Slovak University of Technology in Bratislava
Faculty of Informatics and Information Technologies

Tím 16

Interaction and collaboration in virtual reality

Study program: Software engineering, Information systems

Field of Study: 9.2.5 Software engineering, 9.2.6 Information systems

Place: Institute of applied informatics, FIIT STU

Supervisor: Ing. Peter Drahoš, PhD.

Members: Bc. Lenka Kutlíková, Bc. Erik Bujna, Bc. Mário Csaplár,
Bc. Michal Dobai, Bc. Lukáš Doubravský, Bc. Martin Petráš, Bc. Ondrej Vlček

November, 2015

Contents

List of Figures

1 Introduction

Fully immersive environment has been a subject of on-going research for several years. The progress in technology goes side by side with the progress in the ways the technology is used. Interaction in the current computer era is defined by mechanic movements of mouse, typing on keyboard or physical contact with touch-based devices.

For virtual reality, however, none of these ways of interaction suffice. Using stereoscopic display, the user is immersed into a simulated 3D environment and is unable to use any physical device to communicate with it. In certain scenarios, there is no need for user input as the application might only serve demonstration purposes.

In real world scenarios, though, applications are required to be fully interactive. The question that comes into play is plain and simple. How can a user communicate with an application without using any physical contact? Finding an answer for this question is also the main goal of this project.

This document is a technical documentation for the course Team project at Faculty of informatics and information technologies. The topic Interaction and collaboration in virtual reality is a continuation of previous work supervised by Ing. Peter Drahoš, PhD. and Ing. Peter Kapec, PhD.

The first part of the document describes goals for the fall semester. Since the research has already been taking place for several years, the goals are mainly analytical. Before adding new functionality to the existing project, it is required to perform a deep analysis not only of the project itself, but also of the technologies used in past and the technologies that are planned to be used, namely Oculus Rift and Kinect.

Following parts are dedicated to the analysis of technologies and project modules. At first, the modules are depicted in the way they were available at the beginning of the project. Based on these information, changes are proposed and described.

2 Vocabulary and abbreviations

- *3DSoftviz*. Name of the existing solution that this project would contribute to.
- *API*. Application programming interface.
- *CPU*. Central processing unit.
- *DirectX*. Collection of high-level graphic processing API.
- *GPU*. Graphic processing unit.
- *HDMI*. High-definition multimedia interface.
- *HMD*. Head-mounted display.
- *Virtual reality*. A simulated model of physical world.
- *Immersive environment*. Simulated environment providing the user the illusion of being physically present in it.
- *Stereoscopic display*. Display consisting of two 2D displays creating the illusion of 3D view.
- *SDK*. Software development kit.

3 Goals for fall semester

Fall semester will be dedicated to the analysis of project domain, analysis of technologies and solving all the technical difficulties that would be encountered with the previous version of project.

Since the project called 3DSoftviz has been in development for several years, there have been many contributors, each with different purpose, requirements and needs. Currently, the project is dependent on many external libraries and modules, and even more, it is targeted for a very specific build platform.

The first goal is to generalize the target platform so that it would be possible to build the project on every major operating system. To perform the analysis effectively, it is required for as many team members as possible to get 3DSoftviz to work. Since retargeting such a complex project is a difficult task, it is expected this goal would take several weeks.

Meanwhile, the analysis of available hardware should be performed. Specifically for this project, Oculus Rift and Microsoft Kinect have been obtained. Their ability to help user interact in a fully immersive environment has to be considered from both hardware and software point of view.

Since there are two versions of Microsoft Kinect directly available for use, their capabilities have to be evaluated and compared. Both versions come with a software development kit and standard libraries that provide motion-capturing functionality.

Oculus Rift is a head-mounted display currently in development. Apart from the displaying feature, it provides spatialized audio and positional tracking. This is the main device that is considered for creating the experience of immersive virtual reality.

The current version of 3DSoftviz uses 2D graphic user interface created by means of Qt, which is not suitable for the project's needs. All of the Qt GUI elements have to be removed and the 3D view pane has to be displayed in full-screen mode using Oculus Rift. This as a whole forms one of the key objectives for the fall semester.

The background of this goal is simple. The user has to be able to interact with the 3D view pane having the Oculus Rift mounted on head. At this point, it is not possible to use any external hardware device to move cursor or click buttons. Every interaction task has to be performed within the projected virtual reality.

The user also has to be able to see a visualized model of selected parts of his own body so that he can keep track of his own movements. It would be impossible to interact with the projected scene if the current position of the user was not visualized. This is the main purpose that Microsoft Kinect would be used for. Thus, analysis of Kinect's hardware capabilities and the functionality of bundled software development kit is the second main goal for the semester.

Both functionalities should be at first implemented independently from 3DSoftviz so that they could be quickly evaluated and reworked in case they prove insufficient or incorrect. Together with retargeting of 3DSoftviz, evaluating and proposing new methods of interaction and analysis of existing 3DSoftviz modules, these are the tasks that would form the fall semester.

The product output of the semester should be an application that combines both aforementioned prototypes. Inside a split view designed for Oculus' displays, there should be an object mesh created by Kinect in real time.

4 Analysis of system modules

4.1 3DSoftviz

Project 3DSoftviz consists of following major components, as mentioned in documentations from previous development teams. The main components of 3DSoftViz are:

- *Core*. Consist of the most basic and relevant methods and it is responsible for initialization.
- *Layout*. Responsible for node placement within 3D environment.
- *Data*. Used for structural specification of the graph and defines each type of graph element.
- *Model*. Manages communications with database and maps objects from the graph to relational database and vice versa.
- *Network*. Used when collaborative work over the Internet on the graph is needed.
- *Importer*. Parses data from known file types into 3DSoftviz.
- *Kinect*. Comunicates with physical Kinect device, retrieves data flow and represents retrieved data.
- *Math*. Calculates advanced camera movements.
- *Viewer*. Provides basic camera movement.
- *QOSG*. Generates main graphical user interface with additional windows.
- *Noise*. Generates responsive background for easier orientation.

The current architecture of 3DSoftviz is displayed on the component diagram 1. This is the state after contributions of 2014 team team supervised by Ing. Peter Kapec, PhD. The physical data model in the picture 2 ensures storage of a complete graph structure.

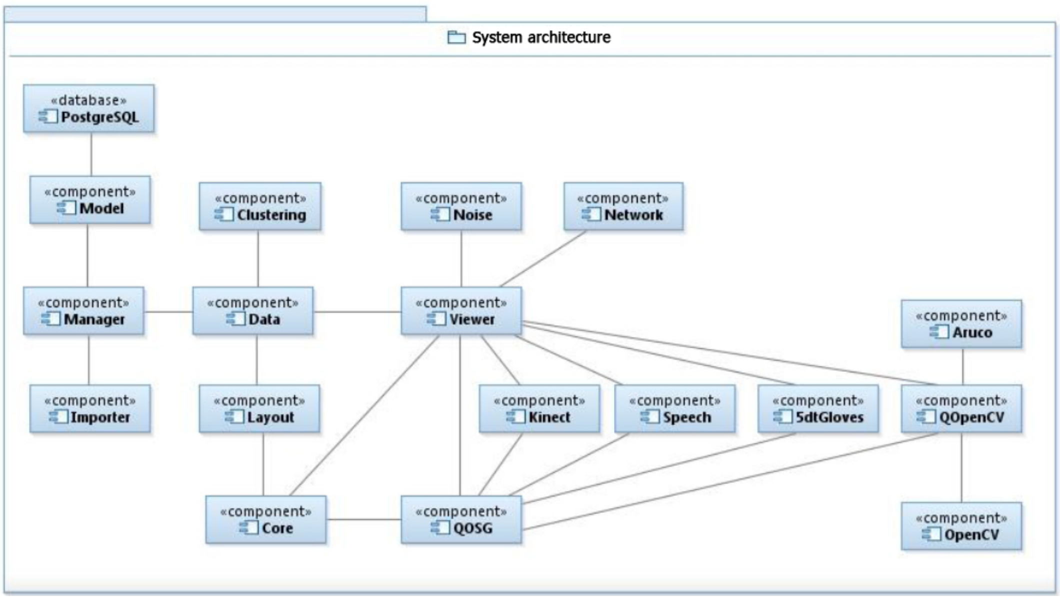

Figure 1: *2014 architecture of 3DSoftviz.*

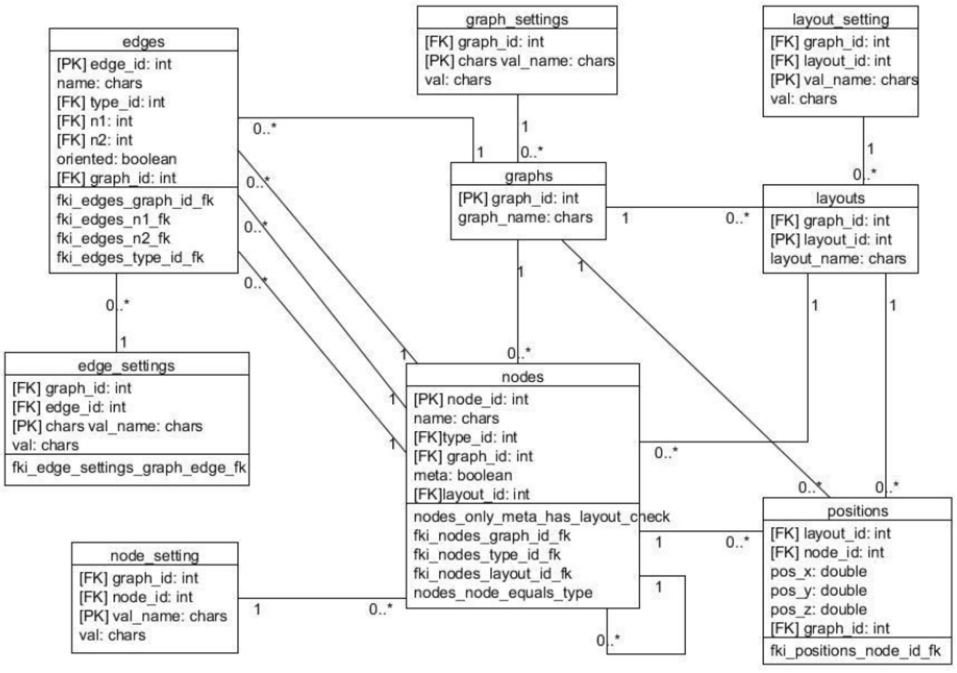

Figure 2: *Data model ensuring storgate of a complete graph structure.*

The picture 3 shows the current graphical user interface of 3DSoftviz created by the means of Qt. There are many 2D visual control elements such as buttons, toolboxes, dialogs or popup menus. None of them are going to be used in this version of project.

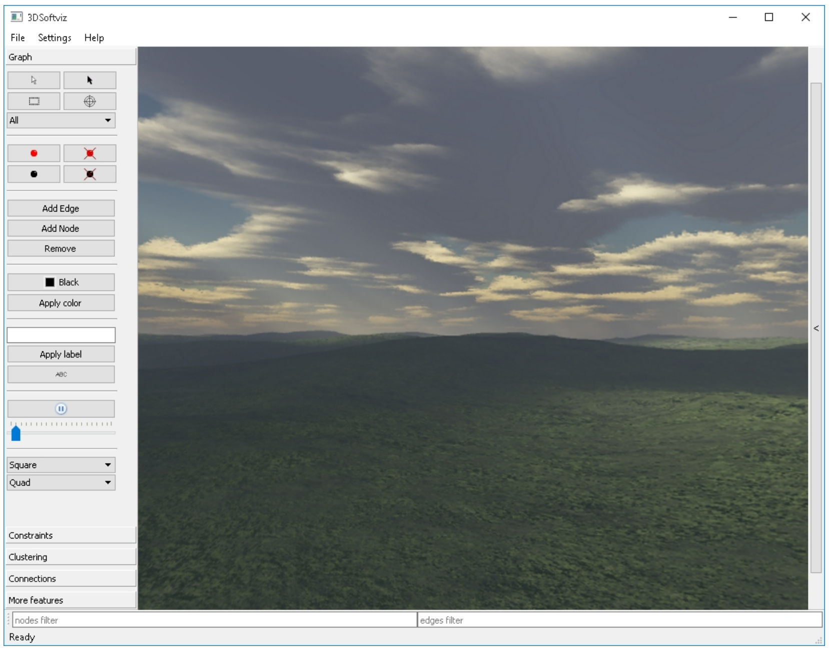

Figure 3: *Main window of 3DSoftviz.*

4.2 Oculus Rift

One of the main tasks of this project is to integrate Oculus Rift virtual reality device within 3DSoftviz. Oculus Rift is a virtual reality head-mounted display currently under development. It is also capable of head tracking and position tracking. This section provides the reader with the results of initial analysis and overview of Oculus' capabilities.

Recommended computer specifications

- Processor: Intel i5-4590 equivalent or greater
- GPU: NVIDIA GTX 970 / AMD 290 equivalent or greater

- 8GB RAM and more
- HDMI 1.3 compatible video output
- 2x USB 3.0 port
- Windows 7 SP1 or newer, DirectX platform update

It is also recommended to install latest GPU drivers.

Setup

The hardware setup is described at [1] and driver setup at [2].To start using oculus, a guide [3] is available with recommended first steps. Guide to setup Oculus Rift SDK can be found at [4] where also example applications can be found.

Development

A reference to Oculus PC SDK can be found at [5]. The SDK contains four C header files:

- *OVR_CAPI_0_8_0.h*. Rrendering and head tracking API.
- *OVR_CAPI_D3D.h*. Direct3D specific interface.
- *OVR_CAPI_GL.h*. OpenGL specific interface.
- *OVR_ErrorCode.h*. Error code declarations.

Application structure

Application is divided to 3 parts:

[1] https://developer.oculus.com/documentation/pcsdk/latest/concepts/dg-hardware-setup/
[2] https://developer.oculus.com/documentation/pcsdk/latest/concepts/dg-software-setup/
[3] https://developer.oculus.com/documentation/pcsdk/latest/concepts/ug-tray-start/
[4] https://developer.oculus.com/documentation/pcsdk/latest/concepts/dg-sdk-setup/
[5] https://developer.oculus.com/doc/0.8.0.0-libovr/index.html

- *Initialization*. In this part, initialization of Oculus Rift device is taking place. That means resource allocation for HMD (head mounted display) and head tracking sensors of the device. This is also the time point where initialization of other application components is being done. Initializations of sensors and rendering initialization is described in Oculus documentation. To initialize Oculus API, it is required to invoke *ovr_Initialize* function.
- *Game loop*. Application logic, rendering and other user interaction is done in the loop, until application is instructed to stop.
- *Shutdown*. This part is responsible for correct resource deallocation. To shut down the API, it is required to invoke *ovr_ShutDown*.

Sensors and head tracking

The Oculus Rift device contains a number of microelectrical mechanical sensors (MEMS) including a gyroscope, accelerometer and magnetometer. Information from each of these sensors is combined in process called sensor fusion process. It is used to determine user's head position in real-world.

By default, all tracking features of connected HDM are enabled, but they may be toggled using the API. The API reports data about user's head position in right-handed coordinate system (X is positive to the right, Y is positive in the up direction and Z is positive backwards, from user point of view). The library provides C++ OVR Math helper classes to simply work with given data.

Rendering

The Oculus Rift requires split-screen stereo with distortion correction for each eye to cancel lens-related distortion. Oculus handles distortion automatically. Application's part here is to render stereo view on a world based on user's head position. That means, application will render a scene with 2 cameras, one for left eye and one for right eye. Z-axes of cameras are parallel, so it is like one conventional camera in the middle, translated to the left and to the right.

The SDK is only for displaying textures on HDM, it doesn't do actual rendering. Rendering can be done in any way with OpenGL or Direct3D. Also engines that can render to OGL or D3D textures can be used. After rendering to the texture is done, Oculus API function is invoked to display this texture, or multiple textures.

Debugging

Oculus provides debugging tool which can be found at [6].

4.3 Microsoft Kinect

For the interaction part of the project, Kinect sensor for Xbox One, also known as Kinect v2, is considered. The first generation of Kinect sensor is also at disposal, but for this project, the second generation shall be used. The official Microsoft Kinect for Windows SDK 2.0 is used for development. Open source and multi-platform libraries (*libfreenect2*) were considered for further development.

Recommended computer specifications

- CPU: 64-bit physical dual-core 3.1 GHz or faster
- GPU: DX11 capable graphics adapter
- USB 3.0 port
- Windows 8 or newer

Kinect sensor for Xbox One specifications

- *Video camera.* The video camera has a 1080p resolution, which can be used for recording or chat. It has ability to capture 6 skeletons at once, read player's heart rate and track gestures performed with Xbox One Controller. The previous generation of sensor has VGA resolution of 640x480 pixels and uses 8-bit color depth.
- *Infrared motion controller.* This controller allows to see in the dark and it is capable to produce a lighting-independent view, so infrared image and color can be used at the same time. Over its predecessor, it has greater accuracy,

[6]https://developer.oculus.com/documentation/pcsdk/latest/concepts/dg-debug-tool/7

and processes the data at 2 Gbit/s. The prior generation creates also infrared image at VGA resolution.

- *Time-of-flight camera for depth analysis*. The depth sensor is a camera with resolution of 512x424 pixels, capable to capture depth from 0.5 to 4.5 meters far from the sensor. As opposed to prior generation, it provides higher depth fidelity and significantly improved noise floor, large angular field of view (70° horizonal and 60° vertical, as opposed to 57° and 43°), 3D visualization and ability to diferentiate smaller objects has been upgraded. The previous sensor used in Kinect has had VGA resolution and provided 11-bit depth.

Usage consideration

Since it is not possible to provide interaction solely using Oculus Rift, Kinect sensor is going to be used to interact with the graphic scene. The depth sensor allows capturing of hands, which can be further transformed into a 3D model. The hands model can be transferred into the scene and visualized from the user's perspective. With hands rendered in the environment, custom hand gestures can be created to invoke custom actions.

A prototype was created to demonstrate the capabilities of skeleton recognision. This can be further reworked and integrated into the project as a part of the detecting system.

Time-of-flight camera The depth sensor is a camera with resolution of 512x424 pixels, capable to capture depth from 0.5 to 4.5 meters from the sensor. As opposed to prior generation, it provides higher depth fidelity and significantly improved noise floor, large angular field of view (70° horizonal and 60° vertical, as opposed to 57° and 43°), 3D visualization and ability to diferentiate smaller objects has been upgraded. The previous sensor used in Kinect has had VGA resolution and provided 11-bit depth.

Infrared controller This controller allows to see in the dark and it is capable to produce a lighting-independent view, so infrared image and color can be used at the same time. Over its predecessor, it has greater accuracy, and processes the data at 2 Gbit/s. The prior generation creates also infrared image at VGA resolution. Video camera The video camera has a 1080p resolution, which can be used for recording

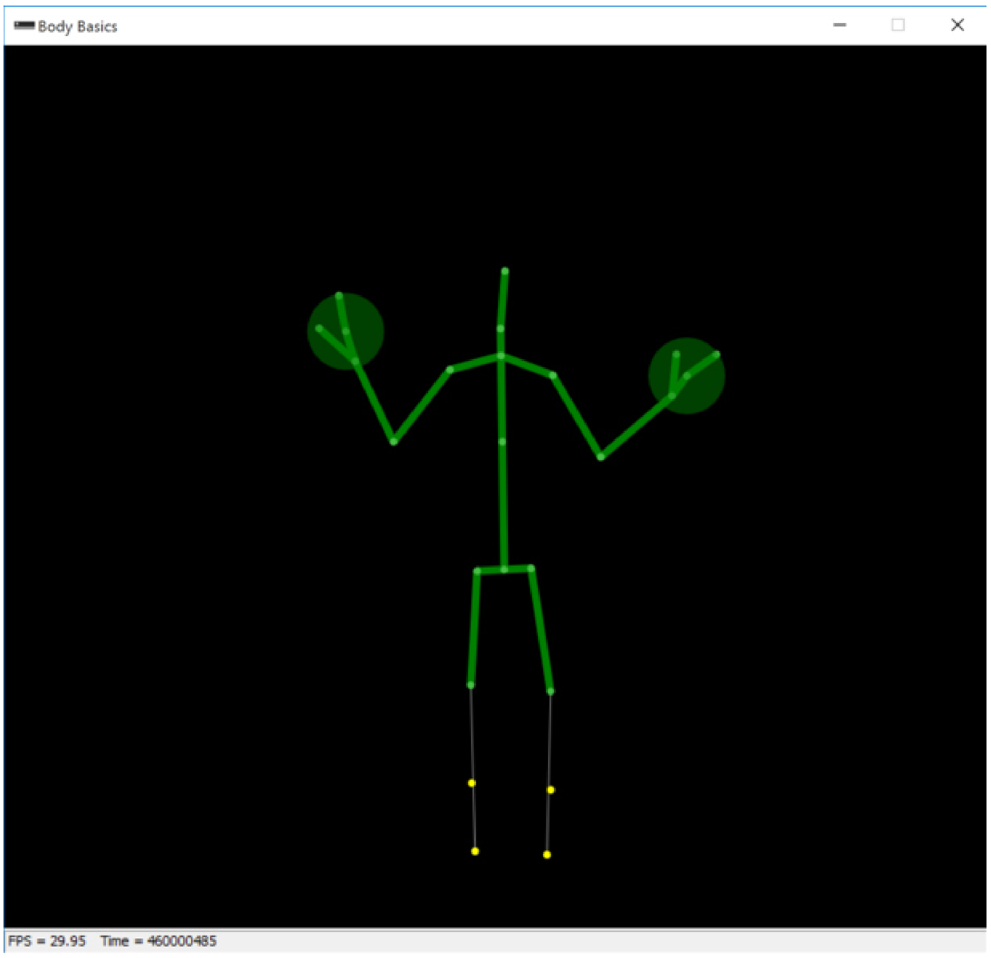

Figure 4: *Skeleton detection.*

or chat. It has ability to capture 6 skeletons at once, read player's heart rate and track gestures performed with Xbox One Controller. The previous generation of sensor has VGA resolution of 640x480 pixels and uses 8-bit color depth.

Use of Kinect in our project Since with Oculus Rift, we cannot provide interaction, we are going to use Kinect sensor to interact with the graphic scene. The depth sensor allows us to capture a 3D representation of hands, which will transformed into 3D model of hands. The hands model will be transferred into the scene and visualized from the person's perspective. In our project, we can then create custom hand gestures, which will invoke custom actions.

5 Design and prototyping

This chapter describes evolutionary prototypes created to test selected functions of the hardware devices.

5.1 VXOculusViewer

This prototype serves a means for straightforward communication with Oculus Rift. Created in OpenSceneGraph, it renders two independent scenes into a single texture which simulates the look target for human eyes. In total, there are 4 examples, out of which 3 demonstrate different techniques and approaches and the 4th is a playground for debugging purposes.

To run this example, application needs to be run with parameter: *OstText.exe n* where n is example number. In examples 1, 2 and 4 it is necessary to move rendered objects to be visible by mouse while holding left mouse button.

Example 1 - texture mapping

This example demonstrates simple texture mapping. It displays a scene which contains one textured quad.

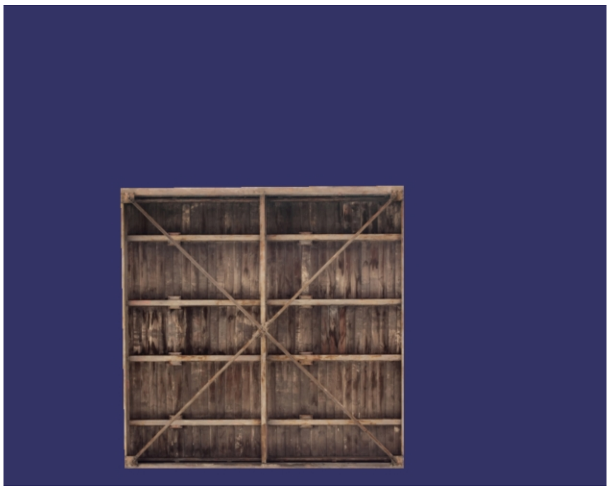

Figure 5: *Example 1.*

Example 2 – render to texture

Example shows how to render to texture with OpenSceneGraph. It contains a scene with textured quad. The texture is result of rendering another scene, in this case a colored quad with green background.

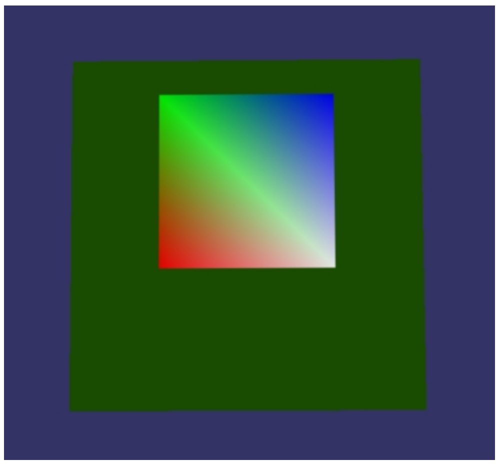

Figure 6: *Example 2.*

Example 3 – render to two textures

In this example, a scene is rendered to two textures, each time with different camera – for left and right eye. Then, these textures are displayed next to each other. The scene is rendered with different clear color, blue for left eye and red for right eye.

This example represents main functionality of the prototype. It is implemented in class VXOculusViewer. This class uses OpenSceneGraph viewer internally to display result described above. It contains scene graph described by diagram.

Nodes eyeLeft and eyeRight are camera nodes. Their child node, scene, is root node of the scene graph which is rendered to textures (textureLeft and textureRight). These are then used as textures on squares in displayed scene - quadLeft and quadRight. Because this solution uses scene graph, it is easy to customize.

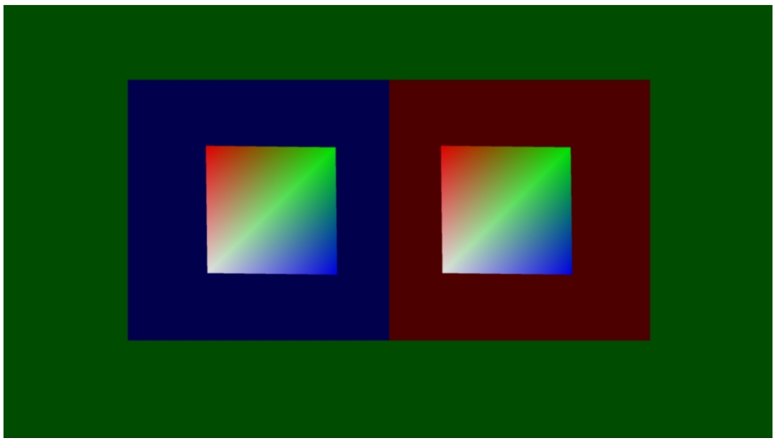

Figure 7: *Example 3.*

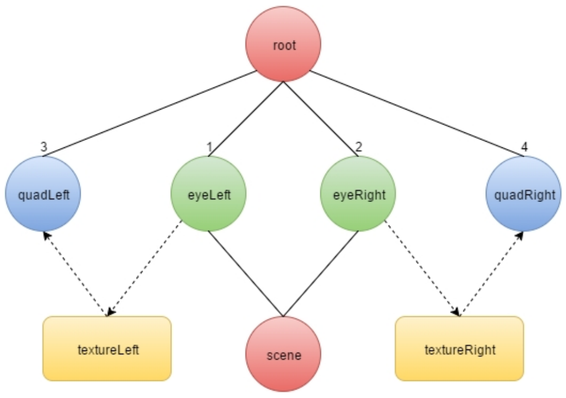

Figure 8: *Diagram.*

Installation

Prototype was built only on Windows. Code itself should be multiplatform, but OpenSceneGraph library has to be compiled on specific platform. Also cmake must be modified.