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

May, 2016

# 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.
- *P2P*. Peer-to-peer.
- *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 Goals for spring semester

Spring semester will be dedicated to the creation of a final application: combining libraries for Kinect and Oculus handling, network communication library, and figuring out what scenario to implement into the final application to test the limits of Kinect's capabilities of tracking human body.

The first library that we want to create is a library for displaying the scene into Oculus Rift: necessary features of the rendering is doubled image of the scene – one for each eye from its perspective – and each image needs to be distorted.

The second library is the Kinect library which will serve as a wrapper for Kinect API and provide necessary information from Kinect, such as body joints, point cloud and depth.

The last library should provide communication between two instances of the same final application and should transfer selected data which will be rendered on either side. Before the implementation, it should be analyzed and decided which library should be used for communication.

As all these libraries will be finished, we can proceed to creating logic for our final application. The app should be capable to test the restrictions of either interaction and communication using Kinect and Oculus. The final product will then have a set of objects in scene which can be moved using gestures provided by Kinect, and mesh representation of another person using second Kinect.

# 5 Analysis of system modules

## 5.1 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.

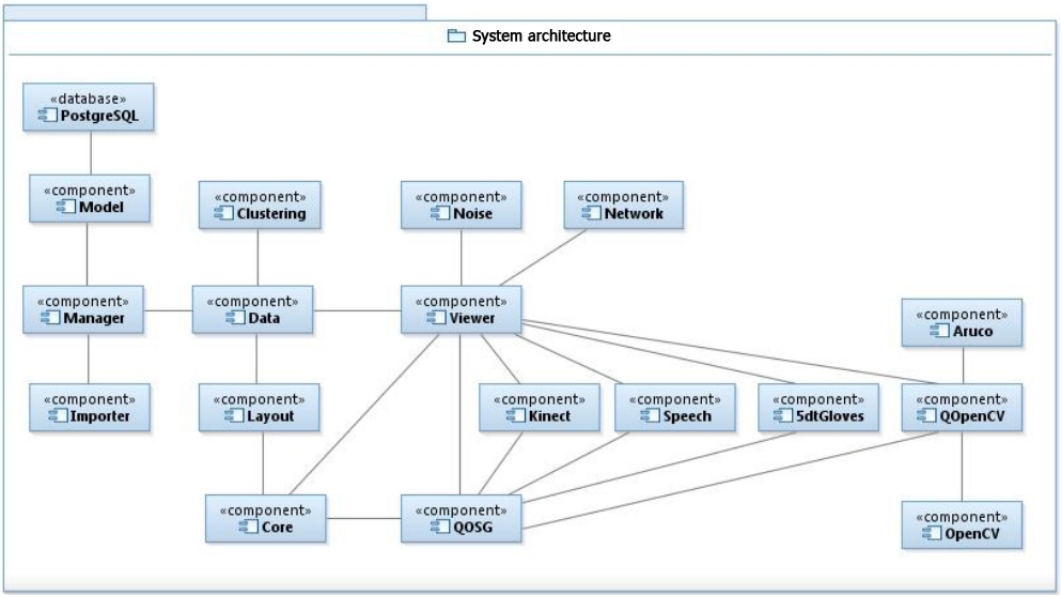


Figure 1: *2014 architecture of 3DSoftviz.*

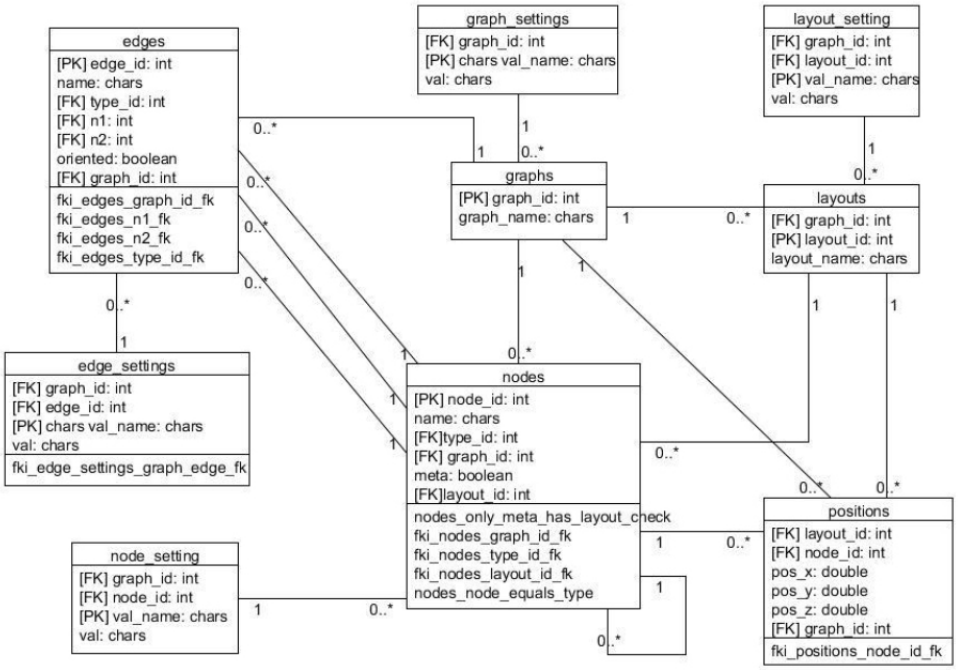


Figure 2: *Data model ensuring storgate of a complete graph structure.*

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.

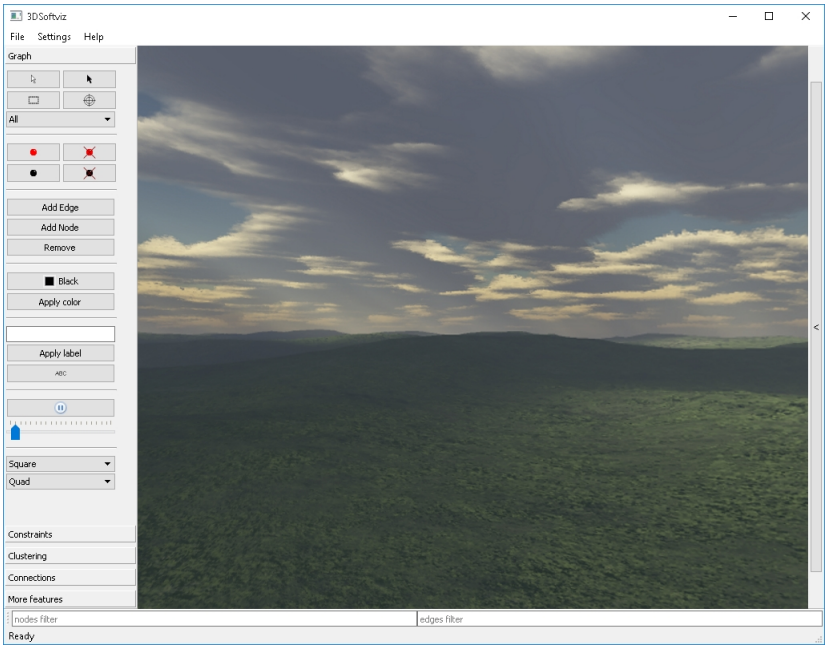


Figure 3: *Main window of 3DSoftviz.*

## 5.2 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].

## 5.3 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.

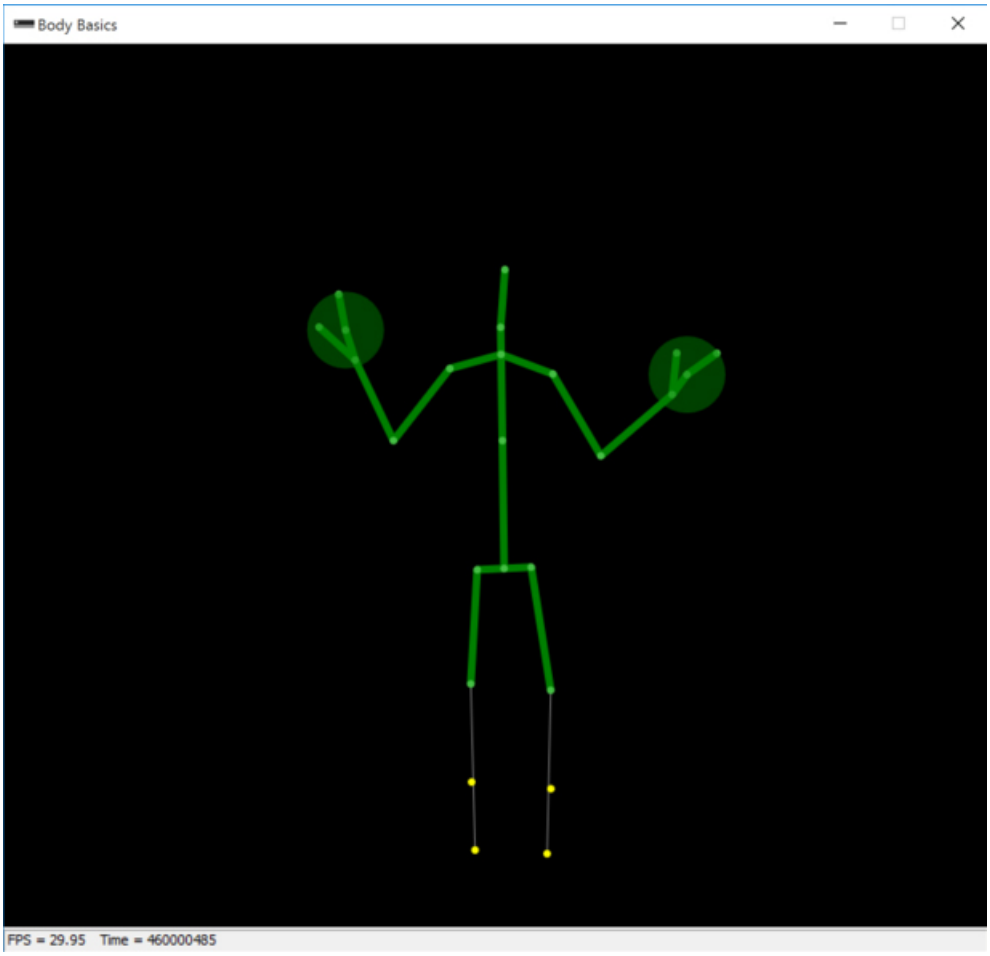


Figure 4: *Skeleton detection.*

### 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 a depth image, which can be used to recreate polygons and subsequently a whole a 3D model. As it is a collaborative application, there are two options for the usage. Either there could be a second user reconstructed within virtual reality or hands visualized from the user's perspective. With hands rendered in the environment, custom hand gestures can be created to invoke custom actions.

Another consideration is to use Kinect to capture the position of user's joints, such as hands, wrists, elbows, collar bone, head and legs to get a full picture of where the user is and what he is doing. Then, a pre-rendered custom 3D object could be displayed in the position of the joints to help user see himself without the requirement of real time user's 3D reconstruction.

### Kinect SDK for Windows

All of the functionality mentioned above is in a certain way abstracted within Kinect SDK. After installation of Microsoft Kinect SDK, the target computer is capable of processing Kinect streams. There are several versions of the package depending on the target programming language.

## 5.4 Networking

Navigation, comunication and collaboration without words in virtual reality is much different from the real-life and therefore much more challenging. Users do not have the freedom of using the traditional techniques like typing on the keyboard or using the mouse. That is why evaluating various alternatives of collaboration is important. Networking is one of the instruments that come in handy to bring distant people together.

Analysis deals with basic analyzing of known libraries and approaches when dealing with issues of networking. First of all the network topology was discussed. The focus was mainly given on peer-to-peer topology where every participant sends own data over the nework, and whoever is located on the network or the multicast

address has the newest available data. This type of topology is difficult to debug and has several discovery problems, as well as problems with data inconsistencies and synchronization

As an alternative, the well-know and widely used client-server type of architecture was discussed. There would be one globally known server to which client would connect. The server would keep the register of connected clients. When a client changes his state, he would share the changes with the server, which would in turn delegate the changes to the other connected clients. Several discovery options on local network were discussed, such as periodic client broadcast or server side broadcasts.

In terms of networking, there are two main types of layer 4 protocol out of which each one is suitable for different purposes. While TCP ensures the delivery of messages but requires more network overhead, UDP on the other side is faster but unreliable.

Out of available low-level implementations the following were evaluated:

- Pure sockets
- Winsock
- Winsock2

Analyzed frameworks count:

- Boost.Asio. Evaluated as too extensive for the purposes of this project.
- Protocols Buffers. Supports easy object and struct serialization but no networking.
- Qt. Also too large.
- Poco. Lacks quality, legible documentation but supports implementiation of TCP and UDP server.

# 6 Design

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

## 6.1 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.

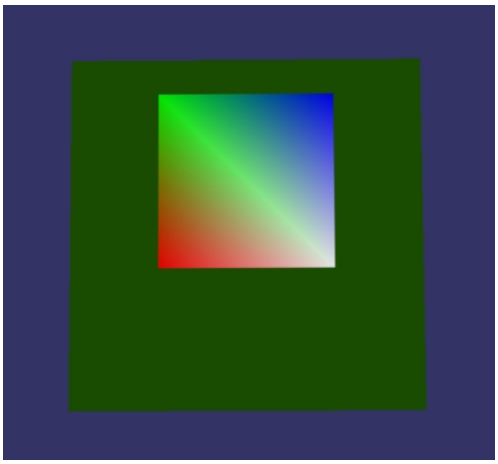

Figure 6: *Example 2.*

### 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.

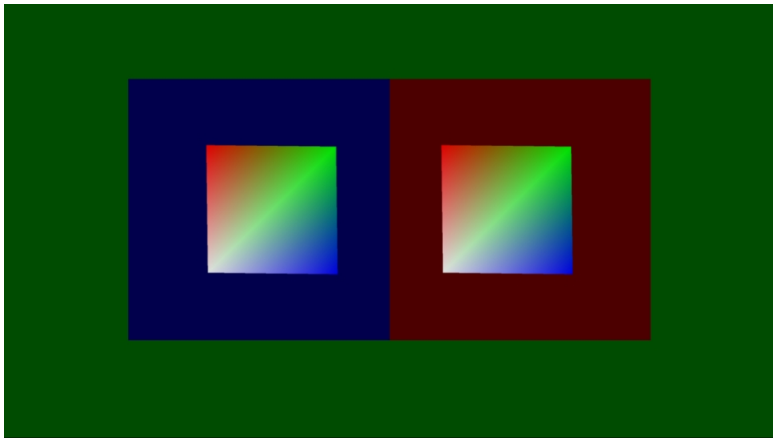

Figure 7: *Example 3.*

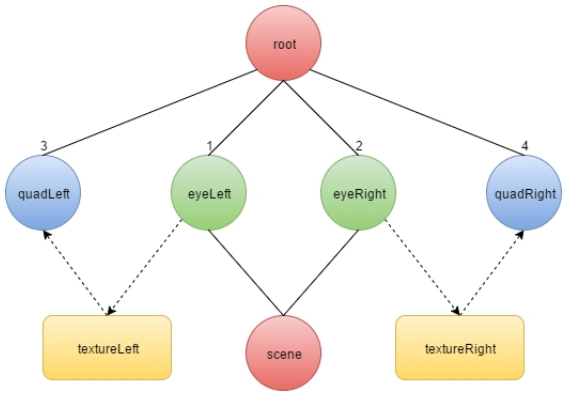


Figure 8: *Diagram.*

### 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.

## 6.2 Module for windows and OpenGL

To use OpenGL, an application must create an OpenGL context. This context enables OpenGL to manage system resources correctly. That is the main purpose of this module. It should provide methods to initialize and maintain an OpenGL context, window-related functionality and interfaces between an application and

the operating system, to handle user input from mouse and keyboard.

Because Oculus Rift device cannot be used on some computers (for example, it does not work with Intel's GPUs, but most members of the team has one, hence it would not be possible to debug an application properly), the application has to be able to display stereoscopic rendering without an Oculus device.

To make the module for Oculus a little bit simpler, it was decided to implement functionality mentioned above within this module. The module should provide two types of windows: one standard window, which displays an image from the default framebuffer (it is possible to copy a content displayed on an Oculus device to a framebuffer, so display what user sees to a window). The second type should display two images (in form of 2 textures) next to the each other.

To simplify and unite the work with shaders, the module should provide classes that wrap OpenGL shaders in a convenient way. That means loading shader code from file system, compiling and linking shaders and submitting uniforms.

## 6.3 Module for Oculus

This module should provide an interface between an Oculus rift device and the application. For the reason described above (Oculus Rift does not work on every computer), it should be able to work with and without an Oculus device with exactly same interface (without any change of client's code or need to recompile an application). This should be achieved by common interface for 2 different classes which share same base class. The module should also wrap any interaction with module for windows and OpenGL that is related with windows, but not with OpenGL (shaders, etc.).

## 6.4 Module for Kinect

Since the analysis consisted mainly of samples from Kinect SDK, it was determined they are based on Direct2D. Another drawback is that the application logic is very highly coupled with presentation logic and untying them would require a lot more effort than building a new module based on OpenGL. This new module would also better correspond to the requirements of this projects, which are mainly:

- The ability to capture image streams. Kinect SDK provides a FullHD color image stream and a couple of low resolution streams from depth sensor and infrared sensor. It should be possible to display all of the available image streams in their native resolution within a single window.
- The ability to filter the image streams. Based on the analysis of Kinect SDK, it was figured that the streams require post processing that can be highly parametrized. Especially the infrared streams shows as monochromatic snow when not properly filtered using color levels selection.
- The ability to track people. Kinect is capable of tracking at most 6 people simultaneously in terms of capturing the position (in meters) of their selected joints. It should be possible to display the joints of a tracked person and get access to the underlying methods.
- The ability to reconstruct the scene. Thanks to Kinect Fusion, it should be possible to combine depth and image streams to create a full 3D scene reconstruction. An output should be available in a simple form that is either a point cloud image or a vertex mesh.
- Separation of concerns. The logic of accessing and transforming the image streams into more sophisticated data structures should be strictly separated from the presentation logic.
- Modularity. Since the whole application is designed to be modular, business logic and presentation logic should be both contained within a separate static library that is simple to reference from any project.
- Resource management. The application logic should be built the way it would spare system resources. If a user only asks for a depth image, the application should not force the Kinect device to create all the other aforementioned data.

The designed general workflow is depicted in the figure 9. Simplified class diagram showing the component structure is in figure 10.

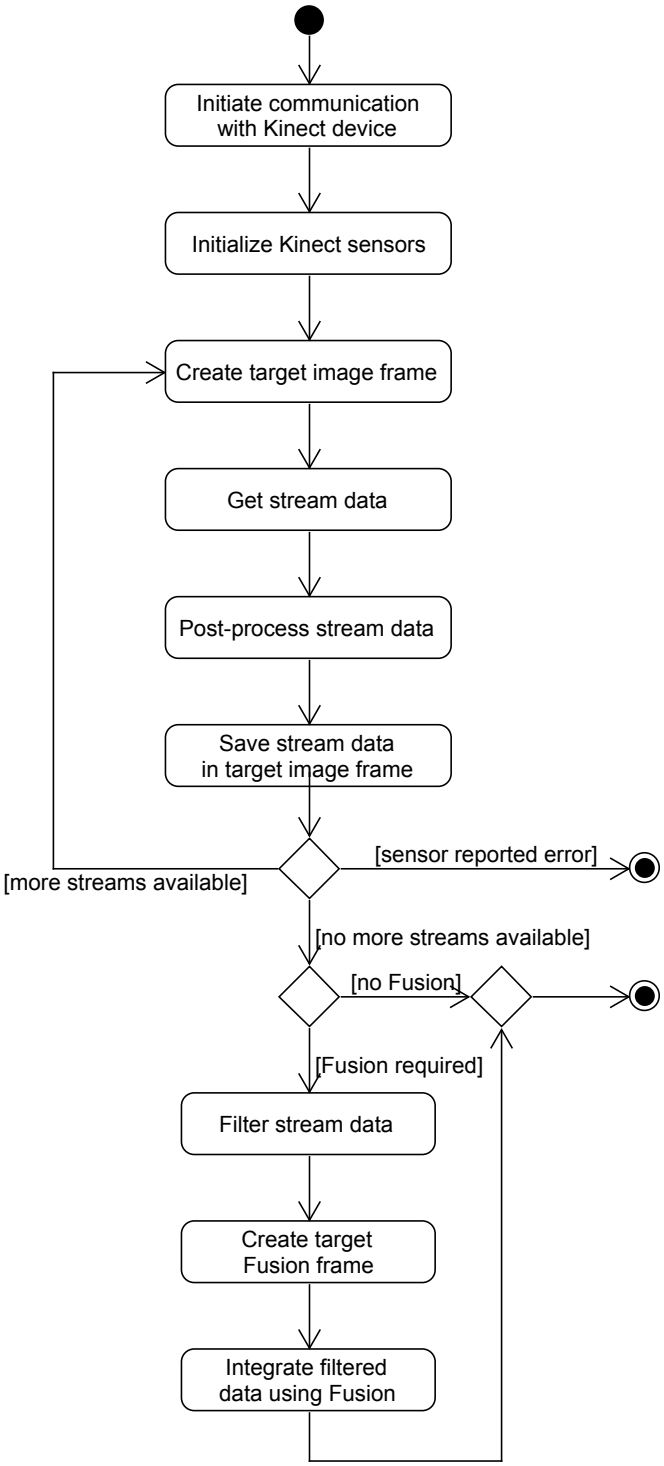


Figure 9: *General workflow of processing Kinect data.*

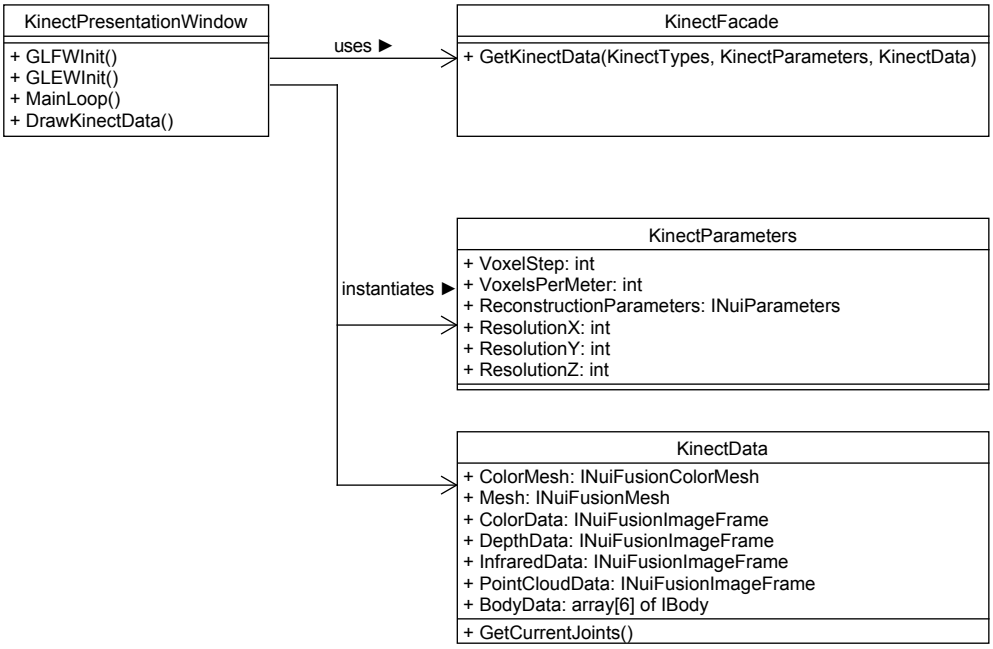


Figure 10: *Simplified class diagram of Kinect module.*

## 6.5 Module for networking

Design section describes the answers that had to answered, like usage and requirements of the communication module. For this we needed to set some basic data requirements that would determine what specifically would be send over the network.

The following data structures were identified:

- Scene related. Objects, hierarchical scene objects.
- User related data. User's position within the scene, the focal point of user's view.
- Kinect related data. Depth map, color image or mesh.
- Interaction objects. Object which the user manipulates or the objects that are generated during runtime.

All of these factors determined the requirement to set a reasonable limitation. Sending Kinect data over the network 30 times per second would require more than just ordinary network bandwidth.

For network communication, sockets were chosen, specifically the library running on .NET framework 2.0 architecture called *System::Net::Sockets*. It implements and abstracts packet fragmentation and defragmentation from the user.

The next thing to consider while choosing an appropriate framework was the coverage of required function. In case it was required to implement additional functionality, it would be necessary to analyze the existing code base which would cost time and risk the possibility of breaking the existing library or framework in topic.

Next defined requirement was that the library should be programmatically easy to use, and the would manage its own resources. All necessary initialization would be hidden from the user as it would be done during the loading and constructing of the library. The only interaction from the library user would consist purely from calling responsible method for sending or receiving.

Final conclusion was that the receive method should always return valid data even if they are not current. The send functionality should be as less blocking as possible, in the best case implemented asynchronous call. Next important decision was to create a separate operation thread for the listen method and that the same application instance would run on all of the clients.

Last issue concerned the data format that would be sent over the network. Because of speed concerns, serialized byte array proved better in comparison to human readable formats such as JSON or XML.

## 6.6 Module for application

The application module will be the core of the developed functionality that combines all the aforementioned modules as shown in the package diagram *f:package*. There should be two display modes available, one for an user with Oculus device connected and one for an user with a classic computer screen.

The purpose of the application is to verify the correctness of individual modules by putting them into a real working scenario. The scenario itself should be created

so that it is possible to evaluate the Oculus Rift and Microsoft Kinect devices in terms of accuracy, speed and usage convenience.

For this purpose, two main game modes were created. The cube game mode tests both interactive and collaborative aspects of the virtual reality prototype. User with a head mounted display sees a triplet of cubes with varying size and his task is to lift the correct cube. The only way to determine which cube is marked correct is to look at his partner scanned by a Kinect device whose 3D reconstruction is placed within the scene.

The scanned partner sees colourful cubes on his computer screen and his task is to aim at the green cube. If the user with head mounted display manages to catch the correct cube aided by a visualization of his own hands, he gets positive score. The purpose of this game is also to be able to change the size of displayed cubes to determine the limit where it starts to be very difficult to catch any cubes.

The second game Hanoi Towers consists of triplet of poles on which cubes are placed. The goal of this minigame is to reorganize the cubes in descending order of their sizes to another pole using the same gripping technique as with the cubes minigame.

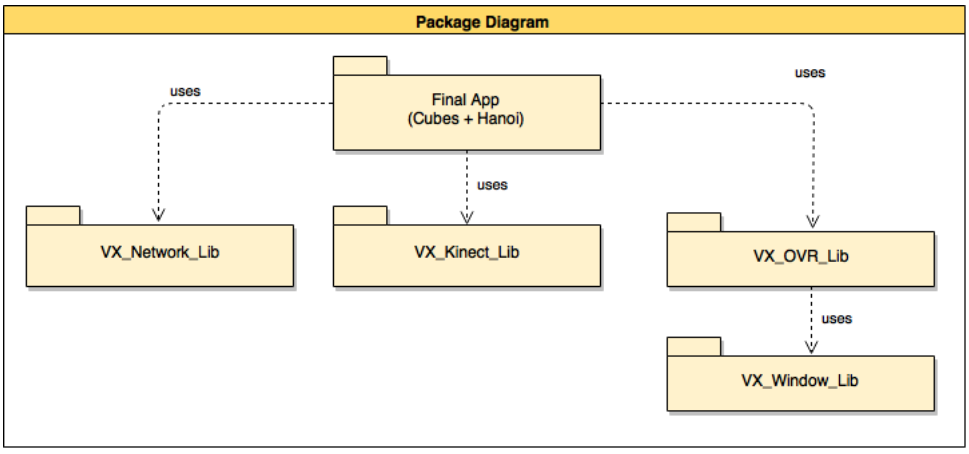


Figure 11: *Package diagram of final application.*

# 7 Implementation

## 7.1 Module for window and OpenGL

This part contains description of this module and should not be considered as a reference. It consists of 2 namespaces – vx_window_namespace_ (with alias vxWnd) for things related with windows and vx_opengl_namespace_ (alias vxOpenGL) for things that are related with OpenGL and their purpose is to support development. All functionality is enabled by including To handle windows and OpenGL context, the GLFW library is used. To load and wrap OpenGL, the module uses GLEW library. Both libraries are multiplatform, and the code of this module should be also portable to any platform which those libraries support (only Windows is tested). The namespace vx_window_namespace_ consists of following classes:

- *BaseWindow*. Pure virtual class, defines an interface and implements some of the basic getters and setters. Also it defines interfaces for callbacks, that are invoked in case of a user input (keyboard and mouse only)
- *GLEWWrapper*. A singleton class, to initialize the GLEW library. The GLEW library loads and stores pointers to OpenGL functions. It can't be initialized without valid OpenGL context. It is only meant for internal use of the module.
- *GLFWWindow*. Inherits from BaseWindow. It implements functions of the BaseWindow class and wraps a window provided by the GLFW library into an object. An OpenGL context is created when a window is created.
- *GLFWStereoWindow*. Inherits from GLFWWindow. It has different interface to update the content of a window (to draw next frame) – it has two input parameters, which are textures, that will be rendered abreast in the window.
- *GLFWWrapper*. Wraps initialization and deallocation of the GLFW library. The only constructor is private and it is possible to create only one instance. It is meant only to be used for internal purposes of the module

- *VX_Window_RuntimeError*. Inherits from std::runtime_error; it is an exception type, its only purpose is to sever other runtime exceptions Figure **??** – hierarchy of window classes The second namespace, vx_opengl_namespace_ contains the following classes:

- *OpenGLShader*. Class to support development. It implements functions to create, compile and link shaders and functions to submit unifroms to a shader program. It holds an id of the opengl shader.

- *OpenGLRuntimeError*. Iinherits from std::runtime_error; serve to sever other runtime exceptions

## 7.2 Module for Oculus

This module handles interaction between the OVRLib (library by Oculus) and an application. It also wraps some common operations provided by this library, so they can be used in more convenient way. The module can be imported by including file VX_OVR_Lib.h. Every class is contained in namespace vx_ovr_namespace_ (alias vxOvr). Oculus module uses the Oculus SDK library. Most recent versions do not support any platform, but Windows. For minor functionality (described later), AntTweakBar is used. It supports Windows, OSX and Linux, but because of the Oculus SDK, this module is probably compatible only with Windows. We have used it only on Windows. The module is composed of these classes:

- *OVRWrapper*. Singleton class; this class handles initialization and deallocation of the whole module. An application should call its method initialize. To obtain an instance of an OVRHmdHandleWithDevice class or OVRHmdHandleNoDevice, an application should call getOVRHMDDeviceHandle function. If an Oculus HMD device is detected, it returns an OVRHmdHandleWithDevice instance, otherwise it returns an OVRHmdHandleNoDevice. An application should never create instances of these classes.

- *OVRHmdHandle*. Pure virtual class, it defines interface for OVRHmdHandleNoDevice/WithDevice classes. Both of these classes also provides methods that returns projection matrix (implemented in this class) and view

matrix (implemented by inherited classes). To obtain correct view matrix, the method that provides this matrix must be invoked each frame.

- *OVRHmdHandleNoDevice*. Inherits from OVRHmdHandle. This class is returned by OVRWrapper, if an Oculus HMD is not detected. Internally it uses GLFWStereoWindow to display image (without HMD the image is not distorted). It also uses AntTweakBar library to create a quaternion bar to simulate head movement. It uses same parameters as Oculus DK2 (for example, screen resolution).

- *OVRHmdHandleWithDevice*. Inherits from OVRHmdHandle. It is returned by OVRWrapper when Oculus HMD is detected (must be turned on). To display image on a window, it copies distorted image from oculus directly to the window's default framebuffer. For this purpose, it only need GLFWWindow. That means, both classes use different interface to display image internally, but from an application point of view it is the same.

- *VX_OVR_RuntimeError*. Exception type, to sever from other runtime exceptions.

## 7.3 Microsoft Kinect

Despite the lack of quality documentation for the Kinect API, it was possible to accomplish all the requirements listed in the design section. A single entry point for the application module was created under the name *KinectFacade*. All the user has to do is instantiate the facade once and pass an instance of each *KinectParameters*, *KinectTypes* and *KinectData* classes.

Based on the selected Kinect types, the facade/proxy object calls private methods to receive all of the required data types and stores them inside a passed in *KinectData* which should be instantiated every time before being passed in.

An important implementation issue was empirical determining of steps required for successful Kinect Fusion scene reconstruction. The picture 12 shows the workflow of Kinect Fusion which unfortunatelly is not described in greater detail anywhere within the documentation. The core of Fusion reconstruction is

a depth map in floats which after further postprocessing can be integrated into a reconstruction frame in order to create volumetric mesh.

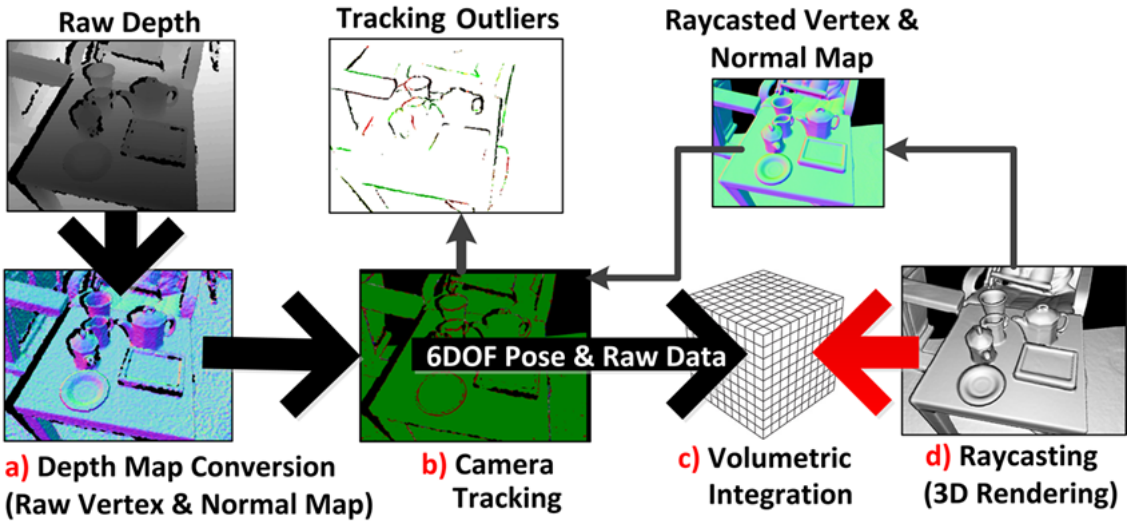


Figure 12: *Workflow of the Kinect Fusion.*

The reconstruction process takes into account several quality parameters in the order:

1. minimal depth (clamping)
2. maximal depth (clamping)
3. smoothing factor (noise reduction)
4. maximal voxel resolution for x,y,z dimensions (determines the amount of captured data in each dimension)
5. voxel:meter ratio of volumetric mesh (determines the amount of captured detail)
6. surface mesh granularity (can skip each n voxels)

For the visualization of generated mesh, shaded point cloud image can be used without any significant performance impact. All of these steps were implemented so that full parametrization of the reconstruction process is possible. To keep 30

frames per second, recommended resolutions are 256/256/256 with 128 ratio. The higher granularity, the less vertices in the final surface mesh but also higher the chance of having 'holes' in the mesh.

Unfortunatelly, it turned out impossible to map the coordinates of the final surface mesh to the coordinates of caputed body data. A custom transformation matrix would be required to create. It is, however, not possible to do so without full understanding of the internal transformation process which is not documented on the required level. The final surface mesh displayed in scene is shown in figure 13.

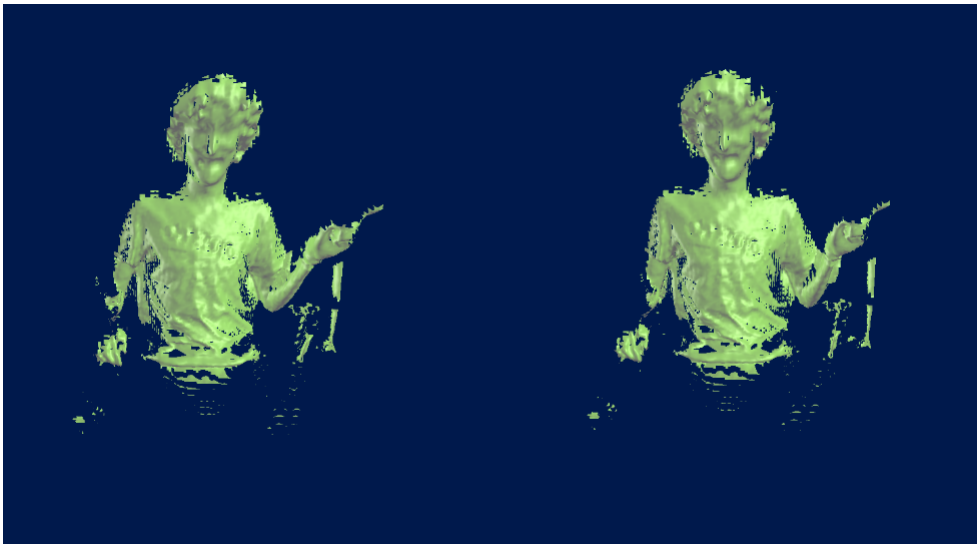

Figure 13: *Grayscale mesh with green ambient light.*

## 7.4 Networking

Following chapter describes the process of implementation which follows the incremental principles, starting with the smallest working code possible and subsequently expanding into larger pieces of functionality. The first attempts consisted of connection establishment for the loopback device.

Then it was advanced to establishing connection between two physical computers. Next step was to send one predefined byte over the network. Then interfaces were implemented and tested which allowed the user to send custom unsigned char and retrieve data from the library itself. At last, simple data structures were send in

the form of an array with custom values.

Configuration files were added for simple configuration option without the need for recompilation. Next, a specialized structure was created which is serializable by design without extra programmatical effort and the base of communication library was implemented.

Next bigger step was to import real data from the user. There were some problems with declaration of unmanaged types and structures in managed types. A unmanaged wrapper needed to be created to cross the gap between native unmanaged C++ and managed C++/CLR. After resolving this problem it was finally possible to send real data from Kinect.

Last step was to implement and test game logic for the collaboration and interaction part of the assignment. Method *IsNewDataAvailable()* was added for better hardware and software resource management.

At the the end, the normals were added, for better user experience when rendering mesh from the Kinect. During implementation a conclusion was reached that in the laboratory environment it is enough to implement one-to-one direct communication instead of a fully functional client-server architecture with multiple clients connected.

The following list defines the biggest problems which occurred during development phase of the communication library:

- During development and testing, several gigabytes were defined by mistake for the send and receive buffers. At that time, runtime memory usage of the application reached 5 to 6 gigabytes, now it is reduced to under 512 MB.
- There was a need for only one instance of the managed object *NetLib*, so Singleton pattern was implemented,
- There was a problem with passing the arguments from unmanaged code to managed, which in the result is done by hand. It can be achieved also by marshaling in managed code, but it is resource consuming, so it was not used.
- It turned out impossible to link a native C++ project with a managed static library, so a dynamic library was created instead.

- It was unable to send object of variable lengths, so static length of object was defined, with known structure for both sides of the communication.

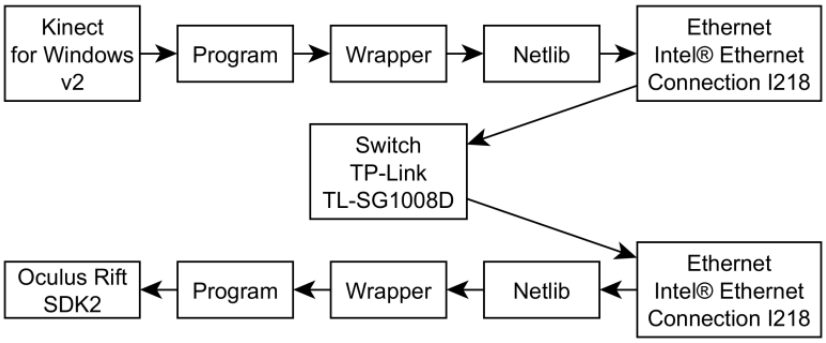


Figure 14: *Testing process.*

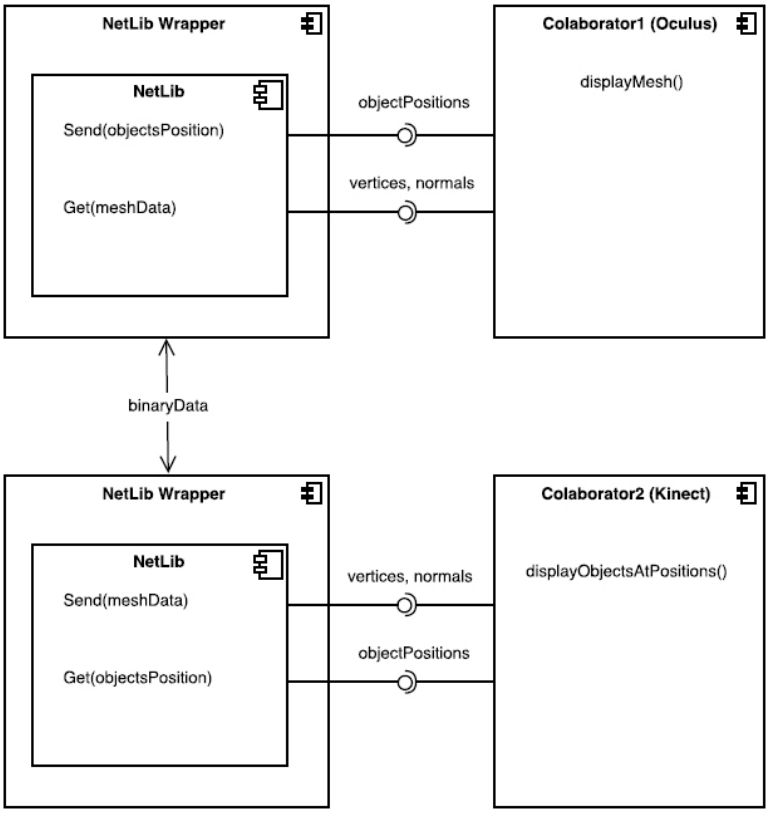


Figure 15: *Deployment diagram.*

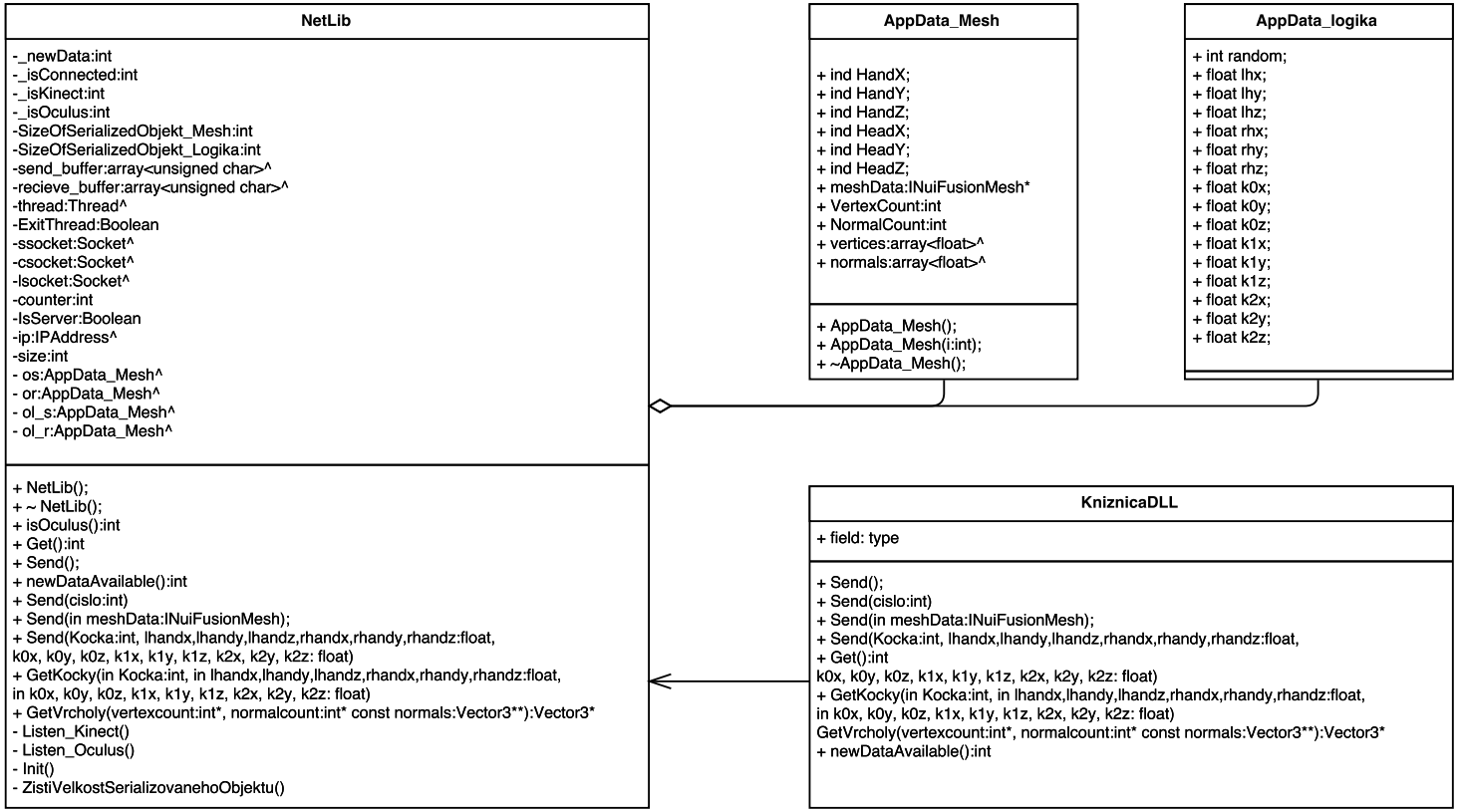


Figure 16: *Class diagram.*

## 7.5 Application

The final application consists of two minigames which can be switched during the run-time (see User Guide).

### Cubes

The goal of the first game in the picture 17 is as follows: a player are presented with three gray cubes, two spheres representing player's hands tracked by Kinect, and in the back, player sees a person which points to one of them. The person on the other side knows which cube they need to point to, as it is colored in green.

The objective is to lift up the cube which the person on the other side points to: if a correct cube has been lifted, the cube would turn green, otherwise it would turn red. A cube can be lifted with either hand by clenching the fist and dragging the cube with the fist clenched.

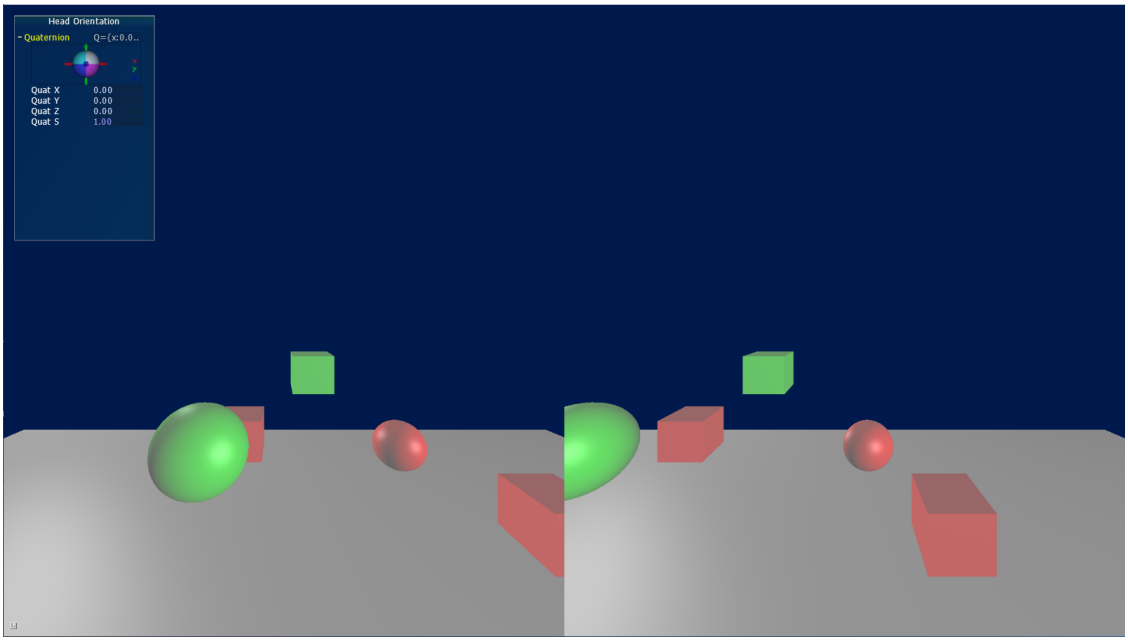


Figure 17: *Cubes.*

This game has three difficulty levels: the size of either the cube and the hand can be gradually made smaller to test the accuracy of Kinect.

### Hanoi towers

The second game is a game of Hanoi Towers as shown in the picture 18: the player is presented with three vertical sticks, and on the middle one, there are three cubes in ascending order of size. The objective is to stack the cubes on either left or right stick in the same order as in the beginning and only placing smaller cube on a bigger one.

The cubes can be moved in the same manner as in the first game. For user-friendlier experience, the cubes can be dragged out of the sticks in any direction, not only by lifting them out of the stick as in a physical version of this game, and the cube can be placed on a stick just by being in the correct position on the X axis. The cube will be snapped to the stick according to the X position of the cube, the Y axis will be adjusted by the amount of cubes on the stick. If the user would want to place a bigger cube onto a smaller one, the cube will be automatically snapped to its original position.

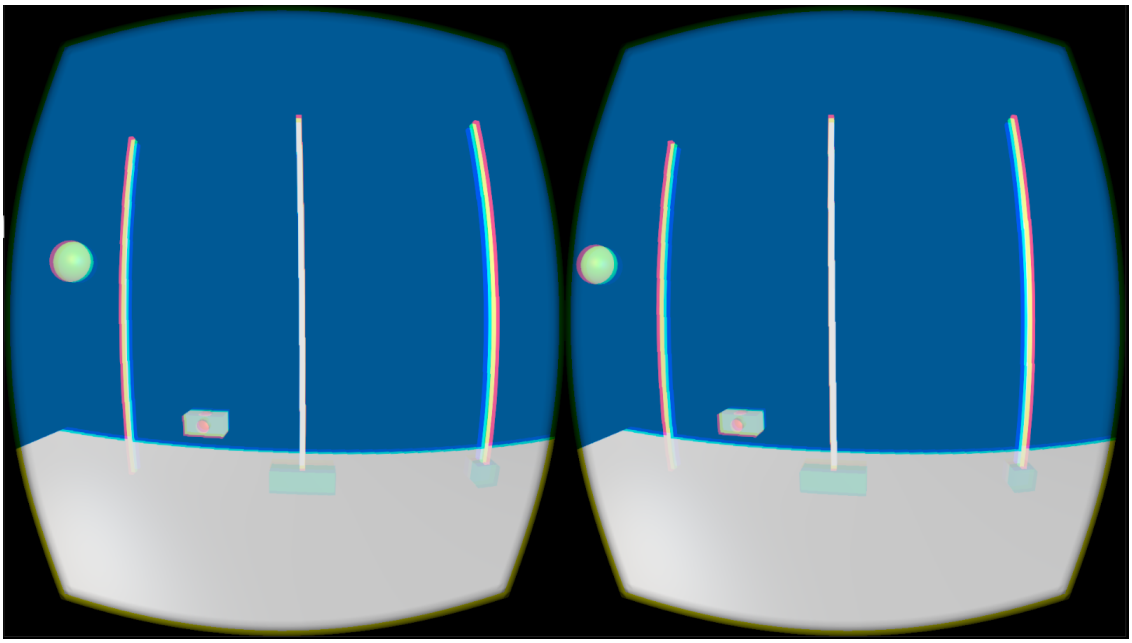

Figure 18: *Hanoi Towers.*

The number and the size of the cubes can be adjusted.

# 8 Evaluation

An experiment was conducted with 12 participants (11 men, 1 woman) aged from 19 to 28 years. All of them are students of Information technologies.

The goal of the experiment is to find limits of Kinect (what the smallest size of cube can be caught?) and response time. Subjective assessment is also taken into consideration.

## 8.1 Experiment scenario

Each participant should play two games (mentioned above) both with and without Oculus. In the cube game the person illustrated with mesh points at one cube. The participant should take the correct cube and pick it up. This procedure is repeated for three times with different sizes of cubes in descending way.

As the second part of experiment participants should win the Hanoi game. The last part of experiment is a short form with questions connected with accuracy and response time.

## 8.2 Evaluation of cube game

There are no significant difference in effectivity in marking the right cube according to mesh in work with and without Oculus. Average number of correct answers is 2,6 in work with Oculus and 2,75 without (from the maximum of 3). It was achieved quite good effectiveness.

There are problems with caught of the smallest cube, 58% participants working without Oculus and 8% working with Oclulus were not able to catch the smallest one.

In the figure 19 there is average number of attempts to catch the cube. Work with Oculus obtained better results because participants were able to catch the cube with smaller number of attempts. The most difficult was the smallest cube during work without Oculus.

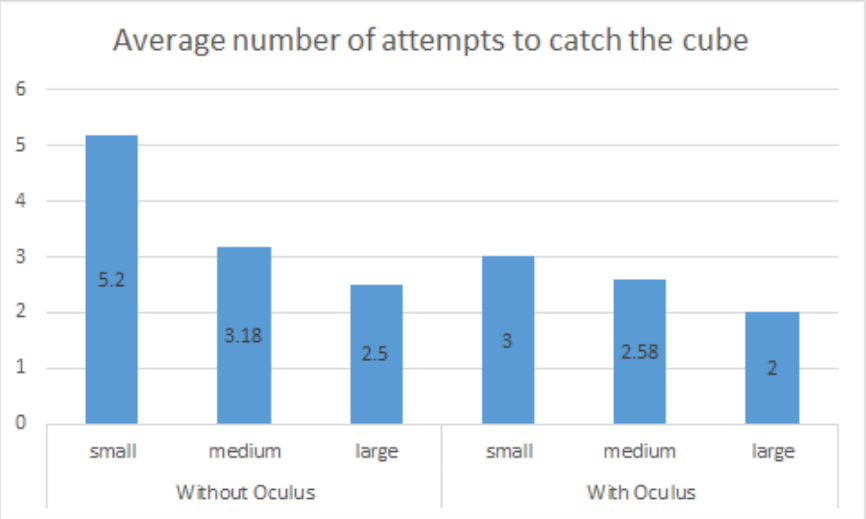


Figure 19: *Average number of attempts to catch the cube.*

## 8.3 Evaluation of Hanoi towers

It was noticed smaller amount of mistakes in the second game - Hanoi towers. Some falls of cubes were recorded: 1633

Error rate is illustrated on Figure 20.

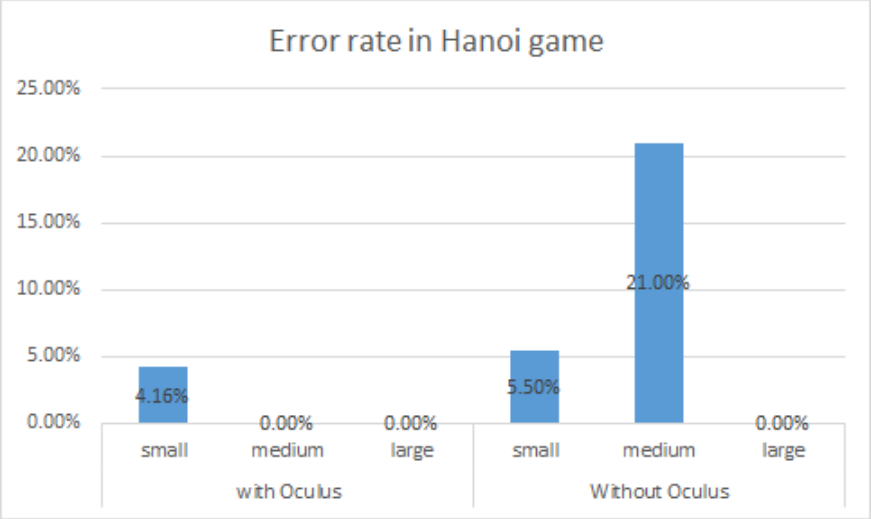


Figure 20: *Error rate in Hanoi game.*

## 8.4 Subjective evaluation

According to subjective evaluation 75% participants evaluated that work is better with Oculus.

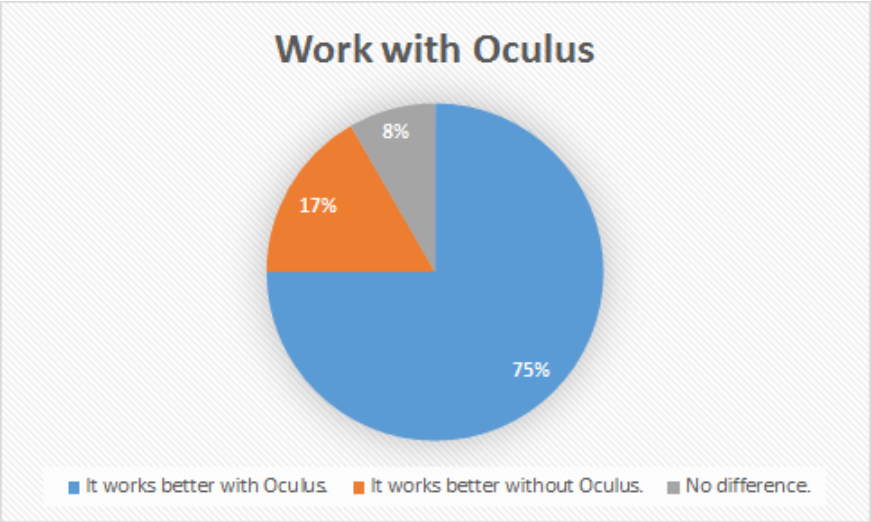


Figure 21: *Work with Oculus.*

Accuracy and speed of an application is assessed on the scale of 1 to 5 (from the best to the the worst). Accuracy obtained the average assessment 2,5 and speed obtained number 2.

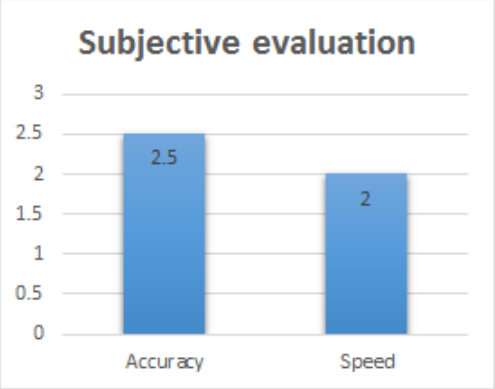


Figure 22: *Subjective evaluation.*

83% of participants does not have any problems but 17% of them experienced arm pain. Participants also noticed that with smaller cubes the work is harder.

There is also question about guesing in cube game. If the participant does not see where the person is poiting he can guess which cube it is. According to participants they are guessing maximum of two times from six possible. They noticed that when person is pointing to the middle cube it is not easily visible.

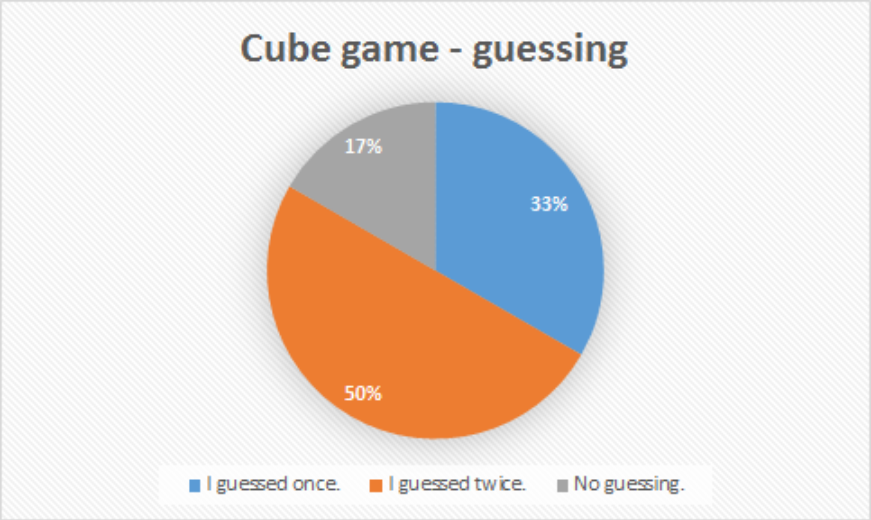


Figure 23: *Cube game - guessing.*

## 8.5 Time evaluation

The following graphs 24 and 25 depict average time which was needed to lift up a cube of listed sizes. As the graph shows, testers have had the most difficulties picking up a cube without Oculus mounted on, possibly because of worse depth perception.

Several tested subject were not able to lift the smallest cube, so these entries were not counted in average. The times listed in graphs are in seconds.

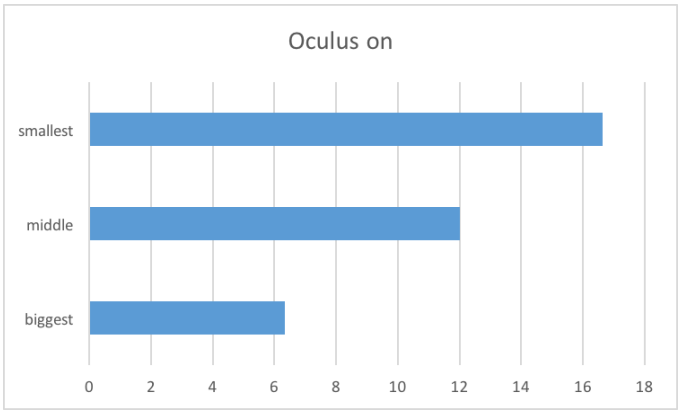


Figure 24: *Oculus on. Horizontal axis shows time in seconds.*

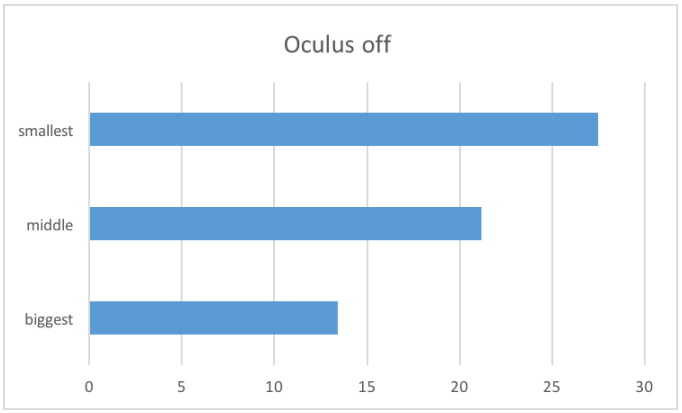


Figure 25: *Oculus off. Horizontal axis shows time in seconds.*

# 9 Summary

The aim of this team project was to get familiar with devices that can be used for creating an experience of fully immersive virtual reality and interacting within it. While the first semester was dedicated to analysis of the devices, the second semester focused on design, implementation and evaluating.

We succeeded at creating an application which combines interaction and collaboration with virtual scene. Its main goal was to find the limits of the technologies in use, namely Microsoft Kinect and Oculus Rift. We produced a user experience study which compares interaction within a 3D scene with and without the aid of a head mounted display.

The first part of development process was mainly modular. Three work groups were created to focus on selected parts of the final application. First team was dedicated to running the Oculus HMD, distorting the scene, adjusting scene for two eyes and creating the final scene to be used as the final product. Second team focused on analysis of Microsoft Kinect, receiving and post processing graphical data and using them for 3D scene reconstruction. In the end it was successfully possible to create a colourful reconstruction of the real people and objects in front of the device.

The third team focused on analysis of networking and implementation of a separate module capable of serializing and sending selected data over network. Thanks to this module, a client-server sort of application was created where two users could see each other within virtual reality.

The final application consisted of two mini games that were tested by several testers with satisfactory results. Even though they reported arm and head pain from the physical inconvenience resulting from using the devices, they also proved them to be useful at making interaction within virtual reality more accurate.

Even though the integration with 3DSoftviz was not successful, we created a product which tested the technological limits of Kinect and Oculus, and these can be taken in consideration in future development and integration.

# A Technical documentation

## A.1 Networking

### Pros and cons

The pros of the final communication module lie in:

- It is self-initializing, user/programmer does not have to worry about initializing the library or calling some init methods in the code.
- The library runs in separate threads.
- It offers methods for simple send and receive functionalities.
- Communication library communicates over reliable TCP communication protocol.
- For configuration it uses dedicated *.config file which is described later.

The possible cons of the library:

- Manual copying of desired data from native C++ to managed C++ which is a small bottleneck.
- Some attributes of serialized objects have fixed maximal length and when longer data are passed in, they are trimmed.
- Harder scalability. When there is a desire to send new types of data, communication library needs to be extended with new send() and receive() methods.

### Description of selected methods

In the text below, there is a description of the core methods which the communication module offers to the user.

Returns value 1, if the communication module is in the mode of the main user, who primary uses Oculus SDK2 device, and returns 0 if it is configured as a second collaborator, who uses Kinect. This method is used mainly in the main loop, for executing Oculus or Kinect specific methods, during different modes of the application.

- int isOculus();
  Returns value of 1 if the communication module is in the mode of the main user who primarily uses Oculus SDK2 device. Returns 0 if it is configured as a second collaborator who uses Kinect. This method is used mainly in the main loop for executing Oculus or Kinect specific methods during different modes of the application.

- void Send(INuiFusionMesh *meshData);
  This method encapsulate and takes care about data received from Kinect. Data extracted from the INuiFusionMesh structure are vertices and their corresponding normals.

- void Send(int Kocka, float lhandx, float lhandy, float lhandz, float rhandx, float rhandy, float rhandz, float k0x, float k0y, float k0z, float k1x, float k1y, float k1z, float k2x, float k2y, float k2z);
  Method sends information about position of objects in scene and main user's hands. It also sends ID of active object.

- Vector3* GetVrcholy(int *vertexcount, int* normalcount, const Vector3** normals);
  Method returns array of points, fills information about how much vertices and normals are returned. It also fills pointer to array with received normals.

- void GetKocky(int *Kocka, float *lhandx, float *lhandy, float *lhandz, float *rhandx, float *rhandy, float *rhandz, float *k0x, float *k0y, float *k0z, float *k1x, float *k1y, float *k1z, float *k2x, float *k2y, float *k2z);
  Method fills pointers with data related with scene objects such as which cube is selected, position of hands and cubes in the scene.

## A.2 Kinect

Within the testing process it was determined that there are certain features of the Kinect API that are not documented and cause problems in production code. The tracking of bodies is not deterministic in terms of indices. According to specification, Kinect is capable of tracking at most 6 people at same time. However,

after each Kinect API call to retrieve body data, the indices of tracked people may and may not change.

Even if there's only one person in the sensor frustum, Kinect may assign random index which may or may not be equal to an expected value of 0. For this reason, a helper method *Joint* ExtractJointsForPerson(int& index)* was created in *KinectData* class returning an array of Joint data along with modifying the input parameter index. This method iterates over all possible indiced in the array of captured bodies and checks whether a body is actually tracked.

Enum class *KinectTypes* provides a convenient way to select data types to be extracted from Kinect. The available options are *NoData, ColorData, DepthData, InfraredData, SurfaceData, BodyData, MeshData* and *ColorMeshData*. For programmer's convenience, | and & operators were overloaded for simple checking against *NoData* value.

The pros of the final communication module lie in:

- It is self-initializing, user/programmer does not have to worry about initializing the library or calling some init methods in the code.
- It is highly parametrized thanks to the KinectParameters class.
- It only obtains the data it is told to obtain.
- Clearly separates concerns.

The possible cons of the library:

- Possible usage of polymorphism is to be considered. The similar structures do not have a common predecessor that would be make this simpler. As a consequence, there is duplicated frame initialization code in multiple occasions.
- Adding new data type requires changes on several places.

# B User manuals

## B.1 Final application

The application is simply controlled via command line using keyboard, since graphical interface was not goal for this application.

There are two games which an user can choose from that are launched by pressing the corresponding letter:

- "k" - launches the game of Cubes.
- "h" - launches the game of Hanoi towers.

The objective of these games is described in the description of the application. The primary aim of the game of Cubes is the possibility of cooperation of two people using two Kinect devices. The game of Hanoi towers is known worldwide, and we wanted to emulate the feeling of moving objects in virtual space using classic gestures from real world. This interaction is provided by gripping the object with a hand when the sphere that represents the hand is in the object we want to grip.

During the game of Cubes, we can adjust the size of cubes and the spheres representing our hands. This function was added in order to test the Kinect and Oculus restrictions, and user's experience. The size of the cubes can be controlled during the execution of the application by pressing the following numbers on keyboard:

- 1 - the smallest cube
- 2 - middle-sized cube
- 3 - the biggest cube

The timer in Kinect sometimes causes glitches in tracking the position of person's body joints, which causes bad user experience when moving an object in scene (the sphere representing a hand jumps in scene leaving the gripped object at the original place). For that, we added additional control for software help with this issue by pressing following letters:

- "p" turns software help on (when the glitch happens, the object we are moving will be snapped to hand)
- "o“ - turns software help off (when the glitch happens, the object stays at the original position)

## B.2 Kinect prototype

The Kinect prototype module serves as a testing application for verifying output data streams. It is controlled via command line using keyboard with following modes available:

- 1 - color map (1920x1080)
- 2 - depth map (512x424)
- 3 - infrared map (512x424)
- 4 - surface map (512x424)
- 5 - skeleton

There are several options available to change the properties of reconstruction parameters:

- m - cycle through voxels per meter (128/256/384/512/640)
- s - save mesh to default location
- x - cycle through X-axis voxel resolution (128/256/384/512/640)
- y - cycle through Y-axis voxel resolution (128/256/384/512/640)
- z - cycle through Z-axis voxel resolution (128/256/384/512/640)
- left arrow - decrease minimum depth
- right arrow - increase minimum depth
- up arrow - increase maximum depth
- down arrow - decrease maximum depth