. J.: *The Visualization Handbook*. Academic Press, 2005, ISBN ISBN: 978-0-12-387582-2.

[3] Lorensen, W. E.; Cline, H. E.: Marching Cubes: A High Resolution 3D Surface Construction Algorithm. *SIGGRAPH Comput. Graph.*, ročník 21, č. 4, Srpen 1987: s. 163–169, ISSN 0097-8930, doi:10.1145/37402.37422. Dostupné z: http://doi.acm.org/10.1145/37402.37422

[4] Accomazzi, V.: Perspective with shear warp. Duben 18 2006, uS Patent 7,031,505. Dostupné z: http://www.google.ch/patents/US7031505

[5] Hlaváček, J.: *GPU-accelerated processing of medical data*. 2008.

[6] Hadwiger, M.: *Real-Time Ray-Casting and Advanced Shading of Discrete Isosurfaces*. ETH Zürich, 2005.

[7] Hadwiger, M.; Ljung, P.; Salama, C. R.; aj.: Advanced Illumination Techniques for GPU-based Volume Raycasting. In *ACM SIGGRAPH 2009 Courses*, SIGGRAPH '09, New York, NY, USA: ACM, 2009, s. 2:1–2:166, doi:10.1145/1667239.1667241. Dostupné z: http://doi.acm.org/10.1145/1667239.1667241

[8] Jamriška, O.: personal communication.

[9] Fedorov, A.; Beichel, R.; Kalpathy-Cramer, J.; aj.: 3D Slicer as an image computing platform for the Quantitative Imaging Network. *Magnetic Resonance Imaging*, ročník 30, č. 9, 2012: s. 1323 – 1341, ISSN 0730-725X, doi:http://dx.doi.org/10.1016/j.mri.2012.05.001, quantitative Imaging in Cancer. Dostupné z: http://www.sciencedirect.com/science/article/pii/S0730725X12001816

[10] Virtual Reality Therapy for Phobias. Dostupné z: http://psychiatry.duke.edu/divisions/general-psychiatry/virtual-reality-therapy-phobias

[11] A Virtual Reality Exposure Therapy Application for Iraq War Military Personnel with Post Traumatic Stress Disorder: From Training to Toy to Treatment.

[12] ROSE, F. D.: Virtual Reality in Brain Damage Rehabilitation: Review. 2005.

[13] ROSE, F. D.: Virtual Reality Social Cognition Training for Young Adults with High-Functioning Autism. 2005.

[14] Surgical Theater. Dostupné z: http://www.surgicaltheater.net/

[15] Leap motion. Dostupné z: https://www.leapmotion.com/

[16] Google I/O 2014, Cardboard: VR for Android. Dostupné z: https://www.google.com/events/io/schedule/session/603fe228-89c5-e311-b297-00155d5066d7

[17] Oculus Rift. Dostupné z: https://www.oculus.com/

[18] HTC Vive. Dostupné z: https://www.htcvive.com/

[19] Unity3D. Dostupné z: https://unity3d.com/

[20] 3D Slicer. Dostupné z: https://www.slicer.org/

Appendix A

Abbreviations

APK Android application package

AR augment reality

CT computed tomography

CPU central processing unit

GPU graphics processing unit

GUI Graphical user interface

HMD head mount display

MRI magnetic resonance imaging

HTTP Hypertext Transfer Protocol

MIP Maximal intensity projection

OBB oriented bounding box

JSON JavaScript Object Notation

XML Extensible markup language

VR virtual reality

Appendix B

Content of CD

- `readme.txt` ..... Describtion of CD
- `exe` ..... The folder with UnityApplication APK build
- `src`
  - `UnityApplication` ..... UnityApplication (Unity3D project)
  - `UnityConnector` ..... UnityConnector (3D Slicer extension)
  - `thesis` ..... Source form of textual part in LaTeX
- `text` ..... Textual part of the thesis
  - `thesis.pdf` ..... PDF format