VIRTUAL PHOTO ART GALLERY

19th August 2010

Ainara Azcona - MSc Creative Software Systems

Supervisor: Ruth Aylett
STATEMENT OF NON-PLAGIARISM

"I, Ainara Azcona, confirm that this work submitted for assessment is my own and is expressed in my own words. Any uses made within it of the words of other authors in any form e.g., ideas, equations, figures, text, tables, programs etc are properly acknowledged. A list of references employed is included."

19/08/2010          Ainara Azcona
ABSTRACT

This dissertation presents the creation of a photo art gallery as a virtual environment. In this environment the user has the ability to look around and walk around, as well as the ability to interact with the gallery, locating his own photographs in empty frames on its walls.

The visualization of the gallery is done through a passive stereo system consisting of two projectors, a screen and stereo glasses. A dance mat is the device chosen for the navigation interface and a real time motion capture system, consisting of infra-red cameras, is used for the interaction.

Everything is integrated using the open source 3D graphics toolkit OpenSceneGraph and programmed in C++. The 3D models are modelled using the software 3ds Max.

This report describes the two prototypes created as part of two cycles of its iterative development process. First one is a very basic gallery with just the navigation feature on it. In the second one, the navigation feature is improved and an interaction is added.

Evaluation shows that this innovative way of navigating and interacting in a virtual gallery is fun and enjoyable by users.
ACKNOWLEDGEMENTS

First, I would like to thank Professor Ruth Aylett, my supervisor, for suggesting this project and guiding me through it.

I would also like to thank Tony Brooks and Jan Ciger, from the Sensoramalab at Aalborg University (Denmark) for giving me some advice about the integration of the dance mat in the system.

I would also like to thank Michael Kriegel for showing us how to use the cameras and their software.

I would also like to include here “the twins” Alex and Markus, for giving me a hand when I needed it and teaching me some German even when I didn’t want to. Thanks to Markus for lending me his yellow tracking finger.

I would also like to thank the students working on lab 1.53 for their constant noise and disrespectful behaviour towards the ones who, like me, need silence to work.

A big thank you to all the people who took part in the evaluation process: Alex, Markus, Michael, Mei, Maria, Florian and Ruth.

A special thank you to my friend Maria, for being always there and for being a great friend.

And finally, and most important, I would like to thank my family and friends, specially my parents, for their continuous support and encouragement all along this tough year.
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>STATEMENT OF NON-PLAGIARISM</td>
<td>1</td>
</tr>
<tr>
<td>ABSTRACT</td>
<td>2</td>
</tr>
<tr>
<td>ACKNOWLEDGEMENTS</td>
<td>3</td>
</tr>
<tr>
<td><strong>CHAPTER 1.</strong> INTRODUCTION</td>
<td>7</td>
</tr>
<tr>
<td><strong>CHAPTER 2.</strong> LITERATURE REVIEW</td>
<td>9</td>
</tr>
<tr>
<td>2.1 VIRTUAL REALITY</td>
<td>9</td>
</tr>
<tr>
<td>2.2 DISPLAY SYSTEM</td>
<td>10</td>
</tr>
<tr>
<td>2.3 INTERACTION WITH THE VIRTUAL GALLERY</td>
<td>12</td>
</tr>
<tr>
<td>2.4 EXISTING VIRTUAL GALLERIES</td>
<td>19</td>
</tr>
<tr>
<td><strong>CHAPTER 3.</strong> DESIGN REQUIREMENTS</td>
<td>23</td>
</tr>
<tr>
<td>3.1 MODELING OF THE GALLERY AND ITS COMPONENTS</td>
<td>23</td>
</tr>
<tr>
<td>3.2 NAVIGATION SYSTEM REQUIREMENTS</td>
<td>23</td>
</tr>
<tr>
<td>3.3 INTERACTION SYSTEM REQUIREMENTS</td>
<td>24</td>
</tr>
<tr>
<td>3.4 GALLERY APPEARANCE REQUIREMENTS</td>
<td>25</td>
</tr>
<tr>
<td><strong>CHAPTER 4.</strong> TOOLS AND EQUIPMENT</td>
<td>26</td>
</tr>
<tr>
<td>4.1 INPUT DEVICES</td>
<td>26</td>
</tr>
<tr>
<td>4.2 VISUALIZATION SYSTEM</td>
<td>29</td>
</tr>
<tr>
<td>4.3 MODELLING PACKAGE</td>
<td>29</td>
</tr>
<tr>
<td>4.4 OSG</td>
<td>29</td>
</tr>
<tr>
<td><strong>CHAPTER 5.</strong> FIRSTPrototype</td>
<td>30</td>
</tr>
<tr>
<td>5.1 DESIGN</td>
<td>30</td>
</tr>
<tr>
<td>5.2 IMPLEMENTATION</td>
<td>34</td>
</tr>
<tr>
<td>5.3 EVALUATION</td>
<td>42</td>
</tr>
<tr>
<td><strong>CHAPTER 6.</strong> SECOND Prototype</td>
<td>44</td>
</tr>
<tr>
<td>6.1 DESIGN</td>
<td>45</td>
</tr>
</tbody>
</table>


6.2 IMPLEMENTATION 48
6.3 EVALUATION 55
CHAPTER 7. CONCLUSIONS AND FURTHER WORK 58
REFERENCES 61
BIBLIOGRAPHY 62

APPENDIX 1. PINNACLE GAME PROFILER AND QUALISYS TRACK MANAGER
APPENDIX 2. EVALUATION QUESTIONNAIRES
APPENDIX 3. FIRST PROTOTYPE CODE
APPENDIX 4. SECOND PROTOTYPE CODE
## TABLE OF FIGURES

Figure 1: Iterative development process.
Figure 2: Projectors [7]
Figure 3: Projecting screen in the classroom [7]
Figure 4: Virtual table tennis system [9]
Figure 5: Sensors in glasses and table tennis racket [9]
Figure 6: Drawing application [10]
Figure 7: Virtual Rap Dancer system [11]
Figure 8: Dance mat interface in SMILE [12]
Figure 9: View of Sysyggy Image Viewer gallery.
Figure 10: Tradky gallery [15]
Figure 11: Gallery by Cando [16]
Figure 13: Users of “The Empty Museum” and their avatars in the virtual environment [17]
Figure 12: Blue object indicating where the action is [17]
Figure 14: Dance mat
Figure 15: IR camera
Figure 16: Cameras distribution
Figure 17: Lab projectors
Figure 18: Gallery floor plan sketch
Figure 19: Dance mat buttons actions
Figure 20: Gallery model
Figure 21: Dance mat mapped into keyboard
Figure 22: Glasses with markers attached
Figure 23: Glasses 6DOF body
Figure 24: Gallery appearance
Figure 25: Gallery view
Figure 26: Height calibration
Figure 27: Frame 1
Figure 28: Frame 2
Figure 29: Photo box model
Figure 30: Finger marker
Figure 31: Finger marker
Figure 32: Hand Calibration
Figure 33: Gallery original aspect.
Figure 34: Process of selecting a photo
Figure 35: Picture placed on a frame
CHAPTER 1. INTRODUCTION

Photography, coming from Greek, means drawing with light. Photography is an art, a communication tool, and a way of capturing a moment.

Nowadays, the digital technology has revolutionized photography. Photos are everywhere and can be taken at any time, in any place and under any circumstances.

Years ago photographs were taken, developed and placed on an album or frame afterwards. However, “what one can do with images when they are digital is quite different” [1]

Today, photography is affordable for every one, and maybe because of that, or because people just enjoy taking pictures, it has become a really common thing. Thousands photographs are shot every moment. Most of these pictures just get lost in digital storage devices though.

Moreover, as stated in the report by many researchers [1]:

“The activities we undertake and the goals we have in mind when we take photos and share them, then, are not at all the same now as they were even five years ago.”

The aim of this MSc project is to create an interactive Virtual Photo Art Gallery. The idea of creating a virtual photo art gallery comes from the necessity of being innovative in a field where almost everything is already invented. The idea of having a look at photos in a different way and interact with them in ways not possible in real world sounds quite appealing; and Virtual Environments seem like an ideal area to create such a gallery.

The main objective is to design, define and develop this interactive virtual photo art gallery in which the user can navigate and interact. The navigation involves walking around the gallery and looking around it. The gallery’s interactive feature is the ability of positioning pictures in different empty frames around it. It can be divided in four major objectives:
- Modeling of the gallery and its components.
- Navigation system implementation.
- Interaction system implementation.
- Gallery realistic appearance creation.

The development of this project is done following an iterative process. The first step is the gathering of functional requirements for the application to be developed. Next steps are inside the development loop consisting on design, implementation and evaluation. This way, the design is progressively refined during a series of cycles getting feedback from the evaluation and applying it to the next design.

![Iterative development process](image)

*Figure 1. Iterative development process.*

Due to the limited time available for the development of this project, I am only going through two iterations. The first one will produce a first prototype with a simple gallery with only the navigation features included on it. The second prototype will have the navigation features and the appearance of the gallery improved, as well as the interaction features introduced.

Next chapter covers the relevant literature review and already existing virtual art galleries. Chapter 3 describes the design requirements for this virtual gallery. The following chapter, number four, goes through the resources used in the creation of the application. The next two chapters, five and six, cover the design, implementation and evaluation of prototypes one and two respectively. Finally, in the last chapter the conclusions are described together with the achievements and suggestions for future work.
CHAPTER 2. LITERATURE REVIEW

2.1 VIRTUAL REALITY

Different authors define virtual reality in different ways. There is no a universal definition for it but many points of view by different researchers.

Sherman and Craig [2] define it as a “new medium brought about by technological advances” and compare it with other mediums along history used by humans to communicate and express ideas. Under their point of view, there are four key elements in a virtual reality experience:

- Virtual world
- Immersion
- Sensory feedback
- Interactivity

Vince [6] states that “Virtual Reality is about using computers to create images of 3D scenes with which one can navigate and interact.”

Nowadays virtual reality is still associated by the general public with head mounted displays popularized during the 90s in movies and the media. However there is more than that going on in this field with people working and researching in many of its different areas.

As essential parts of a virtual environment, in the next sections I will go through the display system chosen for the virtual gallery and the interaction with it; and finally I will present some existing virtual galleries.
2.2 DISPLAY SYSTEM

Sherman and Craig [4] describe the display systems:

“A key component of a virtual reality experience is how the user perceives the environment. Their physical perception of the virtual world is based entirely on what the computer displays. (…) VR systems fool the senses by output of computer-generated stimuli rather than natural stimuli to one or more of the senses”.

Therefore, the properties of the display system are a very important aspect of the whole VR experience.

The display system used for the visualization of the virtual gallery is a stereo projection system in which the images are projected to a screen from two different projectors. The user needs to wear stereo glasses in order to see the virtual world in 3D.

This system is based on how the human eyes (and brain) see in three dimensions. Both our eyes perceive the same images but with a slightly different angle due to the distance between them. The brain processes, then, this information to get the stereo vision.

The projectors are one on top of the other projecting to the same screen. Each of them projects the visual information for each eye using different polarization filters. The glasses worn by the user have the right polarized filter on each lens so each eye only sees the information displayed for it.

This kind of display systems with two projectors has many advantages: “offers the benefit of a brighter, higher resolution image, due to their combined effect [4], it also offers greater user mobility and a wider field of view. Besides, polarized glasses are a non expensive device.

The use of this display system in virtual environments is quite common nowadays. As an example I am going to describe the system successfully utilized by Dvorák et al. in their project “Boosting up Architectural Design Education with Virtual Reality” [7].
The virtual environment system they present was developed in order to help architecture students and lecturers. They chose to use virtual reality because of its “unlimited possibility of the object exploration in real time with high levels of details.”

The system is based on two computers each of them generating the visual information for each eye. “The output is generated for a passive front stereoscopic projection using two data projectors”. This output is polarized so stereoscopic polarized glasses are needed [7].

![Figure 3: Projecting screen in the classroom [7]](image)
2.3 INTERACTION WITH THE VIRTUAL GALLERY

Interaction with a virtual environment is the reciprocal actions between the user and the virtual environment. The virtual environment reacts to user input and vice versa.

Speaking about the interaction with virtual worlds Sherman and Craig [5] argue the following:

“Interaction with a virtual world is a key ingredient of a VR experience. Indeed, if the display of a virtual world does not respond at least to a user’s physical movement, then it is not considered virtual reality.”

How interaction is performed depends on how the interface is designed. Humans can learn and adapt to new ways of interaction [5], leading this into a wide range of options for interacting with virtual environments.

If interaction is essential in a VR system then having an input device is essential too. The input process involves tracking the user’s position and monitoring his actions. The user “influences the virtual world through the system’s input interface” [3].

The technology also plays an important role in the virtual world interaction. Nowadays the selection of devices available to be used as a way of getting input from the user is very big, mainly due to great advances in technology.

Talking about input devices, Sherman and Craig [3] state:

“When a user interacts with a virtual world via a VR experience, they often both receive information and sensations and transmit them via input devices to the computer”

There are three ways of interaction within a virtual environment: manipulation, navigation and communication. Manipulation implies the ability to modify the world and its objects, navigation refers to the action of moving around and exploring, and communication is the contact with virtual agents or other users.

In this project I am only going to manipulate the virtual gallery and navigate through it, there will be no communication with virtual agents or other users. These two interactions and the way of getting the input necessary for them are described in the next two sections.
Manipulation of the virtual Gallery

Manipulation of a virtual environment refers to the ability of modifying it or its objects. In a virtual world no physical laws or forces apply so the manipulation of the objects can be designed in any way the designer wants even if it has no similarities with the real world manipulations.

In this case, the objects manipulated are the photographs which will be located in different frames around the gallery. The way of doing this will be explained in the next chapters; but between the manipulation methods explained by Sherman and Craig [1.inter] direct user control or virtual controls are the ones that I kept in mind when designing the interface. The former is the method in which the user “interacts with objects in the virtual world just as they would in the real world”; and the latter refers to computer-generated representations of physical buttons, trackballs or steering wheels [5]

Something to keep in mind is that “without physical feedback, it can be very difficult to sense” when a “control has been activated” [5] so some kind of feedback should be introduced.

In order to perform these manipulations in the virtual world, the system has to receive information about the actions of the user. To do that I need a tracking system, which provides user’s body posture and gestures information.

The tracking system chosen is an optical tracking system. It is a real time motion capture system, which consists on a set of infrared cameras and some markers attached to the participant. The markers are small balls whose material reflects the infrared light. They are attached to the user’s parts of the body which want to be tracked. The cameras emit infrared light that is reflected by the markers and recorded by the same cameras, then processed by a software and transmitted to the system which interprets this information.

The number of markers and the body parts to attach them will be explained in the next chapters. The first idea was to track both hands and the head. About the cameras and number of markers one aspect to consider is that, as Foursa states [8], “line of sight” is required between them. Therefore, “the orientation of the cameras must be such to ensure that markers are always visible” [6].

The two main advantages of this tracking system are that it “can work fast over a large area and be comfortable to use”[8].

IR tracking is a very common tool for motion capture in animation cinema and gaming. In these cases the actions performed by human actors are transferred to animated 3D characters, and one important aspect is that the capture has to be accurate so many cameras and markers are used. But in VR systems the motion capture has to be done in real time, so the number of cameras and markers used has to be the minimum to ensure a reliable tracking and a fast processing.
This motion capture technique has been widely used in virtual environments. Two projects which make use of it are described next.

One virtual environment which uses these tracking elements is the virtual table tennis application described in the paper by Rusdorf and Brunnett [9]. In this project, they used four infrared cameras and two sets of markers.

The elements tracked were two: the polarization glasses and the table tennis racket (Fig. 4). They had into consideration the characteristics of the tracking system to decide how many markers were necessary:

"The system needs a minimum of 4 markers per target. In order to achieve robust tracking 6 markers are used for the racket. For the glasses we use a standard target construction with five markers." [9]

In this project, a special attention was paid to the tracking of the targets as table tennis is a very quick sport with fast movements; and thus the chance of the markers being out of the sight of the cameras is bigger. They studied different configurations with three, four and five cameras doing the tracking of the targets. The results were acceptable with three cameras and improved considerably with four. Adding a fifth camera improved again the reliability of the tracking system but the differences between having four or five cameras were quite small.
Kavakli et al. [10] present a virtual environment which purpose is to perform design operations. One part of this project was a drawing application (Fig. 5). To accomplish this, they use a motion capture system with four cameras to track up to LED markers.

In this project, however, the cameras did not emit infrared light, they were only able to record it, so an infrared emitting object was necessary as a marker. The object selected is a LED (Light Emitting Diode) which the user holds in his hand and uses to draw.
Navigation through the virtual gallery

According to Vince [6], “navigation allows the user to make their way through the world”. It is a crucial aspect of virtual environments as it is the way in which the user explores them, a “vital part of a participant’s experience” [5].

Normally, navigation involves two tasks: travel and wayfinding; the former refers to how user moves through space and the latter to how user knows about his position and heading direction [5].

Sherman and Craig [5] explain that the use of physical movements in virtual world’s navigation, and specially those that are used in the real world, make the interaction more natural and the user experience more intuitive and easy to learn. According to this, the most natural and intuitive way of exploring a virtual space would be just to walk around it. However the space and tracking limitations make this not always possible.

In my case, the space available for the user to move around is quite limited so I had to think about other way of navigation. But, as Vince [6] affirms “it is possible to navigate a virtual environment without physically moving” -understand moving as changing users location- , all that is necessary is to instruct the user to use other movements in order to walk in the virtual space without physically walking.

Between the many devices available for interacting with virtual environments the one I have chosen for the navigation through my virtual gallery is the dance mat.

The dance mat provides an interface in which the physical movement of stepping is used imitating the real walking and, at the same time, keeps the user in the same location all the time.

A dance mat is a low cost device very popular in gaming. It is a pad with a 3x3 set of square keys which have pressure sensors that detect when the user steps on them. These are some arrow keys (up, down, right and left) and other actions keys.

It is not new to virtual environments; it has already been successfully used in some projects. The dance mat has been used to travel around a virtual learning environment or to guide a virtual agent, between others.

Reidsma et al. present the Virtual Rap Dancer [11], an “interactive and entertaining embodied agent”. The user interacts with this virtual agent guiding him in the dancing. A dance mat is used in this project, between other input devices, “to keep track of the feet of the human dancer”.

16
Adamo-Villani and Jones did a user study of three travel interfaces for an immersive virtual learning environment for children (SMILE) [12]. One of these travel interfaces was a dance mat. The study reports that this body-centered technique is “easy to comprehend and use”.

In this virtual learning environment, the front, back and side arrows of the dance mat were used to move forward, backward, left and right respectively. Stepping on two arrows at the same time resulted in a diagonal move. The diagonal arrows were used for clockwise and anticlockwise rotations; and the center button disabled the rest of the buttons. This shows how versatile this interface is, the buttons can be programmed in many different ways having as a result many different ways of implementing the navigation.

The use of the dance mat for the navigation through a virtual gallery presents many advantages. Using the feet in this task, allows the user to use his hands for other interactions. LaViola Jr. et al. [13] state that in many cases “offloading interaction from the user’s hands to the feet can improve an interface, especially when user’s feet are used in navigation tasks”. Moreover, the use of the feet to control how to move around a virtual
space is a locomotion metaphor, which makes it easier to learn and improves the familiarity of the user with the system.

However, there is also one disadvantage: “the user is required to continuously step on the buttons and this can lead to fatigue and loss of balance” [12].
2.4 EXISTING VIRTUAL GALLERIES

In order to document myself for this project I have looked at different already existing virtual galleries. Some of the most interesting ones I have come across are described next.

The first three described below are desktop or internet application in which the gallery is displayed using the computer screen and the interaction happens through the keyboard or mouse. These are non immersive systems so the sense of presence is really low. The last one however, is a totally immersive system.

- “Sysygy Image Viewer” by Godlike Software. [14]

It is a desktop application. This freeware offers the user the possibility of uploading his photos and after that seeing them in a gallery. The gallery can be selected from a number of predetermined ones.

The user can walk around the gallery using the keyboard and the mouse; and look at the photos while listening to music. There are also some random characters walking around the virtual gallery and making comments about the photos. It is a weird thing but it makes it fun. One thing to highlight is that the graphics in this gallery are quite good.

The application also supports a multi-user mode connecting the gallery to a server.

Figure 9: View of Sysygy Image Viewer gallery.
• “Virtual Art Galleries” by Tradky. [15]

This gallery is a web application. It is a multi-user virtual gallery in which the user can navigate having a look at the pictures and interact with other users and artists.

The navigation can be done through the mouse or keyboard. The wheel with the arrows in the bottom part of the screen can be used as a car steering wheel.

This gallery does not offer the chance of uploading you own photographs, the normal user can only navigate through exiting galleries. However, the content can be modified by the user selecting what he wants to see. Artists can contact the company in order to see their work exhibited.

It is very interactive, apart from modifying the content of the virtual environment the user is able to contact other users and authors. By clicking on each piece or art the user gets redirected to a web with information about the piece and its author. There is also a chat available to talk with other visitors of the gallery, represented by avatars.

The disadvantage of this virtual gallery is the waiting time for the navigation.

![Figure 10: Tradky gallery [15]](image)

• “Virtual Art Gallery Creator” by Cando Interactive. [16]

It is a very-simple and easy to use desktop application. It allows the creation of personalised galleries with your own images.

The process is really simple: the first step is to upload the photographs; next step is to choose a layout for the gallery; and third one is to place the uploaded photographs in the gallery. Once the gallery is completely set up, the virtual environment is created within seconds and the user is able to navigate through it. The user can move around the gallery
and also get information (shown at the bottom of the screen) about each image just by focusing at them.

In order to navigate, the user utilizes the mouse and the arrow keys of the keyboard. It also has the possibility to associate and audio file to each of the images, which will be heard when the user gets close to each of the images displayed.

As seen on its website [16], this virtual gallery is used by different artists and photographers. Once they have created their own gallery, they upload it to a website so everyone can see it and navigate through it.

![Figure 11: Gallery by Cando.][16]

- **“The Virtual Art Gallery” inside “The empty museum” project, by Hernandez et al [17]**

This totally immersive virtual environment application shows multimedia art content in a virtual gallery. The application is multi-user and users can physically walk in a space. The display devices used are head mounted displays (HMDs) and the tracking system is a system which captures wireless movement. The user has to carry in a rucksack one of the part of the system: the rendering engine and the visualization hardware. This connects via wireless with the main part of the system responsible of controlling position of the users, assigning worlds to the users, calibration, etc. See all details here [17].

Hernandez et al. [17] describe how the user experience is in this virtual art gallery:

“When entering the world, the user is immersed in a singular architectural scenario composed of stone and concrete elements within an open space. After a welcome voice message, he/she may see that some semi-transparent blue objects surround
him/her. These blue areas tell them where the action is. Each one of them corresponds to a different art. As the user approaches, an event that sets in motion a series of behaviours related to the art in question takes place.”

Users are represented with avatars in the virtual museum. The avatars head follow user heads movement, “improving the presental feeling of the user inside the world, together with the perception of the other users, instead of being mere decorative elements.”[17]

Figure 12: Blue object indicating where the action is.[17]

Figure 13: Users of “The Empty Museum” and their avatars in the virtual environment.[17]
CHAPTER 3. DESIGN REQUIREMENTS

The first step of the Virtual Photo Art Gallery development process was to carefully think about the user and functional requirements for the virtual gallery. The research done for the literature review chapter, reading and documenting myself about virtual environments and interaction and navigation techniques and interfaces, gave me understanding about how a virtual environment should be designed.

There are many aspects that have to be considered to create a good virtual application. The ones that I thought about to design my gallery are summarised in the following subsections.

3.1 MODELING OF THE GALLERY AND ITS COMPONENTS

For the modelling of the gallery and its elements there are some specific requirements to follow.

First, the 3D model should not be formed by many polygons. The world is constantly being rendered so the number of polygons has to be kept as low as possible to not overload the system.

Talking about its design, the gallery has to be a spacious place with many walls where frames can be hung. It has to be kept as simple as possible without many rooms and corridors so the user does not get lost.

3.2 NAVIGATION SYSTEM REQUIREMENTS

As seen in the previous chapter, navigation is a key aspect to any virtual environment and it involves two tasks: travel and wayfinding. Requirements for these two actions are described next.
For the travel part, the navigation interface should be as natural as possible, and as easy to learn and intuitive as possible. In order to get that, and as previously seen, the interface has to involve some physical movement, and this movement should have some similarities with the real world action, walking. It is also important to maintain the user at a proper height above the ground.

For the wayfinding part, one important aspect to bear in mind is that he user should easily get a mental model of the environment so he knows where he is and where to go at any time. This way the user could get oriented at any time, not getting lost. The user should be located at the entrance of the gallery so he knows all the possible ways around from there.

One of the features which improves the navigation is the tracking of the user's head. By doing this, the view changes as user moves the head to look around. This system has to be very accurate and the corresponding change in the view very precise. Failure to do so would make the experience very annoying.

Another important aspect to take into account is that in real life, people don’t walk through walls. In a virtual environment, however, no physical laws apply and, if not specified, user can move all around no matter how many obstacles he comes across. In order to make my gallery more realistic and avoid user getting lost, a wall avoidance function implementation is required and collision detection has to be considered.

3.3 INTERACTION SYSTEM REQUIREMENTS

Being able to manipulate a virtual environment is also a key aspect of it. In this case, the objects to manipulate are the photographs to be located in different frames around the gallery. The hand gestures will be used for this purpose.

As well as mentioned above for the navigation interface, the manipulation of the gallery interface should be intuitive and easy to learn. By contrary, there is no need to mimic real world actions so there is more freedom for its design.

Another aspect to take into consideration is that in the manipulation of a virtual environment the user gets no physical feedback so the introduction of some kind of visual feedback is required.

The tracking is done in real time so the number of markers should be the minimum which guarantees a reliable tracking and a fast processing.
3.4 GALLERY APPEARANCE REQUIREMENTS

The gallery has to look nice and realistic so it provides a more immersive experience. A careful lighting and texturing is required to provide a nice and real appearance.

The lighting is especially important in an art gallery; it should provide a nice and relaxed atmosphere but at the same time, it should be bright enough to be able to appreciate the photographs properly.

One last requirement is to make each room of the gallery look different so the user finds it easier to know where he is. Different colours for each room would make it.
CHAPTER 4. TOOLS AND EQUIPMENT

4.1 INPUT DEVICES

4.1.1 Dance Mat

As mentioned in the Literature Review chapter, the device chosen for the navigation through the gallery is a dance mat.

A dance mat is a popular game controller. It is a pad with keys on it which are activated when the user steps on them. The particular one I am using for this project is an Xbox one (shown bellow); it has eight arrow keys and two additional keys for “Start” and “Select”.

![Figure 14: Dance mat](image)
This device is designed so it can be connected to the game console. However it also has an USB connector so it can also be connected to a computer. The way of getting the input from it into the computer is not a straight away thing however; it required a lot of research.

After going through a lot of research I got three possible solutions which are explained next:

- **SDL library**

  As states in its official website [20] “Simple DirectMedia Layer, or SDL for short, is a library that allows you low level access to a video framebuffer, audio output, mouse, keyboard, and joysticks across a wide variety of operating systems.”

  It supports Windows and it works with C++ so it sounded like a good option.

- **Direct Input and Xinput**

  DirectInput is a component of Microsoft DirectX application programming interface. As read of its website [21]: “DirectInput is an API for input devices including the mouse, keyboard, joystick, and other game controllers”.

  XInput is also mentioned on the same website [21] “XInput API has been introduced for Xbox 360 controller support on Windows”.

- **Pinnacle Game Profiler**

  Pinnacle Game Profiler is a small piece of software which maps the input from different games controllers into keyboard or mouse input.

  As described on its website [22] “Pinnacle allows you to program what happens when you interact with your game controllers, keyboards, and mice. You tell Pinnacle which input devices to watch and how to react. “

  I went for this last option as it seemed the easiest one. I used the option of mapping the input of the dance mat into keyboard input.

  It is also a good choice as keyboard can be used while implementation is done with no need to be using the dance mat all the time
4.1.2 IR cameras

The system is compound by five infra-red cameras and a laptop with the software to process the information they send. The piece of software is called “Qualisys Track Manager”.

The information about the markers position is processed in this laptop and then transmitted to the main computer. In order to be able to go inside that information Qualisys provide a C++ program with all the code needed to do so, it is named RT_Client. This program needed to be integrated with my OSG program.

We were three students working with these cameras at the same time so we needed to find a configuration which worked for the three of us. I only needed to track the head and the hand, and as my user will stay static on top of the dance mat. My colleagues had more restrictive situations so we discussed about it and found the configuration working fine for everyone.
4.2 VISUALIZATION SYSTEM

As seen in the Literature Review Chapter, the visualization system used in this project is compound by two projectors, a screen and the glasses the user needs to wear.

The projectors project two images slightly offset, one for each eye. Each image is polarised in a different way, and then the glasses filter them so that the user gets a 3D sensation.

![Lab projectors](image)

Figure 17: Lab projectors

4.3 MODELLING PACKAGE

The modelling package used for this project was 3ds max, from Autodesk.

4.4 OSG

The whole system is implemented using Open Scene Graph which is in C++, using Visual Studio.
CHAPTER 5. FIRST PROTOTYPE

As previously mentioned, this first prototype will create a simple application in which only the navigation features are implemented.

5.1 DESIGN

5.1.1 Gallery Design

Taking into consideration the requirements mentioned above for the design of the gallery, and thinking about some galleries I know and some others I saw online, I designed a small gallery with for rooms. In view of the purpose of this gallery being only the host of photographs on its walls, its design involves nothing but blank walls. The initial sketch of its floor plan can be seen below.

![Gallery floor plan sketch](image)
5.1.2 Navigation System Design

The navigation system is divided in two main aspects which are described next. The ability to walk around the gallery whose interface is the dance mat (most of the times referred as simply navigation), and the ability to look around it. For this last aspect, the tracking of the user head is necessary.

- Ability to walk around - Navigation

Designing the navigation system was not an easy task. I needed to design a system in which input comes from a dance mat for something that human beings do more or less unconsciously.

As previously mentioned, navigation involves two tasks: wayfinding and travel. Wayfinding was not really taken into consideration when designing the navigation system. The gallery is designed in a way that prevents loser from getting lost. It has a very simple layout and the user is meant to know where he is and how to reach the rest of the rooms at any time. That said, the navigation system design was focussed on the travel aspect.

When we want to move from one position to another, we just think where we want to head to and move towards that position without actually thinking about how we do it. If the point we want to reach is not exactly in front of us, we rotate and move; but we don’t first rotate and after that start moving forward, it is something we just do in one movement.

However, this has to be translated to a ten-button dance mat in which actions can only happen once at a time. The mat has eight arrows and two additional buttons (see Chapter 4). The actions attached to each of them should somehow match their meaning, it makes no sense to step on one arrow which points to one direction and move towards other.

The most logical design was to set the four main arrows (blue and pink) for moving in the longitudinal axis and lateral axis respectively. Nevertheless, as Sherman and Craig [5] state: “Users rarely want to move in a straight line. Therefore, some means of controlling direction of travel is highly advantageous.” Thus, I needed a way to add rotation on the navigation system. This way, user would be able to head towards all the directions.

Considering that we don’t really move laterally much; and rotation plus straight movement is much more common, I decided to change the distribution of the actions on the dance mat. I went for using the two blue arrows for rotation to the left and rotation to the right respectively, leaving the lateral movement for the diagonal arrows.

The final design for the navigation is at follows:
Even if this is not the way we navigate in normal life, to me it appears like the most natural and intuitive way of doing it with a dance mat. It might seem a bit awkward for the user to navigate in two separate movements, but it was the best solution I found.

An additional idea was to use some of the remaining buttons of the dance mat for diagonal movement but this idea was quickly discarded. Diagonal movement can be easily achieved by using the rotation and then straight movement. Besides, the less buttons the user has to remember about the better, I did not want to overload the user.

Another idea which I thought about was to combine buttons to get the desired movement. In this way, the user would step on two arrows at the same time getting as a result the combination of both movements. This was also discarded as it would not be intuitive or natural, apart from the fact that the user could lose balance really easily.

Another aspect to have a look at was the navigation speed. It had to mimic the walking speed of a person but walking speed can vary a lot depending on the situation and depending on the person. Considering that the virtual environment to create is an art gallery, where most of the times people walk slowly stopping at each piece, the choice for the walking speed was a slow speed. It had to be slow but quick enough to avoid user getting bored.

One last feature required, and a very important one, is to prevent user from walking through walls. A wall avoidance system has to be implemented. The way of doing it will be by finding the collisions with the walls of the gallery and then do something to not let user continue his way.
• **Ability to look around**

As seen in the Design Requirements chapter, the navigation feature includes the ability to look around the gallery by moving the head. I found that incorporating this ability to the system would provide realism to the user experience.

In order to include this ability I need to track the head of the user so I need a way of adding some markers to it. This is a delicate subject; usually people aren’t very comfortable with something on their heads. Therefore I need to find something small which wouldn’t annoy the user. I finally decided that the best choice would be to add the markers to the stereo glasses. User will have to wear them anyway in order to visualise the environment properly.

Thinking about how people look around in a museum or gallery I got to the conclusion that they basically look around in the horizontal axis (right - left) and sometimes in the vertical axis (up - down). Taking into account that all the photographs in the gallery will be at an adequate height so user don’t have to look up or down to look at them, adding the ability to look up or down did not seem quite useful. I decided to include it though. I thought it was a good idea to give the user the freedom of looking everywhere he wants to, even if there is nothing interesting to look at outside the predetermined field of vision in the vertical axis.

Thus, I need information about how much the glasses rotate in order to apply this rotation to the view in the virtual environment. The rotation will be right-left or up-down so I need to get the rotation of the head around z-axis (yaw) and x-axis (roll).

After going through the documentation of the cameras and their software, I learnt that in order to get these rotation angles I had two options:

- 6 DOF data: getting 6 DOF data from the tracking system provides information about the position (x, y, z) and orientation (rotation of x-axis, y-axis and z-axis).

- 3D data: using the 3D tracking provides information about the position (x, y, z). In order to get the rotation angles I would have to use geometry principles to calculate them.

I went for the first option and used 6 DOF tracking. Even if using this kind of tracking will require requesting more data than the needed, it was a simpler option. The 3D option would imply a lot of calculation to get the rotation angles. In fact, the tracking for both is done the same way by the cameras; it is their software the one which calculates the rotation angles and provides 6 DOF data.

There are some special requirements for the 6 DOF tracking. First the creation of a rigid body is needed. This body has to be formed by at least 4 markers, and one of them has to be in a different vertical level. Knowing this, the design of the markers on the glasses was: three markers on the top part of them and the fourth one elevated somehow as a kind of antenna.
5.2 IMPLEMENTATION

5.2.1 Modelling of the gallery

For the modelling of the gallery I used 3DS Max modelling package.

The first version of it was a very simple gallery with just plain walls, floor and ceiling. It was created using just boxes and planes, and boolean subtraction to create the door spaces. In this way the number of polygons forming the gallery was the smallest possible.

Due to some complications with the creation of bounding boxes during the collision detection process that will be described a bit later, I had to remodel the gallery in a different way using more boxes. At this point, I also decided to create independent walls for each room bearing in mind that maybe each room’s walls could have a different texture applied to them or different colours. This way the number of boxes used, and thus, the number of polygons was higher but still a low number.

Figure 20: Gallery model

The model was exported as .3ds so it could be loaded in the system.
5.2.2 Navigation System Implementation

As mentioned above, the navigation system consists of two main aspects: ability to walk around and ability to look around. In order to improve navigation a wall avoidance feature has been also implemented.

- Ability to walk around

As explained in the Tools Chapter, the way of getting input from the dance mat is by using a piece of software (Pinnacle Game Profiler) which maps this input into keyboard input. This little program has to be running at the same time as my application.

First step was to create a profile for the dance mat and edit its configuration assigning a keyboard key to each of its buttons (see Appendix 1). I decided to use the left part of the keyboard and the assignation was at follows:

![Figure 21: Dance mat mapped into keyboard](image)

Once I had the dance mat input mapped into keyboard input by the software, I needed to find the way of programming the navigation. The way of getting input from a keyboard in an OSG program is by creating an event handler and adding it to the viewer.

The class I used for it (CKeyboardHandler - see Appendix 3) derives from the class GUIEvent.Handler, part of the OSG library. Inside this class the actions for each key which is pressed are defined. This class code was taken from the OSG website [] and adapted to my necessities for the navigation.
Once I had a way of getting input from the keyboard into my program, I needed to know how to translate that input into user movement. The approach for creating user movement in the virtual gallery is to move the camera according to the dance mat input. To do that, the view matrix has to be updated every time a key is pressed. The view matrix is multiplied by a translation matrix and by a rotation matrix whose values are updated according to dance mat/keyboard input. It is inside the viewer loop so it is updated every frame even if there is no input from user.

The movement applied to the camera for the navigation is the translation along the x-y plane (the height at which the camera is located remains fixed) and the rotation around the z-axis (vertical axis).

The way of programming this can seem so easy at first, doing it by just adding or subtracting units in the x and y coordinates of the translation matrix, and increasing the rotation angle for the rotation matrix. However, if the user wants to move forward after rotating, this “forward movement” direction is not the same as it was before the rotation was applied. In order to solve this I used trigonometry functions to calculate how much on each axis (x and y) I needed to translate the camera taking into consideration the rotation angle (see code).

This is a very optimal solution and it works really well with a minimum amount of code required. Multiplication of transformation matrices however, is always a tricky task and it has to be done with extreme care. Any little mistake can introduce really weird results.

- **Collision Detection**

The collision detection programming was a bit complicated. OSG provides many classes and functions but the documentation about them is not good or it does not even exist, so trying to figure out how to deal with them is a bit of a nightmare sometimes.

The way it was implemented is by creating a bounding box for each of the walls of the gallery, and a bounding box for the camera. In the viewer loop I check if the camera bounding box collides with any of the walls bounding boxes; and if this happens, a function moves the user position some units back in the opposite direction he was coming from. This is the way to prevent the user from walking through walls.

The first task was to create the bounding boxes for each of the walls of the gallery. I transformed the node containing the 3D model of the gallery into a group node so each of its elements could be accessed. Once I figured out how to do it, I found that the creation of the bounding boxes was not working fine because of the way my gallery was modelled.
The way I had modelled the gallery was using a box for each wall and subtracting a piece from it in order to create the door space. This way, when the bounding boxes were created they included the door spaces so it was no possible to walk through them. This made me having to remodel the gallery so that bounding boxes were created in the correct way.

Another problem I found was that the gallery model I created was at a random location so I needed to apply a translation to it in order to see it centred on the screen. Needless to say that this translation had to be applied to the bounding boxes as well, but at that time I didn’t see it that obvious. I though the bounding boxes were created in the same position the model was (documentation did not help much either). I finally managed to get the corresponding bounding boxes at their right position, but learnt the lesson: it is very important to locate a 3d model centred at its local coordinates system origin point.

The second task was to create the bounding box for the camera (user). The position of this bounding box has to be updated with the camera’s location. In order to do that, I extracted the camera’s position vector from the viewer and use it for setting the position of the camera’s bounding box. One aspect to consider was the size of this bounding box, if set too big it could detect a collision even if user is not that close to the wall; and, if set too small, it could not detect that user is by a wall until he is very close to it. I had to find a medium value.

The function which is called every time the camera bounding box collides with any of the gallery walls, and prevents the user from walking through them, is named bounceBack() (see code at Appendix 3). It sets user position (camera position) some units back in the opposite direction he was coming from. To know which direction to send the user back to, I added some code to the keyboard handler so the last key pressed is stored in a variable. This variable is checked in the bounceBack() function and depending on its value a different value is added to the translation matrix coordinates.

- **Ability to look around**

The first step was to physically add the markers to the glasses. As seen in the design part, three markers go on the top part of the glasses and the fourth one is elevated as a kind of antenna. To create this “antenna” I glued a small pencil to the glasses and attach a marker at its tip. I painted it black so it looked better. (Keep in mind that this is just a prototype).
Next step in the implementation of the head tracking system was to create a 6 DOF body on the Qualisys Tracker Manager software. I created it recording the movement of my glasses around the space the cameras cover, and then naming each of the four markers and grouping them together as a body named a_glasses.

This rigid body was set up to have its own local coordinates and rotation has to be initialised. Under “Real time actions” “Calculate 6DOF” has to be activated (See Appendix 1)
One good idea is to calibrate the local coordinates of the glasses body for each of the users using the gallery. Everyone is different and wears the glasses in a different way so calibration is useful to set the local coordinates when user is looking straight to the screen. This way, the rotation angles are set to zero for user looking straight to the screen and will vary as he moves his head. Thus, when the application starts and user is looking to the centre of the screen he will see the environment from a normal point of view, with no changes produced by not desired rotations.

Next, I had to integrate the tracking data into the OSG program. (Code can be found in the Appendix 3.)

For the head tracking implementation I had to go into the RT_Client code provided by the cameras software to deal with the data coming from the real-time motion capture system.

I had to introduce some changes to it. The more significant ones are listed here:

- In order to ask the system to send the data I needed, 6 DOF data, I changed the command for the request of the parameters to "SendParameters 6dEuler".

- Next, I changed the part of the code in which the labels are read and copied so it worked for the 6 DOF bodies labels.

- Then, in order to get the data of each frame I change the correspondent command to: "SendCurrentFrame 6dEuler"

- And finally I modified the PrintData function so I could access the data I wanted from there.

Inside the PrintData function, the first thing to check is if the data received is 6 DOF data. Next I need to check the label of the body. There can also be more bodies defined in the system so I have to make sure I get the data from mine so I check if the label with the name is my glasses one: "a_glasses". Once this is checked I can take the data from there being sure that it is the data I am looking for. I used two of the rotation angles sent in the array, fR1 and fR2, corresponding to the rotation in x-axis (roll) and the rotation in z-axis (yaw).

These two rotation angles are stored in two global variables and applied to two transformation matrices which are multiplied by the rest of the transformation matrices affecting the view matrix (order of the multiplications is very important).

One important aspect to consider is that the rotation angles are sent in degrees and OSG works in radians. I transform them in the rotation matrix definition.

When programming this part of the head tracking I came across three different factors I had to take into consideration and found a solution for each of them (see PrintData code in RT_Client.cpp).
First. In this lab, the user is located in front of the screen where he sees the virtual gallery. This screen is a flat screen which just covers one plane, so the head angle rotation range is not very broad, nor in the horizontal axis neither in the vertical one. For example, if the user rotates his head ninety degrees to the right he will see nothing but the wall of the lab. Even if the user will not probably try to rotate the head to big angles (because this will make him loose the sight of the screen and thus, the gallery) I decided that it was better to limit this angle in order to avoid problems. The limit was set to +45 and -45 degrees in both axes.

Second. If the tracking is lost at some time, the data sent by the software are not numbers, are something else, and that applied to a transformation matrix which affects the view matrix makes the viewer go crazy. To avoid that and knowing that even if it is not very probable, the tracking can be lost due to different reasons, I introduced some extra code. In this extra code I check if the data received are numbers; if they are, the rotation angle is updated, if they are not the program uses the rotation angels previously received. By doing this, if the tracking is lost at some point the view will stay static not responding to the movement of the head instead of showing some crazy behaviour.

Third. During the process of implementation of the ability to look around the gallery I realised how much we actually move the head. The tracking system is very sensitive so it detects the very minimum movement the user head does. That, transmitted to the changing of the view on the screen was terribly annoying; the view was constantly changing and looked as if the gallery were trembling. For us, this is not a problem as our eyes don’t move in the same way our head does and after all our brain is the one who sees, not our eyes. But this is an objective system and I am tracking the head so I needed to find a solution. I decided to limit the way the view is affected by the movement of the head. The camera angle will not vary unless the change in head angle is bigger than a significant amount. The “significant amount” had to be carefully defined as a big number will make the camera not moving gently.

Another aspect to bear in mind is that, when the user moves the head to the right or to the left, this affects the view in the same way dance mat rotation does. However, it does not affect the angle in which the user moves. This seems a bit like a contradiction but I decided that head tracking should not affect navigation because usually, rotation of the head does not involve rotation of the body. In this lab, the user is located in front of the flat screen so most of the time the user will be looking straight to it. Besides, I believe that the user will be looking to the front when he wants to move forward so this should not be a problem.

I am aware that, at some points, the ability to look around could result a bit annoying. The users who are not used to the dance mat will often have to look down to locate each of the buttons, and that will make the view change all the time.
Looking around ability in the keyboard

I also implemented the ability to look around the gallery by using some keys of the keyboard. I thought it might be useful during the implementation process and it could also be useful for trying this application outside the lab without the equipment necessary for the tracking.

I decided to go for the right part of the keyboard this time using the following keys:

I: look up
M: look down
J: look left
L: look right

Figure 24: Gallery appearance
5.3 EVALUATION

For the evaluation part of this first prototype I used six people who tried the system and filled in a questionnaire after that. I did a bit of follow up interview after they had filled in the questionnaire to make sure I understand their feedback and they understood the questions properly. I also observed them while they were using the application and took some notes.

The questionnaire I used can be found in the Appendix 2. The analysis applied to the data gathered is a qualitative analysis.

I informed the people taking part in this evaluation process that, because of the nature of the visualization system they could suffer some cybersickness symptoms after the virtual experience, such as eye strain or dizziness. However I also told them that this was not very probable, as this is a semi-immersive virtual environment.

The data collected in the process is summarised next.

All of the subjects taking part in this evaluation process had previous experience with either video games, virtual worlds or 3D environments; so navigation in a 3D space was not new to them.

Only two of the participants had previous experience with the dance mat, the rest had never used it. However most of them found it easy to get used to it, only one participant, who had never used it before, found it a bit difficult.

When talking about the navigation system, all of the subjects agreed that it was intuitive. Five out of six found it quite intuitive while the other one found it very intuitive. The walking speed was considered OK by most of them (4/6), however it was considered slow but one subject and too slow but other. The rotation speed was found OK by five of the participants and slow by just one. One of the subjects got lost while navigating in the gallery, “all of the rooms look similar” the subject stated.

About the gallery size, all of them found it ok (meaning by gallery size the size in the y axis, size of doors spaces and walls in comparison to the user).

All of the participants found the ability to look around useful and most of them mentioned it when asked about the aspects of the gallery they found particularly good. One of them commented: “it’s very cool”; and other one said that “head tracking helps a lot”. About its accuracy, four out of six thought it was very accurate while the other two thought it was a bit accurate. “Make it smoother” one participant said.

Most of them, five, enjoyed the experience, the other one reported that “navigation was fun but environment too boring”.

The aspects the participants found good about this prototype are the ability to look around and the way of navigation around the gallery.

On the other side, the problems the subjects encountered are: difficulty to determine the distance to the walls, flickering when looking around, having to look down to the dance mat to locate its buttons, balance issues when stepping on the dance mat and height sensation issues. Apart from that, one of the participants walked through a wall, and another one presented eyes discomfort, “after wearing the glasses I need time to recover and to be able to focus my sight in short distance objects again” said.

The main suggestions the participants did for improvements are: higher walking/rotation speed, adjustable walking/rotation speed, to add lights, colours, pictures and furniture, to add ambient sound, to place camera at higher point and to make the origin of the gallery a bit lower.

After collecting this data from the subjects’ questionnaires, I went through it and reached the following conclusions:

- The users found the environment boring and empty, which is totally understandable as in this first prototype the gallery was nothing but empty walls without any lighting or texturing. Due to this the depth perception was difficult at some points.

- Participants found the navigations features good, specially the ability to look around the gallery by moving the head. However the camera movement as user looks around needs to be smoother and the walking speed needs to be a bit higher. The dance mat interface is easy to use even though it might cause balance problems to some people.

- One subject walked through a wall, which means there is a bug in the collision detection implementation.

- Some participants got a bad perception of the height at which the user is located in the gallery. These participants where the shortest ones, which shows that the system needs some kind of height calibration.

- When looking around the image flickers a bit.

In summary, I can say that, overall the subjects liked the navigation system, which was the aspect to test in this evaluation, and disliked the way the gallery looked. The ability to look around the gallery by tracking the head is a very good feature which improves a lot the user experience. The rest of the navigation system was considered intuitive and easy, but still needs refinement. The gallery needs lighting, texturing and furniture.

This feedback will be used in the second prototype in order to improve the system.
CHAPTER 6. SECOND PROTOTYPE

As explained in the introduction chapter, this second prototype will improve the navigation features and gallery appearance. To do this I will consider the conclusions reached by the first prototype evaluation process. Moreover, the pictures interaction feature will be introduced, and the gallery appearance improved.

Figure 25: Gallery view
6.1 DESIGN

6.1.1 Changes in first prototype features

Taking into account the results of the first prototype evaluation and my own experience while implementing this first prototype, I designed some changes in the navigation system.

- Walking speed needs to be increased a little bit.
- Head tracking needs to be adjusted.
- Collision detection: bug in its code has to be fixed.
- Anti-aliasing function has to be introduced to avoid the image flickering.
- Height calibration needs to be introduced. The idea is to use two of the spare buttons from the dance mat in order to let the user move the camera up or down.
- Gallery model needs to be centred at its local coordinates’ origin to avoid adding translation matrices in OSG and make things easier when using bounding boxes.

6.1.2 Modelling of frames + box + photos

The frames where the photographs will be placed have a very simple design. I decided it to be like this keeping in mind that the important thing is the photographs and not the frames hosting them. A very sophisticated design will get the attention and prevent user from focus on the photographs. The frames are designed for big format landscape orientation photographs.

Another reason for the simplicity of the frames is the limitation in number of polygons for the 3D model. The number of polygons should be small not to slow down the application.

Taking into consideration the distribution of the walls around the gallery I decided to have twenty one frames around it. All of the rooms have from four to six frames depending on their size and walls, and the corridor is left with no frames.

The box also has a very simple design. It is meant to be a very simple cardboard box where the user carries his photographs. I was not trying to design a beautiful box but to design a box which matches the standard image of a box so it could be easily identified by the user. I decided to place the word photos on its side.

The 3D model for each photo is just a plane with the photograph applied as a texture.
6.1.3 Design of the interaction with the photographs

During the design process for this interactive feature I came up with so many interesting ideas about how it could be done. However, I had to be realistic and consider that this is a short time project. That is why I decided to design a simple, but nice, photo interaction system. Some of the ideas I came up with will be added in the “Recommendations for future work” section.

As seen in the requirements chapter, the way of interacting with the photographs should be intuitive and easy to learn, and there is no need to mimic real world actions. The aim is to place the pictures in the empty frames which are hung on the gallery walls. The pictures are originally with the user.

In order to do this I need to track the hand of the user. The idea is to track the pointer finger.

The general design is the following: the user has a virtual box next to him with the photos in it. When he wants to place a photo in a frame, he goes close to that frame, open the box and select the desired photo.

The box is located in front of the camera and moves as the user do, being always accessible. The box position will even have to change when the user looks around, this way the box is always seen on the same place on the screen.

The idea is to have the box always visible at one of the corners of the screen and, at any time the user wants to check the photos inside it, he can “touch” the box and then the photos will come out. The photos will come out from the box with an animation moving from the box to a position in front of the user; their size will become bigger so user can see them properly and select which one he wants on the frame. To select a photo, the user has to “touch” it with his finger. If the user wants to put the pictures back in the box he will just have to “touch” it again and the photos will go back to the photo with animation.

When designing the way of selecting a frame in which to place the photo I took into consideration that the user hand is already involved in taking the photos out of the box and selecting a photo, so I decided not to give another task to the hand. I decided to do it by using user’s position. The user has to be in front of a frame and relatively close to it in order to select it. This way, when the photos are out of the box and the user selects one of them, this photo will go to the closest frame in front of the user.

One key aspect about this interactive feature is the design of how to select or “touch” objects in the virtual gallery. As seen in the literature review chapter, in a virtual environment it is very difficult to know when something has been selected as there is no physical feedback, so some kind of visual feedback has to be introduced.
I decided to introduce a pointer which follows the users hand movements in order to make the pointing easier. This way, the user always knows where his hand is pointing at. I did not introduce however, any special way of visual feedback for the selected item, the visual feedback for it will be the execution of the selected actions (for example, if the box is selected the photos will go out of it).

Moreover, I needed to design the way of selecting using the hand. The way I designed it is the following. The user moves the pointer on screen as he moves his hand, if he wants to select something he will have to do a pointing gesture moving his hand to the front. When the user's hand is in the most front area, that is consider as a click in a normal computer application. However the user will have to remember to put his hand back to normal pointing position, if not he would be "clicking" all the time, selecting thing he doesn't want to select.

6.1.4 Gallery appearance design

When designing the gallery appearance, I took into consideration some of the evaluation subjects’ suggestions for improvements. Some of them mentioned that furniture would make it nicer. However, under my point of view, it does not matter if it looks empty because after all, a gallery is an empty place where the important features are hung on the walls. What I did consider, which matches my original idea for the gallery appearance, was the suggestion of adding lights and shadows and colours. Careful lighting and appropriate texturing would really make the gallery look good.

The goal is to make the gallery a quiet and comfortable place even if it has nothing but frames on the walls. Lighting plays a very important role in the creation of different atmospheres, it changes the way we perceive things. As seen in the requirements the lighting has to be rich enough to let the user see the pictures properly, but soft enough to create a nice atmosphere. The designed lighting should be a warm lighting with soft lights.

Another means of improving the look of the gallery is texturing, adding textures to the environment helps to create a more realistic appearance. Nice textures applied to the walls and the floor would do it for this environment. The designed texturing is a wooden texture for the floor and different soft colours textures for the walls.

In summary, the way I designed the gallery appearance to be was to have a warm lighting, a wooden floor and soft colours walls.
6.2 IMPLEMENTATION

6.2.1 Implementation of changes to first prototype

Before implementing new features in this second prototype I had to implement the changes designed for the improvement of the first prototype. They are listed here:

- **Walking speed.** I increased the walking speed by just modifying the amount of translation applied to the camera every time a dance mat key is pressed. (see CKeyboardHandler class - Appendix 4)

- **Head tracking.** I adjusted the head tracking system by modifying the “significant amount” the change in head angle has to exceed (explained in 5.2.2). I made it slightly bigger so the change in the view is smoother; it is not affected by very small changes (see RT_Client.cpp – Appendix 4).

- **Collision detection.** I found the bug in the collision detection implementation and fixed it. It was in the bounceBack function and it was just a wrong symbol.

- **Anti-aliasing.** I introduced a line of code in the main to change the display settings to avoid the image flickering. (Main.cpp – Appendix 4)

- **Height calibration.** I decided that the best way of calibrating the height at which camera (user) is located would be by adding an image in front of the camera when the application starts. This image has a cross on it, and it is placed at a specific height. If the user sees that cross in front of him without rotating his head at all, that will make him being at the adequate height to enjoy the gallery. The user can move the camera up and down with the two back diagonal arrows of the dance mat to place himself at the adequate height. To start the gallery experience once this is done, “Start” has to be pressed on the dance mat. (Main.cpp – Appendix 4)

![Figure 26: Height calibration](image)
o **Gallery model.** The original idea was to centre the gallery model at its local coordinates. What I did however was to translate it a bit from this origin coordinates to my convenience so the gallery was in the place it should when I load it into OSG. Like this I avoid having to move the camera’s original position.

### 6.2.2 Modelling of frames + box + photos

I created two different frame models using 3ds Max software.

The two models are quite simple. The way of creating them was by first drawing the profile of the shape and then applying it to a rectangle using a modifier to build the frame and give volume to it. The two designs can be seen in the images below (the colour has been added so the shape can be better appreciated):

![Figure 27: Frame 1](image1.png) ![Figure 28: Frame 2](image2.png)

I went for the simplest one, green.

The photo box was modelled by just using boxes primitives. The text “photos” was set on one of its sides. A cardboard texture was applied to it afterwards in OSG.

![Figure 29: Photo box model](image3.png)
As previously mentioned the object for the photographs is just a plane which size is proportional to 3/4, the traditional photo size for landscape pictures.

6.2.3 Interaction with the photographs

The implementation of this simple photo interaction system required a lot of hard work. The most important aspects are summarised next. Code can be found in Appendix 4, it is properly commented and separated in functions.

The way the user interacts with the gallery is by using his hand. Thus, the user’s hand is tracked. In this case I only needed information about the hand’s position so I used just one marker and got 3D data about it (x, y and z position). I had these two options shown next. I finally went for the yellow finger as it was easier to wear.

![Figure 30: Finger marker](image1.png)  ![Figure 31: Finger marker](image2.png)

I used one unlabelled marker so I introduced changes in the RT_Client code in order to get information from both my 6 DOF (glasses) and from the unlabelled marker (finger). These changes are programmed in a way that only one unlabelled marker is considered in order to prevent the accidental consideration of other markers.

The first things to implement were the translation of the hand movement to the pointer and the way of selecting something. I soon realised I needed a calibration system and some calculation to properly transfer the movement of the hand to the movement of the point in around the screen. I decided to consider the marker position as “in selecting mode” if the marker was in a certain y-position (further from the user). I also realised that the frequency at which the system receives information from the marker position is very high so I changed the conditions for considering the marker in “selecting mode”. It would be considered as
selecting only if it was in that position for a certain time (see handTrackingCalibration() and getPointerInfo() at Main.cpp).

The calibration is done by taking some measurements before the application starts. Like this a box where the user is going to be moving his hand is defined. These measurements are taking into account for translating the hand movement to the point on the screen.

The box with the photographs together with the pointer, move as the user does so they are all the time in front of the user. They are all under the same transformation matrix which is updated with the user position and orientation, and head rotations.

If the box is selected, the pictures come out from the box with an animation. In order to do that I first checked if the pointer is in the box position, and then check if it is in the “selecting mode”. If so, the animation starts and the photos go out.

One thing to mention is that I decided to stop the head tracking while the photos are out from the box as it was quite disturbing.

Considering that there are 21 photos, one for each frame, only half of them are shown at first while they come out from the box. In order to see the other ones, the user has to select the option “More Photos”, and once on the other side, he has to select “Back” to go back to its original position. In order to put the pictures back on the box, the user just needs to select the box. The pictures will then go back to it with an animation.

Once the photos are out, if the user is in front of a frame and relatively close to it, he can choose a picture by just selecting it. In order to know which frame from the available frames (which were hung on the gallery walls by the function hangFrames() see code) to sent the photo to, the function findClosestFrame is called. This function returns the number of the
frame. Then the picture is removed from its position on the scene graph and added in a new transformation matrix with the frame position and orientation.

6.2.4 Gallery appearance

Due to a lack of time I was not able to improve the gallery appearance the way I had designed it. However, I did include some of the designed features.

I applied a wooden texture to the floor. This was done inside the startupScene function (see Main.cpp – Appendix 4).

I implemented a lighting system which is formed by five lights, one light on each room (See createLights function – Appendix 4).

These two aspects do not give the gallery a realistic appearance but make it look better.

➢ Second Prototype Gallery Screen shots

The next images show how the gallery looks. The first one shows the gallery with the empty frames. The second one shows the process of selecting a picture, and the last one shows a picture placed in a frame.
Figure 34: Process of selecting a photo
Figure 35: Picture placed on a frame
6.3 EVALUATION

The evaluation of this second prototype was done in the same way as the first prototype evaluation. I had some people to try the virtual gallery, asked them to fill in a questionnaire and talked to them about it afterwards. Observation was also a part of it. This time the number of subjects was seven.

The second prototype questionnaire was similar to the questionnaire used in the first prototype evaluation; it had some extra questions about the photo interaction part. It can be found in the Appendix 2.

I explained to them that they could use their own photographs in the virtual gallery. However, all of them used the sample ones.

Participants in this second evaluation process were also informed about the possible cybersickness symptoms they might suffer.

The results obtained in this second evaluation questionnaires are shown next.

As it happened in the first evaluation process most of the participants had previous experience with navigation in 3D environments but, this time, most of them also, had use a dance mat before. Probably in the first prototype evaluation process, most of the participants were the same ones.

Even though two of the participants had some balance issues with the dance mat, all of them found it easy to get used to it. One of them stated: “Initially I found it tricky to balance on the dance mat while looking straight ahead at the screen. However, I had never used a dance mat before. This quickly became easier”.

About the navigation system, all of the subjects agreed that it was to some extent intuitive. Four out of seven found it quite intuitive, two of them found it very intuitive and one of them said it was a bit intuitive. The walking speed was considered OK by all of the participants. The rotation speed was found OK by five of the subjects, slow by one and quick by other one. Only one of the participants got lost while navigating around the gallery. About the gallery size, all of them found it ok.

All of the participants found the ability to look around useful and two of them mentioned it as a good aspect of the application. About its accuracy, all of them found it very accurate.

When asking them about the photo interaction system, all of them agreed that it was intuitive. Four out of seven found it very intuitive and the rest quite intuitive. In relation to getting used to use the hand as a pointer, most of them found it easy or very easy. One of the participants however, found it a bit difficult. All of them thought it was an accurate system, five participants said it was very accurate and the other one said it was a bit accurate. It was described in the good aspects section as a “cool interaction system” and an “easy interaction by just pointing”.

55
This time, all of the participants enjoyed the experience. Apart from highlighting as a good feature the ability to look around the gallery and the photo interaction system, some of the other aspects mentioned were: the possibility of using your own pictures and create your own gallery, the floor and colours of the gallery and the photo animations. One of the participants liked to “have a look at photos placed in other rooms”.

Regarding the problems encountered while using the virtual gallery, most of the subjects stated more or less the same ones: difficulty in selecting the box, involuntary selection of photos, accidental selection of a picture after selecting “More photos” or “Back”. Two subjects mentioned balance issues with the dance mat and another one mentioned “unintended movement by stepping on a button when I didn’t want to”.

Finally, the suggestions the participants made are summarised next. One of the most repeated suggestions was to include the ability to replace pictures from the frames. Another one was to include more photographs than frames available. One participant suggested that different albums would come out from the box and another one said that he would introduce the option of hiding the box and the pointer.

In addition, one of the subjects suggested the idea of using 3D photos and other one suggested including the ability of adding more exhibition space and frames and a floor plan on a corner of the screen. Two participants suggested adding objects in the rooms and another two adding music to the virtual gallery.

Finally, one participant commented that it would be nice to exit the application by walking through the main gallery door. Another one said about the virtual gallery: “this was a very engaging way to view pictures (…); this could be a fun way to present photos to friends and family”.

In addition to the data collected in the evaluation questionnaires I took some notes while observing them using the gallery.

About the tracking I noticed that, at some points the tracking of the hand was lost, and regarding the head tracking, there were some problems if the user had a fringe.

About selecting a frame, I realised that the participants tend to forget what I explained them about its selection. They tried to place a picture on a frame from a not valid orientation. Some users tend to classify the pictures by topic for each room.

About the selection of photos, most of the users selected photos or the box accidentally. Moreover, most of the users selected by mistake the picture in the second round of pictures which is in the position of “More Photos”. The same happened with “Back”. I also realised that it was annoying having to go back to the first group of photos in order to put them back in the box. As well, I realised that users learnt from their “mistakes” and tried to do things in a
different way after that, for example avoid pointing to a picture not to select it after having selected one accidentally. Some of the users also had problem when selecting the box.

I observed balance problems stepping on the dance mat in two of the participants. One of them had big feet and had also problems stepping by mistake on different keys. The other one had a lot of balance issues at the beginning until she got used to the dance mat. While trying to keep balanced she put the photos out of the box all the time.

One aspect that I observed was that people don’t see their own head shadow on the screen which was one of my concerns about using a front projection screen. It was not noticeable as they were focus on other parts of the screen.

After going through all the data gathered I came to the following conclusions:

- Participants liked the navigations features, specially the ability to look around the gallery. Walking speed is now ok. The dance mat interface is easy to use but it causes balance problems to some people.

- The height perception problem is solved after including calibration for it.

- Participants really enjoyed the photo interaction system and the idea of using their own photographs. They liked the pointing finger interface.

- The selection system needs to be improved as sometimes pictures are selected by mistake.

- A good hand calibration process is essential for the photo interaction system. The “deep” point has to be properly measured to avoid selecting problems.

- Gallery appearance is much better now. Participants highlighted the floor it doesn’t look real.

In summary, I can say that in this second prototype the navigation features were improved and a nice photo interaction system implemented. Participants enjoyed using the gallery even if they encountered some selecting problems.
CHAPTER 7. CONCLUSIONS AND FURTHER WORK

In this project I have created a virtual photo art gallery in which input for navigation and interaction is obtained from a dance mat and a real-time motion capture system. The evaluation showed that this innovative way of navigating and interacting is intuitive and easy to get used to, and it makes the virtual gallery a fun and enjoyable application.

Overall it has been a not easy project in which I have invested so much time. OSG information or documentation is either not good or it does not exist so developing an application with it is so time consuming, every little thing takes hours and hours. The experience is rewarding though when seeing the results. I actually feel it is a shame that by the time I got confident with OSG the project was on its last part.

The summary of my achievements is the following:

- I have managed to create a virtual art gallery integrating input from a dance mat and IR cameras.
- I have implemented a nice navigation system with the ability to walk around and look around.
- I have implemented an effective collision detection system which prevents user from walking through walls.
- I have implemented a fun interaction system to place photographs around the gallery.

In comparison with the objectives described in the Chapter 1, there is one objective which I have not totally achieved: to create a realistic appearance for the gallery.

Considering my limited previous knowledge of C++, the poor OSG documentation and the unexpected events I think I have finally managed to do quite well.

During this project I have learnt a lot, not only about the use of OSG and C++ but about the design of virtual environments and all of the aspects related to it.
The first recommendation for further work would be to improve the photo interaction system. Apart from the improvements needs shown by the evaluation process, I think it would be nice to include features such as gesture recognition to make the interaction more intuitive and fun. It would also be a good idea to track both hands and improve the interface by adding with more animations and visual effects.

A second recommendation is to introduce sound in the virtual gallery. Some background music would be nice and also some background noises. This could improve the experience by making the user feel more present in the virtual environment.

Finally, one last recommendation is to include intelligent virtual agents inside the gallery with whom the user could interact. These could be other visitors or gallery assistants. This would make it more fun, as galleries apart from showing art can also be a social place.
REFERENCES


BIBLIOGRAPHY


- OSG Quick Start Guide

- OSG Reference Guide

- OpenSceneGraph website. http://www.openscenegraph.org/

APPENDIX 1. PINNACLE GAME PROFILER AND QUALISYS TRACK MANAGER
PINNACLE GAME PROFILER

The software used to map the input from the dance mat into keyboard input is a very simple to use program.

First step is to create a new profile for the dance mat and edit its configuration.
Next I edit that profile and create a new configuration for it, defining which key is attached to which dance mat button.
QUALISYS TRACK MANAGER

First of all the 6DOF has to be activated.

Creation of the 6 DOF body for the glasses. It has four markers.
Local coordinates adjustment:
APPENDIX 2. EVALUATION QUESTIONNAIRES
Q1. Did you have previous experience with video games, virtual worlds or 3D environments?
   - No
   - Yes, some.
   - Yes, a lot.

Q2. Had you used a dance mat before?
   - No
   - Yes

Q3. How intuitive did you find the navigation system?
   - Zero intuitive
   - A bit intuitive
   - Quite intuitive
   - Very intuitive

Q4. How easy did you find to get used to the dance mat?
   - Very difficult
   - A bit difficult
   - Easy
   - Very easy

Q5. How did you find the size of the gallery?
   - Too Small
   - Quite small
   - Ok
   - Quite big
   - Too big

Q6. How did you find the walking speed?
   - Too slow
   - Slow
   - Ok
   - Quick
   - Too quick

Q7. How did you find the rotation speed?
   - Too slow
   - Slow
   - Ok
   - Quick
   - Quite quick
Q8. Did you get lost at any point?
- No
- Yes

Q9. How did you find the ability to look around the gallery?
- Useful
- Useless
- Annoying
- Distracting

Q10. How accurate did you find the head tracking system?
- Not accurate
- A bit accurate
- Very accurate

Q11. Overall, did you enjoy the experience?
- No
- Yes

Q12. Any aspect you found particularly good?

Q13. Any problems you encountered?

Q14. Any suggestions for improvements?

Q15. Any other comments?

Appendix 2. Evaluation questionnaires.
SECOND PROTOTYPE EVALUATION

Q1. Did you have previous experience with video games, virtual worlds or 3D environments?
   - No
   - Yes, some.
   - Yes, a lot.

Q2. Had you used a dance mat before?
   - No
   - Yes

Q3. How intuitive did you find the navigation system?
   - Zero intuitive
   - A bit intuitive
   - Quite intuitive
   - Very intuitive

Q4. How easy did you find to get used to the dance mat?
   - Very difficult
   - A bit difficult
   - Easy
   - Very easy

Q5. How did you find the walking speed?
   - Too slow
   - Slow
   - Ok
   - Quick
   - Too quick

Q6. How did you find the rotation speed?
   - Too slow
   - Slow
   - Ok
   - Quick
   - Quite quick

Q7. Did you get lost at any point?
   - No
   - Yes

Q8. How did you find the ability to look around the gallery?
   - Useful
   - Useless
   - Annoying
   - Distracting

Q9. How accurate did you find the head tracking system (looking around)UNKEN?
   - Not accurate
   - A bit accurate
   - Very accurate
Q10. How intuitive did you find the interaction system (pointing with your hand)?
- Zero intuitive
- A bit intuitive
- Quite intuitive
- Very intuitive

Q11. How easy did you find to get used to use your hand as a pointer?
- Very difficult
- A bit difficult
- Easy
- Very easy

Q12. How accurate did you find the interaction system (pointing with your hand)?
- Not accurate
- A bit accurate
- Very accurate

Q13. How did you find the size of the gallery (in comparison with you)?
- Too Small
- Quite small
- Ok
- Quite big
- Too big

Q14. Overall, did you enjoy the experience?
- No
- Yes

Q15. Any aspect you found particularly good?
__________________________________________________________
__________________________________________________________
__________________________________________________________

Q16. Any problems you encountered?
__________________________________________________________
__________________________________________________________
__________________________________________________________

Q17. Any suggestions for improvements?
__________________________________________________________
__________________________________________________________
__________________________________________________________

Q18. Any other comments?
__________________________________________________________
__________________________________________________________
__________________________________________________________
• List of files included in the project:

```plaintext
Solution 'MyGallery' (1 project)
  MyGallery
    Header Files
      - CKeyboardHandler.h
      - globalVariables.h
      - main_header.h
      - NBC_Markup.h
      - RT_Client.h
      - StdString.h
    Resource Files
    Source Files
      - CKeyboardHandler.cpp
      - globalVariables.cpp
      - Main.cpp
      - NBC_Markup.cpp
      - RT_Client.cpp
```

**NOTE:** NBC_Markup.h, StdString.h and NBC_Markup.cpp have been omitted here as they are not relevant to this project.
// CKeyboardHandler.h

#ifndef CKEYBOARDHANDLER_H
#define CKEYBOARDHANDLER_H

class CKeyboardHandler : public osgGA::GUIEventAdapter
{
    public:
        virtual bool handle( const osgGA::GUIEventAdapter& ea, osgGA::GUIActionAdapter& aa, osg::Object* pObject, osg::NodeVisitor* pNodeVisitor);

    // Function with handles the input from the keyboard
    // for each key it does an action
};
#endif
// globalVariables.h

#ifndef GLOBALVARIABLES_H
#define GLOBALVARIABLES_H

// global variables

extern const double PI;
extern double myX;
extern double myY;
extern double myZ;
extern double myR;
extern double myHR;
extern double myHR2;
extern char lastKeyPressed;

#endif
// main_header.h

ifndef MAIN_HEADER_H
#define MAIN_HEADER_H

#include <iostream>
#include <osg/Group>
#include <osg/Node>
#include <osgDB/ReadFile>
#include <osgViewer/Viewer>
#include <osgGA/GUIEventHandler>
#include <osg/PositionAttitudeTransform>
#include <osg/BoundingBox>
#include <osg/ComputeBoundsVisitor>
#include <osg/ref_ptr>
#include <math.h>
#include <windows.h>

using namespace std;

#endif
1 // RT_Client.h
2
3 #include "StdString.h"
4 #include <vector>
5 #include <iostream>
6
7 #include <typeinfo>
8 #include <cmath>
9
10 #pragma pack ( push, 4 )
11
typedef struct _SHeader
12 {
13     long nSize;
14     long nType;
15 } SHeader;
16
typedef union _UHeader
17 {
18     SHeader s;
19     char c[sizeof(SHeader)];
20 } UHeader;
21
#define BUFFER_SIZE 500
22
typedef union _USendBuffer
23 {
24     SHeader s;
25     char c[BUFFER_SIZE+sizeof(SHeader)];
26 } USendBuffer;
27
typedef struct _S3dNoLabels
28 {
29     double fX;
30     double fY;
31     double fZ;
32     long nId;
33     long nPadding;
34 } S3dNoLabels;
35
typedef struct _S3d
36 {
37     double fX;
38     double fY;
39     double fZ;
40 } S3d;
41
typedef struct _SForce
42 {
43     S3d sForce;
44     S3d sMoment;
45     S3d sPoint;
46 } SForce;
47
typedef struct _S6d
48 {
49     double fX;
50     double fY;
51     double fZ;
52     double fR[9];
53 } S6d;
54
typedef struct _S6dEuler
55 {
56     double fX;
57     double fY;
58     double fZ;
59     double fR1;
60     double fR2;
61     double fR3;
62 } S6dEuler;
typedef struct _SPacketHeader
{
    __int64 nTimeStamp;
    long nFrameNumber;
    long nComponentCount;
} SPacketHeader;

typedef struct _S3dComponent
{
    long nSize;
    long nType;
    long nCount;
    long nPadding;
    S3d as3dPoints[BUFFER_SIZE];
} S3dComponent;

typedef struct _S3dNoLabelsComponent
{
    long nSize;
    long nType;
    long nCount;
    long nPadding;
    S3dNoLabels as3dPoints[BUFFER_SIZE];
} S3dNoLabelsComponent;

typedef struct _SForceComponent
{
    long nSize;
    long nType;
    long nCount;
    long nPadding;
    SForce asForceSamples[BUFFER_SIZE];
} SForceComponent;

typedef struct _SAnalogComponent
{
    long nSize;
    long nType;
    long nCount;
    long nPadding;
    double aSamples[BUFFER_SIZE];
} SAnalogComponent;

typedef struct _S6dComponent
{
    long nSize;
    long nType;
    long nCount;
    long nPadding;
    S6d aSamples[BUFFER_SIZE];
} S6dComponent;

typedef struct _S6dEulerComponent
{
    long nSize;
    long nType;
    long nCount;
    long nPadding;
    S6dEuler aSamples[BUFFER_SIZE];
} S6dEulerComponent;

enum EComponentTypes
{
    cn3d = 1,
    cn3dNoLabels,
    cnAnalog,
    cnForce,
    cn6d,
    cn6dEuler
};

#pragma pack ( pop )

void PrintError(char *t);
151 bool RecvMessage(SOCKET hSocket, SHeader & rsHeader, CStdString & rtData);
152 bool SendCommand(SOCKET hSocket, CStdString tCommand);
153 SOCKET Connect(CStdString tServer, int nPort);
154 void Disconnect(SOCKET hSocket);
155 void PrintData(SHeader sHeader, CStdString tData, std::vector<CStdString> atLabels);
156 void Example2(CStdString tServer, int nPort);
157 void trackingStart();
158
// CKeyboardHandler.cpp

#include "main_header.h"
#include "globalVariables.h"
#include "CKeyboardHandler.h"

bool CKeyboardHandler::handle(const osgGA::GUIEventAdapter &ea, osgGA::GUIActionAdapter &aa, osg::Object *pObject, osg::NodeVisitor *pNodeVisitor)
{
    osgViewer::Viewer* pViewer = dynamic_cast<osgViewer::Viewer*>(&aa);
    if ( !pViewer )
    {
        return false;
    }
    if ( ea.getEventType()==osgGA::GUIEventAdapter::KEYDOWN ) // CHANGE to switch
    {
        switch (ea.getKey())
        {
        case 'e': // move laterally to the right - E
            cout << "e pressed: move to the right " << endl;
            myX += 2*cos(myR);
            myY += 2*sin(myR);
            lastKeyPressed = 'e';
            return true; // return true, event handled
            break;
        case 'q': // move laterally to the left - Q
            cout << "q pressed: move to the left " << endl;
            myX -= 2*cos(myR);
            myY -= 2*sin(myR);
            lastKeyPressed = 'q';
            return true;
            break;
        case 'w': // move forward - W
            cout << "w pressed: move forward " << endl;
            myX += 2*sin(myR);
            myY += 2*cos(myR);
            lastKeyPressed = 'w';
            return true;
            break;
        case 'x': // move backwards - X
            cout << "x pressed: move backwards " << endl;
            myX -= 2*sin(myR);
            myY -= 2*cos(myR);
            lastKeyPressed = 'x';
            return true;
            break;
        // rotate left - A
        case 'a':
            cout << "a pressed: rotate left " << endl;
            myR = -0.02;
            return true;
            break;
        // rotate right - D
        case 'd':
            cout << "d pressed: rotate right " << endl;
            myR += 0.02;
            return true;
            break;
        // rotate head up - I
        case 'i':
            cout << "i pressed: look up " << endl;
            myHR += 0.5;
            return true;
            break;
        // rotate head down- M
        }
case 'm':
    cout << "m pressed: look down " << endl;
    myHR += 0.5;
    return true;
    break;

// rotate head up - J
case 'j':
    cout << "j pressed: look left " << endl;
    myHR2 -= 0.5;
    return true;
    break;

// rotate head down- L
case 'l':
    cout << "l pressed: look right " << endl;
    myHR2 += 0.5;
    return true;
    break;

return false;
}
1 // globalVariables.cpp
2
3 #include "globalVariables.h"
4
5 extern const double PI=3.141592;
6
7 double myX=0;
8 double myY=0;
9 double myZ=0;
10 double myR=0;
11 double myHR=0;
12 double myHR2=0;
13
14 char lastKeyPressed='g';
15
16
1 // main.cpp
2
3 #include "main_header.h"
4 #include "globalVariables.h"
5 #include "CkeyboardHandler.h"
6 #include "RT_Client.h"
7
8 void bounceBack();
9 // function to avoid user going through walls
10
11 // the thread routine
12 DWORD RTT_routine(LPVOID tParam) {
13    trackingStart();
14    return 0;
15 }
16
17 int main()
18 {
19    double n=0;
20    unsigned int i=0;
21    osg::BoundingBox BBarray[27];
22    osg::Matrixd trans;
23    osg::Matrix headRot;
24    osg::Matrix headRot2;
25    osg::Matrix rot;
26    osg::Vec3f eye;
27    osg::Vec3f dir;
28    osg::Vec3f up;
29    osgViewer::Viewer viewer;
30    //viewer.setUpViewInWindow( 32, 32, 800, 600 );
31
32    // add the handler to the viewer
33    osg::ref_ptr<CkeyboardHandler> rKeyboardHandler = new CkeyboardHandler;
34    viewer.addEventHandler( rKeyboardHandler.get() );
35
36    // set coordinates for the camera:
37    // camera position, point which it is looking at and orientation
38    viewer.getCamera()->setViewMatrixAsLookAt(osg::Vec3d(0,0,0),
39                                                                    osg::Vec3d(0,1,0),
40                                                                    osg::Vec3d(0,0,1));
41
42    osg::Matrixd myNewViewMatrix;
43    myNewViewMatrix = osg::Matrixd (viewer.getCamera()->getViewMatrix());
44
45    osg::ref_ptr<osg::Group> root = new osg::Group ;
46    osg::ref_ptr<osg::Node> modelNode = new osg::Node ;
47    modelNode = osgDB::readNodeFile("gallery.3DS");
48
49    if(!modelNode) {
50        cout << "Model missing" << endl;
51        getchar();
52    }
53
54    osg::ref_ptr<osg::Group> galleryGroup = modelNode->asGroup();
55    cout << "Number of children in gallery node " <<
56        galleryGroup.get()->getNumChildren() << endl;
osg::ref_ptr<osg::PositionAttitudeTransform> myNodeTransform = new osg::PositionAttitudeTransform;
myNodeTransform->setPosition(osg::Vec3(-40.0, 750, 70)); //;

osg::Vec3 *box_position;
box_position = new osg::Vec3(-40.0, 750, 70);
root->addChild(myNodeTransform.get());
myNodeTransform->addChild(modelNode.get());

// create bounding boxes for each wall of the gallery
for (i=0; i< galleryGroup->getNumChildren(); i++)
{
    osg::ComputeBoundsVisitor cbbv_t;
galleryGroup->getChild(i)->accept(cbbv_t);
    BBArray[i]= cbbv_t.getBoundingBox();
    osg::Vec3f sizeBB = BBArray[i].max - BBArray[i].min;

    // set position of the bb
    BBArray[i].set (BBArray[i].min.x() + box_position->x(),
    BBArray[i].min.y() + box_position->y(),
    BBArray[i].min.z() + box_position->z(),
    BBArray[i].max.x() + box_position->x(),
    BBArray[i].max.y() + box_position->y(),
    BBArray[i].max.z() + box_position->z());

    cout << i << " BBmax " << BBArray[i].max.x() << " "
    " BBArray[i].max.y() << " " << BBArray[i].max.z() <<endl;
    cout << i << " BBmin " << BBArray[i].min.x() << " "
    " BBArray[i].min.y() << " " << BBArray[i].min.z() <<endl;
    cout << i << " Bounding Box size " << sizeBB.x() << " " << sizeBB.y()
    " sizeBB.z() << endl;
}

// create a bounding box for the camera
osg::BoundingBox camera_bb;
camera_bb.expandBy(viewer.getCamera()->getBound());
viewer.setSceneData( root );

/// init and start thread
HANDLE RTT_routine_thread;
DWORD tId;
RTT_routine_thread = CreateThread(
NULL,
0,
(LPTHREAD_START_ROUTINE) &RTT_routine,
(LPVOID) 1,
0,
&tId
);

while (!viewer.done())
{
    // update transformation matrices
    rot.makeRotate( myR, osg::Vec3( 0., 0., 1. ) );
    headRot.makeRotate( (myHR*PI/180), osg::Vec3( 1., 0., 0. ) );
    headRot2.makeRotate( (myHR2*PI/180), osg::Vec3( 0., 0., 1. ) );
    trans.makeTranslate(-myX, -myY, -myZ);

    // Set the view matrix (the concatenation of the rotation and translation matrices).
    viewer.getCamera()->setViewMatrix(trans*rot*headRot*headRot2*myNewViewMatrix);
viewer.getCamera()->getViewMatrixAsLookAt(eye, dir, up);

// eye vector: camera position
camera_bb.set(eye.x()-0.5, eye.y()-0.5, eye.z()-0.5,
   eye.x()+0.5, eye.y()+0.5, eye.z()+0.5);

// check if any of the bounding boxes intersects with the camera one
for (i=0; i<26; i++)
{
   if (camera_bb.intersects(BBarray[i]))
   {
      n +=1;
      cout << n << "Collision detected " << " with BB " << i << endl;
      bounceBack();
   }
}

// Draw the next frame.
viewer.frame();

TerminateThread(RTT_routine_thread, 0);

return 0;

// function to avoid user going through walls
void bounceBack()
{
   if (lastKeyPressed == 'w')
   {
      myX = myX-4*sin(myR);
      myY = myY-4*cos(myR);
   }
   else if (lastKeyPressed == 'x')
   {
      myX = myX+4*sin(myR);
      myY = myY+4*cos(myR);
   }
   else if (lastKeyPressed == 'e')
   {
      myX = myX-4*cos(myR);
      myY = myY-4*sin(myR);
   }
   else if (lastKeyPressed == 'q')
   {
      myX = myX+4*cos(myR);
      myY = myY+4*sin(myR);
   }
}
void trackingStart()
{
    WORD wVersionRequested = MAKEWORD(2, 2);
    WSADATA wsaData;
    // Initialize WinSock and check version
    //
    int nRet = WSAStartup(wVersionRequested, &wsaData);
    if (nRet != 0)
    {
        PrintError("Couldn't load winsock library (ws2_32.dll)");
        return ;
    }
    if (wsaData.wVersion != wVersionRequested)
    {
        PrintError("Wrong version of winsock (ws2_32.dll)");
        return ;
    }
    // By default assume you want to connect to QTM at the same machine - just for testing
    //
    CStdString tServer = "localhost";
    CStdString tServer = "137.195.27.252";
    // Example showing how to read data frame by frame
    //
    Example2(tServer, 22222);
    // Release WinSock
    //
    WSACleanup();
    return ;
}

void Example2(CStdString tServer, int nPort)
{
    //
    // Connect to the server
    //
    SOCKET hSocket = Connect(tServer, nPort);
    if (hSocket == INVALID_SOCKET)
    {
        return ;
    }
    SHeader sHeader;
    CStdString tData;
    CStdString tCmd;
    // Welcome message
    //
    if(RecvMessage(hSocket, sHeader, tData))
    {
puts(tData);

// Set protocol version
//
tCmd = "Version 1.0";
SendCommand(hSocket, tCmd);
if(RecvMessage(hSocket, sHeader, tData))
{
  puts(tData);
}

// Set byte order to machine native ordering for easier processing
//
tCmd = "ByteOrder LittleEndian";
SendCommand(hSocket, tCmd);
if(RecvMessage(hSocket, sHeader, tData))
{
  puts(tData);
}

// Request capture parameters
//
std::vector< CStdStringA > atLabels;
tCmd = "SendParameters 6dEuler";
SendCommand(hSocket, tCmd);
if(RecvMessage(hSocket, sHeader, tData))
{
  if(sHeader.nType == 2)
  {
    NBC_CMarkup oXML;
oXML.SetDoc(tData);
    //
    // Read 6d marker labels
    //
    if (oXML.FindChildElem("The_6D") && oXML.IntoElem() &&
    oXML.FindChildElem("Bodies")
    )
    {
      int nLabels = atoi(oXML.GetChildData());
      printf("There are %d labeled bodies\n", nLabels);
      atLabels.resize(nLabels);
      for (int iLabel=0; iLabel<nLabels; iLabel++)
      {
        if (oXML.FindChildElem("Body")
        )
        {
          oXML.IntoElem();
          if (oXML.FindChildElem("Name")
          )
          {
            atLabels.at(iLabel) = oXML.GetChildData();
            printf("Body %d: %s\n", iLabel+1, atLabels.at(iLabel).c_str());
          }
          oXML.OutOfElem();
        }
      }
      oXML.ResetPos();
    }
  
  
  
else
  {
    // Error
    //
    puts(tData);
  }
  
}

// Variables for timing test
//
Appendix 3. First Prototype Code.    15
151 int nReceivedFrames = 0;
152 LARGE_INTEGER nBefore;
153 QueryPerformanceCounter(&nBefore);
154
155 // // Retrieve one data frame
156 // tCmd = "SendCurrentFrame 6dEulaer";  // <<-
157 // tCmd = "StreamFrames FrequencyDivisor:1 3dNoLabels";  //
158 SendCommand(hSocket, tCmd);
159 bool bHasData = true;
160 int nLastFrame = 0;
161 while (bHasData && RecvMessage(hSocket, sHeader, tData))
162 {
163     switch (sHeader.nType)
164     {
165         case 0:
166             // sHeader.nType 0 indicates an error
167             //
168             fprintf(stderr, "Error at SendCurrentFrame: %s\n", tData);  
169             bHasData = false;
170             break;
171         case 1:
172             // sHeader.nType 1 indicates successful completion of an operation and/or a string message
173             //
174             fprintf(stderr, "Message: %s\n", tData);  
175             break;
176         case 3:
177             // sHeader.nType 3 means data
178             //
179             nReceivedFrames++;
180             //
181             // Do some processing here
182             //
183             // PrintFrameNumber(sHeader, tData, nReceivedFrames, nLastFrame);
184             PrintData(sHeader, tData, atLabels);
185             break;
186         case 4:
187             //
188             // sHeader.nType 4 means no more data
189             //
190             bHasData = false;
191             break;
192         default:
193             break;
194     }
195     if (bHasData)
196     {
197         // Retrieve next data frame if last frame was ok.
198         //
199         SendCommand(hSocket, tCmd);
200     }
201 }
202
203 // Disconnect from the server
204 //
205 std::cout << "This is the end ...";
206 getchar();
207 Disconnect(hSocket);
208 }
209 //
210 // Print data to console
211 //
```cpp
void PrintData(SHeader sHeader, CStdString tData, std::vector< CStdStringA > atLabels)
{
    char* pData = const_cast<char*>(tData.c_str());
    SPacketHeader* pHeader = reinterpret_cast< SPacketHeader* >(pData);
    char* pComponent = pData + sizeof(SPacketHeader);
    //float fTime = pHeader->nTimeStamp*0.000001;
    //printf("Time %3.2f\n", fTime, pHeader->nFrameNumber);
    for (int iComponent=0; iComponent<pHeader->nComponentCount; iComponent++)
    {
        //S3dComponent* p3d = reinterpret_cast< S3dComponent* >(pComponent);
        //S3dNoLabelsComponent* p3dNoLabels = reinterpret_cast< S3dNoLabelsComponent* >(pComponent);
        S6dEulerComponent* p6dEuler= reinterpret_cast< S6dEulerComponent* >(pComponent);
        if (p6dEuler->nType==cn6dEuler)
        {
            cout<< "6D Euler data" "<<endl;
            for (int i = 0; i < p6dEuler->nCount; i++)
            {
                if (!strcmp(atLabels.at(i).c_str(),"a_glasses")==0)
                {
                    printf( "Body %d: %f\t%f\t%f\n", i + 1,
                        p6dEuler->aSamples[i].fX,
                        p6dEuler->aSamples[i].fY,
                        p6dEuler->aSamples[i].fZ,
                        p6dEuler->aSamples[i].fR1,
                        p6dEuler->aSamples[i].fR2,
                        p6dEuler->aSamples[i].fR3);
                    if (typeid(p6dEuler->aSamples[i].fR1) == typeid(myHR))
                    {
                        if ((abs(p6dEuler->aSamples[i].fR1 - myHR)) > 0.2)
                        {
                            myHR = p6dEuler->aSamples[i].fR1;
                            if (myHR>45) myHR=45;
                            else if (myHR<45) myHR=-45;
                        }
                    }
                    else
                    {
                        cout<< "no right data received" "<<endl;
                    }
                    pComponent += p6dEuler->nSize;
                }
            }
        }
    }
} // Receive a packet. Data is stored in a CStdString, so allocation and deallocation is made fairly easy
```
while (nHeaderRecved < sizeof(rsHeader)) {
  // Receive start of a new packet from the client. The first 8 bytes holds the header.
  nRecved = recv(hSocket, &uHeader.c+nHeaderRecved, sizeof(rsHeader)-nHeaderRecved, 0);
  if (nRecved == SOCKET_ERROR)
    { PrintError("Error at recv()"); return false; }
  if (nRecved == 0)
    { // The server closed the connection
      return false; }
  nHeaderRecved += nRecved;
}

// Convert the header from network byte order to host byte order
rsHeader.nSize = ntohl(uHeader.s.nSize) - sizeof(rsHeader);
rsHeader.nType = ntohl(uHeader.s.nType);

// Allocate memory to hold the entire packet
rtData.resize(rsHeader.nSize);

// Receive more data until we have read the whole packet
while (nBodyRecved < rsHeader.nSize) {
  // As long as we haven't received enough data, wait for more
  nRecved = recv(hSocket, &rtData[nBodyRecved], rsHeader.nSize-nBodyRecved, 0);
  if (nRecved == SOCKET_ERROR || nRecved == 0)
    { PrintError("Error at recv()"); return false; }
  nBodyRecved += nRecved;
}

// Successfully received a packet
return true;

bool SendCommand(SOCKET hSocket, CStdString tCommand) {
  if (tCommand.size() > BUFFER_SIZE)
    { fprintf(stderr, "Command string too long"); return false; }
  return true;
}
USendBuffer uSendBuffer;

// Commands are string which means type is 1
uSendBuffer.s.nType = htonl(1);

// Header size + length of the string + terminating null char
int nSize = sizeof(SHeader) + tCommand.size() + 1;
uSendBuffer.s.nSize = htonl(nSize);
memcpy(uSendBuffer.c+(sizeof(SHeader)), tCommand.c_str(), tCommand.size() + 1);

int nSent = 0;
int nTotSent = 0;
while (nTotSent < nSize)
{
    nSent = send(hSocket, uSendBuffer.c+nTotSent, nSize-nTotSent, 0);
    if (nSent == SOCKET_ERROR)
    {
        PrintError("Error at send()");
        return false;
    }
    nTotSent += nSent;
}

// Successfully sent the command
return true;

// Connect to the server
SOCKET Connect(CStdString tServer, int nPort)
{
    SOCKET hSocket = socket(AF_INET, SOCK_STREAM, 0);
    sockaddr_in sAddr;

    // First check if the address is a dotted number "A.B.C.D"
    sAddr.sin_addr.s_addr = inet_addr(tServer.c_str());
    if (sAddr.sin_addr.s_addr == INADDR_NONE)
    {
        // If it wasn't a dotted number lookup the server name
        //
        hostent *psHost = gethostbyname(tServer.c_str());
        if (!psHost)
        {
            PrintError("Error looking up host name");
            closesocket(hSocket);
            return INVALID_SOCKET;
        }
        sAddr.sin_addr = *((in_addr*)psHost->h_addr_list[0]);
    }
    sAddr.sin_port = htons(nPort);
    sAddr.sin_family = AF_INET;

    if (connect(hSocket, ( sockaddr*)(&sAddr), sizeof(sAddr)) == SOCKET_ERROR)
    {
        PrintError("Error at connect()");
        closesocket(hSocket);
        return INVALID_SOCKET;
    }

    // Disable Nagle's algorithm
446  //
447  char bNoDelay = 1;
448  if (setsockopt(hSocket, IPPROTO_TCP, TCP_NODELAY, &bNoDelay, sizeof(bNoDelay)))
449  {
450    PrintError("Error at connect()");
451    closesocket(hSocket);
452    return INVALID_SOCKET;
453  }
454  
455  // Successfully connected to the server
456  //
457  // return hSocket;
458  }
459 
460  void Disconnect(SOCKET hSocket)
461  {
462    //
463    // Try to shutdown gracefully
464    //
465    shutdown(hSocket, SD_SEND);
466    int nRecved = 1;
467    char pData[500];
468    while (nRecved > 0)
469    {
470      //
471      // There shouldn't be anything left to receive now, but check just to make sure
472      //
473      nRecved = recv(hSocket, pData, sizeof(pData), 0);
474    }
475    closesocket(hSocket);
476  }
477 }
478
479  void PrintError(char *t)
480  {
481    char *tError = NULL;
482    DWORD nError = GetLastError();
483    DWORD nRet = FormatMessage(
484      FORMAT_MESSAGE_ALLOCATE_BUFFER | FORMAT_MESSAGE_FROM_SYSTEM,
485      NULL, nError, 0, reinterpret_cast<LPTSTR>(tError), 0, NULL);
486    fprintf(stderr,"%s: %d %s\n", t, nError, tError);
487    LocalFree(tError);
488  }
489  
490
APPENDIX 4. SECOND PROTOTYPE CODE
• List of files included in the project:

- Solution 'MyGallery' (1 project)
  - MyGallery
    - Header Files
      - KeyboardHandler.h
      - globalVariables.h
      - main_header.h
      - NBC_Markup.h
      - RT_Client.h
      - StdString.h
    - Resource Files
    - Source Files
      - Keyboard-handler.cpp
      - globalVariables.cpp
      - Main.cpp
      - NBC_Markup.cpp
      - RT_Client.cpp

**NOTE:** NBC_Markup.h, StdString.h and NBC_Markup.cpp have been omitted here as they are not relevant to this project.
// CKeyboardHandler.h

#ifndef CKEYBOARDHANDLER_H
#define CKEYBOARDHANDLER_H

class CKeyboardHandler : public osgGA::GUIEventExceptionHandler
{
  public:
    virtual bool handle(osgGA::GUIEventAdapter& ea, osgGA::GUIActionAdapter& aa, osg::Object* pObject, osg::NodeVisitor* pNodeVisitor);
    // Function with handles the input from the keyboard
    // for each key it does an action
};

#endif
// globalVariables.h

// global variables

#ifndef GLOBALVARIABLES_H
#define GLOBALVARIABLES_H

extern const double PI;
extern double myX;
extern double myY;
extern double myZ;
extern double myR;
extern double myHR;
extern double myHR2;
extern double sphereX;
extern double sphereZ;
extern double sphereY;
extern float sphY_t;
extern double markerX;
extern double markerZ;
extern double markerY;
extern float markerXmax, markerXmin;
extern float markerYmax, markerYmin;
extern float markerZmax, markerZmin;
extern float xfactor, zfactor, yfactor;
extern char lastKeyPressed;
extern bool photosIn;
extern bool selectFrame;
extern float margin;
extern double phX;
extern bool spherePressed;
extern bool calibrationDone;
extern bool trackHead;

#endif
// main_header.h

#ifndef MAIN_HEADER_H
#define MAIN_HEADER_H

#include <iostream>
#include <osg/Group>
#include <osg/Node>
#include <osgDB/ReadFile>
#include <osgViewer/Viewer>
#include <osg/MatrixTransform>
#include <osg/PositionAttitudeTransform>
#include <osg/BoundingBox>
#include <osgDB/ReadFile>
#include <osg/StateSet>
#include <math.h>
#include <windows.h>
#include <string.h>

using namespace std;

#endif
// RT_Client.h

#include "StdString.h"
#include <vector>
#include <iostream>
#include <typeinfo>
#include <cmath>

#pragma pack ( push, 4 )

typedef struct _SHeader
{
    long nSize;
    long nType;
} SHeader;

typedef union _UHeader
{
    SHeader s;
    char c[sizeof(SHeader)];
} UHeader;

#define BUFFER_SIZE 500

typedef union _USendBuffer
{
    SHeader s;
    char c[BUFFER_SIZE+sizeof(SHeader)];
} USendBuffer;

typedef struct _S3dNoLabels
{
    double fX;
    double fY;
    double fZ;
    long nId;
    long nPadding;
} S3dNoLabels;

typedef struct _S3d
{
    double fX;
    double fY;
    double fZ;
} S3d;

typedef struct _SForce
{
    S3d sForce;
    S3d sMoment;
    S3d sPoint;
} SForce;

typedef struct _S6d
{
    double fX;
    double fY;
    double fZ;
    double fR[9];
} S6d;

typedef struct _S6dEuler
{
    double fX;
    double fY;
    double fZ;
    double fR1;
    double fR2;
    double fR3;
} S6dEuler;
typedef struct _SPacketHeader
{
    _int64 nTimeStamp;
    long nFrameNumber;
    long nComponentCount;
} SPacketHeader;

typedef struct _S3dComponent
{
    long nSize;
    long nType;
    long nCount;
    long nPadding;
    S3d as3dPoints[BUFFER_SIZE];
} S3dComponent;

typedef struct _S3dNoLabelsComponent
{
    long nSize;
    long nType;
    long nCount;
    long nPadding;
    S3dNoLabels as3dPoints[BUFFER_SIZE];
} S3dNoLabelsComponent;

typedef struct _SForceComponent
{
    long nSize;
    long nType;
    long nCount;
    long nPadding;
    SForce asForceSamples[BUFFER_SIZE];
} SForceComponent;

typedef struct _SAnalogComponent
{
    long nSize;
    long nType;
    long nCount;
    long nPadding;
    double aSamples[BUFFER_SIZE];
} SAnalogComponent;

typedef struct _S6dComponent
{
    long nSize;
    long nType;
    long nCount;
    long nPadding;
    S6d aSamples[BUFFER_SIZE];
} S6dComponent;

typedef struct _S6dEulerComponent
{
    long nSize;
    long nType;
    long nCount;
    long nPadding;
    S6dEuler aSamples[BUFFER_SIZE];
} S6dEulerComponent;

enum EComponentTypes
{
    cn3d = 1,
    cn3dNoLabels,
    cnAnalog,
    cnForce,
    cn6d,
    cn6dEuler
};

#pragma pack ( pop )

void PrintError(char *t);
bool RecvMessage(SOCKET hSocket, SHeader & rsHeader, CStdString & rtData);
bool SendCommand(SOCKET hSocket, CStdString tCommand);
SOCKET Connect(CStdString tServer, int nPort);
void Disconnect(SOCKET hSocket);
void PrintData(SHeader sHeader, CStdString tData, std::vector<CStdStringA> atLabels);
void Example2(CStdString tServer, int nPort);
void trackingStart();

```cpp
// CKeyboardHandler.cpp

#include "main_header.h"
#include "globalVariables.h"
#include "CKeyboardHandler.h"

bool CKeyboardHandler::handle(const osgGA::GUIEventAdapter &ea, osgGA::GUIActionAdapter &aa, osg::Object *pObject, osg::NodeVisitor *pNodeVisitor)
{
    osgViewer::Viewer* pViewer = dynamic_cast<osgViewer::Viewer*>(&aa);
    if ( !pViewer )
    {
        return false;
    }
    if ( ea.getEventType()==osgGA::GUIEventAdapter::KEYDOWN )
    {
        switch (ea.getKey())
        {
        case 'e':  // move laterally to the right - E
            //cout << "e pressed: move to the right " << endl;
            myX += 3*cos(myR);
            myY += 3*sin(myR);
            lastKeyPressed = 'e';
            return true;  // return true, event handled
            break;
        case 'q':  // move laterally to the left - Q
            //cout << "a pressed: move to the left " << endl;
            myX -= 3*cos(myR);
            myY += 3*sin(myR);
            lastKeyPressed = 'q';
            return true;
            break;
        case 'w':  // move forward - W
            //cout << "w pressed: move forward " << endl;
            myX -= 3*sin(myR);
            myY -= 3*cos(myR);
            lastKeyPressed = 'w';
            return true;
            break;
        case 'x':  // move backwards - X
            //cout << "x pressed: move backwards " << endl;
            myX -= 3*sin(myR);
            myY += 3*cos(myR);
            lastKeyPressed = 'x';
            return true;
            break;
        case 'c':  // move up - C
            //cout << "c pressed: move up " << endl;
            myZ += 1;
            return true;
            break;
        case 'z':  // move down - Z
            //cout << "z pressed: move down " << endl;
            myZ -= 1;
            return true;
            break;
        // rotate left - A
        case 'a':
            //cout << "a pressed: rotate left " << endl;
            myR -= 0.025;
            if(myR<=margin) myR=2*PI-margin;
            return true;
            break;
        // rotate right - D
        case 'd':
            //cout << "d pressed: rotate right " << endl;
```
myR += 0.025;
if (myR > (2 * osg::PI - margin)) myR -= margin;
return true;
break;

// pointer up - I
case 'i':
sphereZ += 0.5;
return true;
break;

// pointer down - M
case 'm':
sphereZ -= 0.5;
return true;
break;

// pointer left - J
case 'j':
sphereX -= 0.3;
return true;
break;

// pointer right - L
case 'l':
sphereX += 0.3;
return true;
break;

// pointer further - U
case 'u':
sphereY += 0.1;
return true;
break;

// pointer closer - N
case 'n':
sphereY -= 0.1;
return true;
break;

// calibration done - start
case 's':
calibrationDone = true;
return true;
break;
return false;
}
// globalVariables.cpp

#include "globalVariables.h"

extern const double PI=3.141592;

double myX=0;
double myY=0;
double myZ=100;
double myR=0;
double myHR=0;
double myHR2=0;

extern double sphereX=0;
extern double sphereZ=0;
extern double sphereY=0;
extern float sphY_t=0;

extern double markerX=0;
extern double markerZ=0;
extern double markerY=0;
extern float markerXmax=0;
extern float markerXmin=0;
extern float markerYmax=0;
extern float markerYmin=0;
extern float markerZmax=0;
extern float markerZmin=0;
extern float xfactor=0;
extern float zfactor=0;
extern float yfactor=0;
char lastKeyPressed='g';
extern bool photosIn=false;
extern bool selectFrame=false;
extern float margin=0.4;
extern double phX=0;
extern bool spherePressed = false;
extern bool calibrationDone = false;
extern bool trackHead = true;
// main.cpp

#include "main_header.h"
#include "globalVariables.h"
#include "CkeyboardHandler.h"
#include "RT_Client.h"

// Function prototypes

osg::ref_ptr<osg::Node> myModelNode;
osg::ref_ptr<osg::Node> myFrameNode;
osg::ref_ptr<osg::Node> boxNode;
osg::ref_ptr<osg::Node> photoNode[23];
osg::ref_ptr<osg::Image> photoImage[21];
osg::ref_ptr<osg::MatrixTransform> boxTransform;
osg::ref_ptr<osg::MatrixTransform> photoTransform[23];
osg::ref_ptr<osg::MatrixTransform> inFrameTransform[21];
osg::ref_ptr<osg::MatrixTransform> frameTransform[21];
osg::ref_ptr<osg::MatrixTransform> mySphereTransform;
osg::ref_ptr<osg::AnimationPathCallback> photoAnimCallBack[3][8];
osg::ref_ptr<osg::AnimationPathCallback> opp_photoAnimCallBack[3][8];

bool boundingBox BBarray[27];

struct locationInfo
{
  osg::Vec3f position;
  float rotation;
  float scaleFactor;
  bool withPhoto;
};

locationInfo frames_locationInfo[21];

// the thread routine

DWORD RTT_routine(LPVOID tParam) {
  trackingStart();
  return 0;
}
```cpp
int main() {
    bool boxSelected = false;
    bool photosIn = true;
    int fframe = 0;
    int selectedPhoto = 21;
    bool calibration = false;
    int p = 0;
    double n = 0;
    osg::Matrixd trans;
    osg::Matrix headRot;
    osg::Matrix headRot2;
    osg::Matrix rot;
    osg::Vec3f eye;
    osg::Vec3f dir;
    osg::Vec3f up;

    // set up viewer
    osgViewer::Viewer viewer;
    //viewer.setUpViewInWindow( 32, 32, 1024, 768 );
    // set coordinates for the camera:
    //camera position, point which it is looking at and orientation
    viewer.getCamera()->setViewMatrixAsLookAt ( osg::Vec3d(0,0,0),
            osg::Vec3d(0,1,0),
            osg::Vec3d(0,0,1));

    // antialiasing
    osg::DisplaySettings::instance()->setNumMultiSamples(4);
    // add the handler to the viewer
    osg::ref_ptr< CKeyboardHandler > rKeyboardHandler = new CKeyboardHandler;
    viewer.addEventHandler( rKeyboardHandler.get() );

    // create a bounding box for the camera
    osg::BoundingBox camera_bb;
    camera_bb.expandBy(viewer.getCamera()->getBound());

    viewer.setLightingMode(osg::View::SKY_LIGHT);

    // view matrix
    osg::Matrixd myNewViewMatrix;
    myNewViewMatrix = osg::Matrixd (viewer.getCamera()->getViewMatrix());

    // create scene
    osg::ref_ptr<osg::Group> myRoot = startupScene();

    // add lights to the scene
    osg::ref_ptr<osg::Group> myLightGroup = createLights(myRoot);
    myRoot->addChild(myLightGroup->get());

    // add calibration frame in front of the camera
    osg::ref_ptr<osg::MatrixTransform> calibrationTransform = new osg::MatrixTransform;
    calibrationTransform->setMatrix(( osg::Matrix::rotate(PI, osg::Vec3(0,0,1))
            *(osg::Matrix::translate(osg::Vec3(0,200,125)) ) ));
    calibrationTransform->addChild(createCalibrationNode());
    myRoot->addChild(calibrationTransform->get());
```
// create small sphere for pointer
mySphereTransform = createSpherePointer();
boxTransform->addChild(mySphereTransform);

// create bounding boxes for the collision detection
osg::BoundingBox myGalleryBB = getBoundingBoxes(myModelNode);

// position frames on the walls
hangFrames(myGalleryBB, myFrameNode, myRoot);

// load photographs
loadImagesandAssignTextures();

// create the animations for the photos
createPhotoAnimations();

viewer.setSceneData(myRoot);

// initialize and start THREAD
HANDLE RTT_routine_thread;
DWORD tId;
RTT_routine_thread = CreateThread(NULL, 0, (LPTHREAD_START_ROUTINE)&RTT_routine, (LPVOID)1, 0, &tId);

// calibration for hand tracking
handTrackingCalibration();

// initialise. all photos unselected
for(int i=0; i<23; i++)
{
    photoSelection[i]=true;
}

// VIEWER LOOP
while (!viewer.done())
{
    // height calibration
    if (calibrationDone)
    {
        myRoot->removeChild(calibrationTransform.get());
        calibrationDone=false;
    }

    //update transformation matrices
    rot.makeRotate(myR, osg::Vec3(0., 0., 1.));
    headRot.makeRotate((myHR*PI/180), osg::Vec3(-1., 0., 0.));
    headRot2.makeRotate((myHR2*PI/180), osg::Vec3(0., 0., 1.));
    trans.makeTranslate(-myX, -myY, -myZ);

    // Set the view matrix (the concatenation of the rotation and translation matrices)
    viewer.getCamera()->setViewMatrix(trans*rot*headRot*headRot2*myNewViewMatrix);

    viewer.getCamera()->getViewMatrixAsLookAt(eye, dir, up);
    // eye vector: camera position

    // box, photos and pointer have to move as camara moves
boxTransform->setMatrix(
    (osg::Matrix::translate(osg::Vec3(+10-phX,+24,-10)))
    * (osg::Matrix::rotate(-myHR2*PI/180, osg::Vec3(0,0,1)))
    * (osg::Matrix::rotate(-myHR*PI/180), osg::Vec3(1,0,0)))
    * (osg::Matrix::rotate (-myR, osg::Vec3( 0., 0., 1. )))
    * (osg::Matrix::translate(osg::Vec3(eye.x(),eye.y(),eye.z())))
);

// get position info of the pointer (sphereX, sphereZ, sphereY)
spherePressed = getPointerInfo();

// apply movement to pointer - only moving in x-z plane
mySphereTransform->setMatrix ( osg::Matrix::translate(osg::Vec3f(-10+sphereX+phX,-4,sphereZ)));

// check if box is selected

// if pointer at box position
if (!
    (mySphereTransform->getMatrix().getTrans().x()>=1.5)
    && (mySphereTransform->getMatrix().getTrans().x()<2) )
    && (mySphereTransform->getMatrix().getTrans().z()>=1)
    && (mySphereTransform->getMatrix().getTrans().z()<2.5) )
{
    if (spherePressed)
    {
        // not able to select while photos are going in/out
        if ( (photoAnimCallBack[2][2]->getAnimationTime()>=0.1)
            && (photoAnimCallBack[2][2]->getAnimationTime()<=2.0))
            || ( (opp_photoAnimCallBack[2][2]->getAnimationTime()>=0.1)
                && (opp_photoAnimCallBack[2][2]->getAnimationTime()<=2.0))
        {
            boxSelected = false;
        }
        else
        {
            cout << " box selected" " endl;
            boxSelected = true;
            spherePressed = false;
        }
    }
}

if (boxSelected)
{
    if (photosIn) // if photos are in the box put them out with an animation
    {
        // stop tracking head
        trackHead = false;
        myHR=0;
        myHR2=0;

        p=0;
        for (int i=0; i<3; i++)
        {
            for (int j=0; j<8; j++)
            {
                if (((i-2) && (j-3)) )
            }
            else
            {
                photoAnimCallBack[i][j]->reset();
                photoAnimCallBack[i][j]->setPause(false);
                photoTransform[p]->
                    setUpdateCallback(photoAnimCallBack[i][j].get())
                    p+=1;
                }
            }
        }
        photosIn = false;
    }
    else if (!photosIn) // if photos are out the box put them in with an animation
    {
trackHead = true;  // start tracking head

p=0;
for (int i=0; i<3; i++)
{
   for (int j=0; j<8; j++)
   {
      if ((i==2) && (j==3)) {}
      else
      {
         opp_photoAnimCallBack[i][j]->setPause(false);
         photoTransform[p]->
            setUpdateCallback(opp_photoAnimCallBack[i][j].get()) ;
         opp_photoAnimCallBack[i][j]->reset();
         p+=1;
      }
   }
   photosIn = true;
}
boxSelected = false;

if (!photosIn) // photos are out
{
   // wait until animation is finished
   if (photoAnimCallBack[2][2]->getAnimationTime()>2.0)
   {
      selectedPhoto = selectPhoto(mySphereTransform);
      //cout << "photo number "<< selectedPhoto << " selected" << endl;
      if (selectedPhoto<23)
      {
         if (selectedPhoto==18) // more photos
            phX +=26.75;
         else if (selectedPhoto==19) // back
            phX -=26.75;
         else
         {
            // find closest frame
            fframe = findClosestFrame(eye);
            //cout << "closest frame: " << fframe << endl;
            // if there is a closest frame and it hasnt a picture on it
            if ((fframe<22) && (frames_locationInfo[fframe].withPhoto))
            {
               // remove animation of that photo
               photoTransform[selectedPhoto]->removeUpdateCallback
                  (photoTransform[selectedPhoto]->getUpdateCallback());
               // create a new transform node with the properties of
               // the selected frame (stored in the struct)
               inFrameTransform[selectedPhoto] = new osg::MatrixTransform;
               inFrameTransform[selectedPhoto]->setMatrix(
                  (osg::Matrix::scale
                      {frames_locationInfo[fframe].scaleFactor, 1,
                       frames_locationInfo[fframe].scaleFactor})
                      *(osg::Matrix::rotate
                         {frames_locationInfo[fframe].rotation+PI, 0, 0, 1})
                      *(osg::Matrix::translate
                         {frames_locationInfo[fframe].position})
                  );
               myRoot->addChild(inFrameTransform[selectedPhoto].get());
               inFrameTransform[selectedPhoto]->
            }
   }
}
addChild(photoNode[selectedPhoto].get());

// remove selected photo from its original location on scene graph
photoTransform[selectedPhoto]->
    removeChild(photoNode[selectedPhoto].get());
boxTransform->
    removeChild(photoTransform[selectedPhoto].get());

// mark the photo as selected so it can not be selected again
photoSelection[selectedPhoto] = false;

// mark frame as "with photo"
frames_locationInfo[fframe].withPhoto = true;

// update camera bounding box
camera_bb.set (eye.x()-5, eye.y()-5, eye.z()-5, eye.x()+5, eye.y()+5, eye.z()+5);

// check if any of the bounding boxes intersects with the camera one
for (int i=0; i<27; i++)
{
    if (camera_bb.intersects(BBarray[i]))
    {
        n +=1;
        // cout << n << "Collision detected " << " with BB " << i << endl;
        bounceBack();
    }
}

// Draw the next frame.
viewer.frame();

} // End viewer

// Terminate Thread
TerminateThread(RTT_routine_thread, 0);

return 0;

Appendix 4. Second Prototype Code. 16
FUNCTION DEFINITIONS

startupScene

Function which creates the root node, transformation nodes and nodes for the main objects in the scene.

It also creates textures and apply them.

It returns the root node created.

osg::ref_ptr<osg::Group> startupScene()
{
    // Creating the root node
    osg::ref_ptr<osg::Group> root = new osg::Group;

    // LOAD 3D MODELS
    // gallery
    myModelNode = new osg::Node;
    myModelNode = osgDB::readNodeFile("Models/gallery.3DS");
    if(!myModelNode) {
        cout << "Model missing" << endl;
        getchar();
    }

    // box
    boxNode = new osg::Node;
    boxNode = osgDB::readNodeFile("Models/photo_box.3DS");
    if(!boxNode) {
        cout << "Model missing" << endl;
        getchar();
    }

    // frame
    myFrameNode = new osg::Node;
    myFrameNode = osgDB::readNodeFile("Models/frame.3DS");
    if(!myFrameNode) {
        cout << "Model missing" << endl;
        getchar();
    }

    // 21 photos
    for (int i=0; i<23; i++)
    {
        photoNode[i] = new osg::Node;
        photoNode[i] = osgDB::readNodeFile("Models/plane.3DS");
        if(!photoNode[i]) {
            cout << "Model missing" << endl;
            getchar();
        }
    }
root->addChild(myModelNode.get());

// photo box transform
boxTransform = new osg::MatrixTransform;
root->addChild(boxTransform.get());
boxTransform->addChild(boxNode.get());

// photoTransform
for (int i=0; i<23; i++)
{
    photoTransform[i] = new osg::MatrixTransform;
    // control point 0
    photoTransform[i]->setMatrix((osg::Matrix::scale(osg::Vec3f(0.0175, 1, 0.0175)))
        *(osg::Matrix::rotate(osg::Quat(-osg::PI, 0, 1, 0)))
        *(osg::Matrix::translate(osg::Vec3f(0, 0.05*i, 0.01*i)));

    boxTransform->addChild(photoTransform[i].get());
    photoTransform[i]->addChild(photoNode[i].get());
}

// apply textures to the scene

floor osg::ref_ptr<osg::Group> myModelGroup = myModelNode->asGroup();

osg::ref_ptr<osg::Image> floorImage = new osg::Image;
floorImage = osgDB::readImageFile("Textures/Floor.jpg");

osg::ref_ptr<osg::StateSet> floorState = new osg::StateSet;
floorState = myModelGroup->getFirstChild()->getOrCreateStateSet();

osg::ref_ptr<osg::Texture2D> floorTexture = new osg::Texture2D;
floorTexture = setImage(floorImage.get());

floorState->setTextureAttributeAndModes(0, floorTexture.get());

// box

osg::ref_ptr<osg::Image> boxImage = new osg::Image;
boxImage = osgDB::readImageFile("Textures/Cardboard.jpg");

osg::ref_ptr<osg::StateSet> boxNodeState = new osg::StateSet;
boxNodeState = boxNode->getFirstChild()->getOrCreateStateSet();

osg::ref_ptr<osg::Texture2D> boxTexture = new osg::Texture2D;
boxTexture = setImage(boxImage.get());

boxNodeState->setTextureAttributeAndModes(0, boxTexture.get());

// frame

osg::ref_ptr<osg::Group> myFrameGroup = myFrameNode->asGroup();

osg::ref_ptr<osg::StateSet> frameNodeSt = new osg::StateSet;
frameNodeSt = myFrameGroup->getFirstChild()->getOrCreateStateSet();

osg::ref_ptr<osg::Material> frameNodeMaterial = new osg::Material;
frameNodeMaterial->setDiffuse(osg::Material::FRONT_AND_BACK, osg::Vec4(0.5, 0.0, 0.1, 1.0));

frameNodeSt->setAttribute(frameNodeMaterial.get());

return root;
osg::ref_ptr<osg::Group> createLights(osg::ref_ptr<osg::Group> root)
{
    osg::ref_ptr<osg::StateSet> myLightState (root->getOrCreateStateSet());
    //Create a light group node
    osg::ref_ptr<osg::Group> lightGroup (new osg::Group);
    //Create a light source objet
    osg::ref_ptr<osg::LightSource> myLightSource = new osg::LightSource;
    //Create light object
    osg::ref_ptr<osg::Light> myLight = new osg::Light();
    //Set the properties of the light objects (position, etc)
    myLight->setLightNum(1);
    myLight->setPosition(osg::Vec4(-312.5, 375, 250, 1));
    myLight->setAmbient(osg::Vec4(0.6, 0.6, 0.6, 0.1f));
    myLight->setDiffuse(osg::Vec4(0.95, 0.9, 0.8, 1.0));
    myLight->setDirection(osg::Vec3(0, 0, 0));
    myLight->setConstantAttenuation(1.0);
    myLight->setLinearAttenuation(0.0075);
    //Add the light object to the light source node
    myLightSource->setLight(myLight);
    myLightSource->setLocalStateSetModes(osg::StateAttribute::ON);
    myLightSource->setStateSetModes(*myLightState, osg::StateAttribute::ON);

    // light number 2
    osg::ref_ptr<osg::LightSource> myLightSource2 = new osg::LightSource;
    osg::ref_ptr<osg::Light> myLight2 = new osg::Light();
    myLight2->setLightNum(2);
    myLight2->setPosition(osg::Vec4(+312.5, 375, 200, 1));
    myLight2->setAmbient(osg::Vec4(0.6, 0.6, 0.6, 1.0f));
    myLight2->setDiffuse(osg::Vec4(0.95, 0.9, 0.8, 1.0));
    myLight2->setDirection(osg::Vec3(0, 0, 0));
    myLight2->setConstantAttenuation(1.0);
    myLight2->setLinearAttenuation(0.0075);
    myLightSource2->setLight(myLight2);
    myLightSource2->setLocalStateSetModes(osg::StateAttribute::ON);
    myLightSource2->setStateSetModes(*myLightState, osg::StateAttribute::ON);

    // light number 3
    osg::ref_ptr<osg::LightSource> myLightSource3 = new osg::LightSource;
    osg::ref_ptr<osg::Light> myLight3 = new osg::Light();
    myLight3->setLightNum(3);
    myLight3->setPosition(osg::Vec4(0, 375, 200, 1));
    myLight3->setAmbient(osg::Vec4(0.6, 0.6, 0.6, 0.1f));
    myLight3->setDiffuse(osg::Vec4(0.95, 0.95, 0.9, 1.0));
    myLight3->setDirection(osg::Vec3(0, 0, 0));
    myLight3->setConstantAttenuation(1.0);
    myLight3->setLinearAttenuation(0.0075);
    myLightSource3->setLight(myLight3);
    myLightSource3->setLocalStateSetModes(osg::StateAttribute::ON);
    myLightSource3->setStateSetModes(*myLightState, osg::StateAttribute::ON);

629 // light number 4
630 osg::ref_ptr<osg::LightSource> myLightSource4 = new osg::LightSource;
631 osg::ref_ptr<osg::Light> myLight4 = new osg::Light();
632 myLight4->setLightNum(4);
633 myLight4->setPosition(osg::Vec4(-312.5, 975, 200, 1));
634 myLight4->setAmbient(osg::Vec4(0.6, 0.6, 0.6, 1.0f));
635 myLight4->setDiffuse(osg::Vec4(0.95, 0.9, 0.8, 1.0));
636 myLight4->setDirection(osg::Vec3(0, 0, 0));
637 myLight4->setConstantAttenuation(1.0);
638 myLight4->setLinearAttenuation(0.0075);
639 myLightSource4->setLight(myLight4);
640 myLightSource4->setLocalStateSetModes(osg::StateAttribute::ON);
641 myLightSource4->setStateSetModes(*myLightState, osg::StateAttribute::ON);
642
643 // light number 5
644 osg::ref_ptr<osg::LightSource> myLightSource5 = new osg::LightSource;
645 osg::ref_ptr<osg::Light> myLight5 = new osg::Light();
646 myLight5->setLightNum(5);
647 myLight5->setPosition(osg::Vec4(187.5, 975, 200, 1));
648 myLight5->setAmbient(osg::Vec4(0.6, 0.6, 0.6, 1.0f));
649 myLight5->setDiffuse(osg::Vec4(0.95, 0.95, 0.9, 1.0));
650 myLight5->setDirection(osg::Vec3(0, 0, 0));
651 myLight5->setConstantAttenuation(1.0);
652 myLight5->setLinearAttenuation(0.0075);
653 myLightSource5->setLight(myLight5);
654 myLightSource5->setLocalStateSetModes(osg::StateAttribute::ON);
655 myLightSource5->setStateSetModes(*myLightState, osg::StateAttribute::ON);
656
657 // Add the light source nodes to the light group node
658 lightGroup->addChild(myLightSource);
659 lightGroup->addChild(myLightSource2);
660 lightGroup->addChild(myLightSource3);
661 lightGroup->addChild(myLightSource4);
662 lightGroup->addChild(myLightSource5);
663
664 return lightGroup;
665
666 }
createCalibrationNode

it creates a plane with the texture for height calibration
it returns the node with that plane

osg::ref_ptr<osg::Node> createCalibrationNode()
{
    osg::ref_ptr<osg::Node> calibrationNode = new osg::Node;
    calibrationNode = osgDB::readNodeFile("Models/plane.3DS");
    osg::ref_ptr<osg::StateSet> calibrationSt = new osg::StateSet;
    calibrationSt = calibrationNode->getOrCreateStateSet();
    osg::ref_ptr<osg::Image> calibrationImage = new osg::Image;
    calibrationImage = osgDB::readImageFile("Textures/calibration.jpg");
    osg::ref_ptr<osg::Texture2D> calibrationTexture = new osg::Texture2D;
    calibrationTexture->setImage(calibrationImage.get());
    calibrationSt->setTextureAttributeAndModes(0, calibrationTexture.get());
    return calibrationNode;
}
createSpherePointer

Function which creates a green small sphere.
It returns the transform node which contains it.

osg::ref_ptr<osg::MatrixTransform> createSpherePointer()
{
    osg::ref_ptr<osg::MatrixTransform> sphereTransform = new osg::MatrixTransform;
    osg::ref_ptr<osg::Sphere> mySphere = new osg::Sphere();
    mySphere->setRadius(0.35);
    osg::ref_ptr<osg::ShapeDrawable> mySphereDrawable = new osg::ShapeDrawable(mySphere);
    osg::ref_ptr<osg::Geode> mySphereGeode = new osg::Geode;
    mySphereGeode->addDrawable(mySphereDrawable.get());
    sphereTransform->addChild(mySphereGeode.get());
    osg::ref_ptr<osg::StateSet> sphereSt = mySphereGeode->getOrCreateStateSet();
    osg::ref_ptr<osg::Material> modelNodeMaterial = new osg::Material;
    modelNodeMaterial->setDiffuse(osg::Material::FRONT_AND_BACK, osg::Vec4(0.0, 0.8, 0.2, 1.0));
    sphereSt->setAttribute(modelNodeMaterial.get());
    return sphereTransform;
}
osg::BoundingBox getBoundingBoxes(osg::Node* modelNode)
{
    osg::BoundingBox galleryBB;
    osg::ComputeBoundsVisitor cbbv_g;
    modelNode->accept(cbbv_g);
    galleryBB = cbbv_g.getBoundingBox();

    // size of the BB of the whole gallery
    osg::ref_ptr<osg::Group> myGalleryGroup = modelNode->asGroup();
    cout << "Number of children in gallery node " <<
    myGalleryGroup->get() -> getNumChildren() << endl << endl;

    // create the BB for each of the walls
    for (int i = 0; i < myGalleryGroup->get() -> getNumChildren(); i++)
    {
        osg::ComputeBoundsVisitor cbbv_t;
        myGalleryGroup->getChild(i) -> accept(cbbv_t);
        BBArray[i] = cbbv_t.getBoundingBox();

        osg::Vec3f sizeBB = BBArray[i].max - BBArray[i].min;

        //cout << i << " BBox " << BBArray[i] . max.x() << " " <<
        // BBArray[i] . max.y() << " " << BBArray[i] . max.z() << endl;

        //cout << i << " BBox " << BBArray[i] . min.x() << " " <<
        // BBArray[i] . min.y() << " " << BBArray[i] . min.z() << endl;

        //cout << i << " Bounding Box size " << sizeBB.x() << " " <<
        // sizeBB.y() << " " << sizeBB.z() << endl << endl;
    }

    return galleryBB;
}
void hangFrames(osg::BoundingBox galleryBB, osg::Node* frameNode, osg::Group* root) {
    int numberOfFrames=1;
    float scaleFrame;
    float rotFrame;
    int j=0;
    float x,y;
    bool firstnotDone=true;
    float sepFromWall = 2;
    osg::Vec3f size_galleryBB = galleryBB._max - galleryBB._min;

    for (int i=0; i<27; i++)
    {
        numberOfFrames=1;
        scaleFrame = 1;
        rotFrame=0;
        osg::Vec3f sizeBB = BBarray[i]._max - BBarray[i]._min;

        /*cout << i << " Bounding Box size " << sizeBB.x() << " " << sizeBB.y() << " " << sizeBB.z() << endl << endl;*/

        if (sizeBB.z()>200)
        {
            if (sizeBB.x()<9) //((sizeBB.x()<8)||(sizeBB.x()==8))
            {
                // "VERTICAL" WALLS
                if (sizeBB.y()>700) numberOfFrames = 2;

                for (int n=1; n<(numberOfFrames+1); n++)
                {
                    if (floor(abs(BBarray[i]._min.x()-galleryBB._min.x()))==0)
                    {
                        if (numberOfFrames>1)
                            y = sizeBB.y()*(2*n-1)*0.25 + BBarray[i]._min.y();
                        else
                            y = sizeBB.y()*n*0.5 + BBarray[i]._min.y();
                        x = BBarray[i]._max.x()+ sepFromWall;
                    }
                    else if (floor(abs(BBarray[i]._max.x()-galleryBB._max.x()))==0)
                    {
                        if (numberOfFrames>1)
                            y = sizeBB.y()*(2*n-1)*0.25 + BBarray[i]._min.y();
                        else
                            y = sizeBB.y()*n*0.5 + BBarray[i]._min.y();
                        x = BBarray[i]._min.x()- sepFromWall;
                        rotFrame = PI;
                    }
                    else
                    {
                        y = sizeBB.y()*n*0.5 + BBarray[i]._min.y();
                    }
                }
            }
        }
    }
}
if (sizeBB.y()<300)
{
  // corridors left walls
  if(abs(BBArray[i]_.max.x()-galleryBB_.max.x())>
    size_galleryBB_.x()/2)
  {
    x = BBArray[i]_.min.x() - sepFromWall;
    rotFrame = PI;
  }
  else
  {
    x = BBArray[i]_.max.x() + sepFromWall;
  }
}
else if (firstnotDone) //right
{
  x = BBArray[i]_.max.x()+ sepFromWall;
  i =1;
  firstnotDone = false;
}
else // left
{
  x = BBArray[i]_.min.x() - sepFromWall;
  rotFrame = PI;
  firstnotDone=true;
}
}

frameTransform[j] = new osg::MatrixTransform;
frameTransform[j]->setMatrix(
  (osg::Matrix::scale(osg::Vec3(scaleFrame,1,scaleFrame)))
  * (osg::Matrix::rotate(osg::PI_2,rotFrame,osg::Vec3(0,0,1)))
  * (osg::Matrix::translate(osg::Vec3(x,y,125)))
);

root->addChild(frameTransform[j].get());
frameTransform[j]->addChild(frameNode);
frames_locationInfo[j].position = osg::Vec3f(x,y,125);
frames_locationInfo[j].rotation = 0.5*PI+rotFrame;
frames_locationInfo[j].scaleFactor = scaleFrame;
frames_locationInfo[j].withPhoto = false;

//cout << "Frame " " j " hung vertical" " endl;
}
x = sizeBB.x() * n * 0.5 + BBarray[i].min.x();
y = BBarray[i].min.y() + sepFromWall;
rotFrame = PI;
}
else
{
    if (firstnotDone)
    {
        x = sizeBB.x() * n * 0.5 + BBarray[i].min.x();
y = BBarray[i].min.y() - sepFromWall;
i -= 1;
        firstnotDone = false;
    }
    else
    {
        x = sizeBB.x() * n * 0.5 + BBarray[i].min.x();
y = BBarray[i].max.y() + sepFromWall;
rotFrame = PI;
        firstnotDone = true;
    }
}
frameTransform[j] = new osg::MatrixTransform;
frameTransform[j]->setMatrix(
    {osg::Matrix::scale(osg::Vec3(scaleFrame, 1, scaleFrame))
    * (osg::Matrix::rotate(rotFrame, osg::Vec3(0, 0, 1)))
    * (osg::Matrix::translate(osg::Vec3(x, y, 125))
    )
    root->addChild(frameTransform[j].get());
frameTransform[j]->addChild(frameNode);
    //cout << "Frame " << j << " hung horizontal" << endl;
    frames_locationInfo[j].position = osg::Vec3f(x, y, 125);
    frames_locationInfo[j].rotation = rotFrame;
    frames_locationInfo[j].scaleFactor = scaleFrame;
    frames_locationInfo[j].withPhoto = false;
    j += 1;
}
/*}
for (int k = 0; k < 21; k++)
{
    cout << k << endl;
    cout << "position " << frames_locationInfo[k].position.x() << " " << frames_locationInfo[k].position.y() << " " << frames_locationInfo[k].position.z() << endl;
    cout << "rotation " << frames_locationInfo[k].rotation << endl;
    cout << "scale " << frames_locationInfo[k].scaleFactor << endl;
    cout << "with photo " << frames_locationInfo[j].withPhoto << endl;
}*/
loadImagesandAssignTextures

Function which loads the photographs to be used in the application

It created textures with them and attach them to each node.

void loadImagesandAssignTextures()
{
    int photoNumber=0;
    for (int i=0; i<23; i++)
        {
            osg::ref_ptr<osg::Texture2D> photoTexture[23];
            osg::ref_ptr<osg::StateSet> photoState[23];
            string imageRoute_0;
            string imageRoute_1;
            char imageNumber[2];
            imageRoute_0 = "Images/image_";
            imageRoute_1 = ".jpg";
            sprintf(imageNumber, "%d", (photoNumber));
            photoImage[i] = osgDB::readImageFile (imageRoute_0+imageNumber+imageRoute_1);
            if (!photoImage[i])
                {
                    cout << "Warning: could not open files." <<endl;
                    getchar();
                }
            photoNumber +=1;
            photoState[i] = new osg::StateSet;
            photoState[i] = photoNode[i]->getOrCreateStateSet();
            photoTexture[i] = new osg::Texture2D;
            photoTexture[i]->setImage(photoImage[i].get());
            photoState[i]->setTextureAttributeAndModes(0, photoTexture[i].get());
        }
}

void createPhotoAnimations()
{
    int p=0;
    int j=0;
    osg::ref_ptr<osg::AnimationPath> photoPath[3][8];
    osg::ref_ptr<osg::AnimationPath> opp_photoPath[3][8];
    for (int i=0; i<3; i++)
    {
        for (j=0; j<8; j++)
        {
            if ((j==2) && (j==3))
                else
            {
                photoPath[i][j] = new osg::AnimationPath;
                photoPath[i][j]->setLoopMode(osg::AnimationPath::NO_LOOPING);
                opp_photoPath[i][j] = new osg::AnimationPath;
                opp_photoPath[i][j]->setLoopMode(osg::AnimationPath::NO_LOOPING);

                // control point 0 : photos inside the box
                osg::AnimationPath::ControlPoint controlPoint_0
                    (osg::Vec3f(0, 0.05*p, 0.01*p),
                     osg::Quat(-osg::PI, 0, 1, 0),
                     osg::Vec3f(0.0175, 1, 0.0175));

                // control point 1 : photos outside the box
                osg::AnimationPath::ControlPoint controlPoint_1
                    (osg::Vec3f(0-6.7*(3-i), -3, 15-5.5*i),
                     osg::Quat(0, 0, 1, 0),
                     osg::Vec3f(0.035, 1, 0.035));

                // animation photos going out of the box
                photoPath[i][j]->insert(0.0, controlPoint_0);
                photoPath[i][j]->insert(2.0, controlPoint_1);

                // animation photos going inside the box
                opp_photoPath[i][j]->insert(0.0, controlPoint_1);
                opp_photoPath[i][j]->insert(2.0, controlPoint_0);

                photoAnimCallBack [i][j] =
                    new osg::AnimationPathCallback( photoPath[i][j].get() );
                opp_photoAnimCallBack [i][j] =
                    new osg::AnimationPathCallback( opp_photoPath[i][j].get() );

                photoTransform[p]->setUpdateCallback(photoAnimCallBack[i][j].get() );
                photoAnimCallBack[i][j]->setPause(true);
            }
            p+1;
        }
    }
}
void handTrackingCalibration()
{
    // get measures for hand tracking calibration
    cout << endl << endl << "Press enter to start calibration." << endl;
    getchar();

    cout << "Measure 1 - RIGHT: press enter when you are ready";
    getChar();
    markerXmax = markerX;
    cout << "marker x max " << markerXmax << endl;
    cout << "marker y min " << markerYmin << endl;

    cout << "Measure 2 - LEFT: press enter when you are ready";
    getChar();
    markerXmin = markerX;
    cout << "marker x min " << markerXmin << endl;

    cout << "Measure 3 - UP: press enter when you are ready";
    getChar();
    markerZmax = markerZ;
    cout << "marker z max " << markerZmax << endl;

    cout << "Measure 4 - DOWN: press enter when you are ready";
    getChar();
    markerZmin = markerZ;
    cout << "marker z min " << markerZmin << endl;

    cout << "Measure 5 - FRONT: press enter when you are ready";
    getChar();
    markerYmax = markerY;
    cout << "marker y max " << markerYmax << endl;

    cout << "All measurements done." << endl;

    xfactor = (1/(fabs(markerXmax-markerXmin)))*26.5;
    zfactor = (1/(fabs(markerZmax-markerZmin)))*20;
    yfactor = (1/(fabs(markerYmax-markerYmin)))*3;

    //getchar();
}
void bounceBack()
{
  if (lastKeyPressed == 'w') {
    myX = myX - 4 * sin(myR);
    myY = myY - 4 * cos(myR);
  }
  if (lastKeyPressed == 'x') {
    myX = myX + 4 * sin(myR);
    myY = myY + 4 * cos(myR);
  }
  if (lastKeyPressed == 'e') {
    myX = myX - 4 * cos(myR);
    myY = myY + 4 * sin(myR);
  }
  if (lastKeyPressed == 'q') {
    myX = myX + 4 * cos(myR);
    myY = myY - 4 * sin(myR);
  }
}
getPointerInfo

Function which applies tracking data to the pointer position. It does it by transforming the tracking data to world coordinates. It also limits the movement of the pointer so it does not go out of the screen.

It checks if pointer is in selection position, and if it is there for a little while then it is considered selected.

It returns whether the pointer is in selected mode or not.

bool getPointerInfo()
{
    bool pressed;

    // apply tracking data to sphere point
    sphereX = -13 + (markerX-markerXmin) * xfactor;
    if (sphereX < -13) sphereX=-13;  // limit so it does not go out of the screen
    else if (sphereX > 13.5) sphereX=13.5;

    sphereZ = (markerZ-markerZmin) * zfactor;
    if (sphereZ < 0) sphereZ=0;  // limits
    else if (sphereZ > 20) sphereZ=20;

    sphereY = (markerY-markerYmin) * yfactor;  // (0-3)

    if (sphereY > 2.5)
        { sphY_t +=0.1;
          if (sphY_t>4)
              { pressed = true;
                // cout<< "sphere down"<<endl;
                sphY_t = 0;
              }
        } else
        { pressed = false;
          sphY_t=0;
        }
    return pressed;
}
// Function which returns the number of the selected photo taking as input the transform node of the pointer

int selectPhoto(osg::MatrixTransform *sphereTransform)
{
    float mysphereX, mysphereZ;
    float photoX, photoZ;
    int selected=23;
    float photoSizeX=5.6; // 160*0.035 (scale)
    float photoSizeZ=4.2; // 120*0.035
    mysphereX = sphereTransform->getMatrix().getTrans().x();
    mysphereZ = sphereTransform->getMatrix().getTrans().z();
    if (spherePressed)
    {
        for (int i=0; i<23; i++)
        {
            if (photoSelection[i])
            {
                photoX = photoTransform[i]->getMatrix().getTrans().x();
                photoZ = photoTransform[i]->getMatrix().getTrans().z();
                if ( ((mysphereX>photoX-photoSizeX/2) && (mysphereX<photoX+photoSizeX/2))
                    && ((mysphereZ>photoZ-photoSizeZ/2) && (mysphereZ<photoZ+photoSizeZ/2))
                {
                    selected=i;
                    //cout << "photo number " << i << " selected" << endl;
                    spherePressed = false;
                }
            }
        }
    }
    return selected;
}

int findClosestFrame(osg::Vec3f camPos)
{
    float min = 1000;
    int frameIndex=22;
    int myRotat;
    for (int i=0; i<21; i++)
    {
        myRotat = frames_locationInfo[i].rotation;
        if ((myR>-margin) && (myR<margin)) // around 0 - horizontal wall
            { 
                if (myRotat==0)
                {
                if (fabs(frames_locationInfo[i].position.x()-camPos.x())<min)
                    
                    min = fabs(frames_locationInfo[i].position.x()-camPos.x()); 
                    frameIndex = i;
                }
        }
        else if ((myR>PI-margin) && (myR<PI+margin)) // around PI - horizontal wall
            { 
                if (myRotat==3)
                {
                if (fabs(frames_locationInfo[i].position.y()-camPos.y())<min)
                    
                    min = fabs(frames_locationInfo[i].position.y()-camPos.y()); 
                    frameIndex = i;
                }
        }
        else if ((camPos.y()-frames_locationInfo[i].position.y())<min)
            { 
                if (camPos.y()-frames_locationInfo[i].position.y())<350)
                    
                    min = fabs(camPos.y()-frames_locationInfo[i].position.y()); 
                    frameIndex = i;
                }
        }
        else if ((frames_locationInfo[i].position.x()-camPos.x())<min)
            { 
                if ((frames_locationInfo[i].position.x()-camPos.x())<min)
                    
                    min = fabs(frames_locationInfo[i].position.x()-camPos.x()); 
                    frameIndex = i;
                }
        }
    }
    // frame is in front of camera
    // around PI/2 - vertical wall (at the right looking to the left)
    else if ((myR>0.5*PI-margin) && (myR<0.5*PI+margin))
        { 
            if (myRotat==4)
            {
                if (frames_locationInfo[i].position.x()-camPos.x())>0 )
                {


// frame is in front of camera
if ((frames_locationInfo[i].position.x()-camPos.x()) < 350 )
    {
        if ((fabs(frames_locationInfo[i].position.y()-camPos.y()))<min)
            {min = fabs(camPos.y()-frames_locationInfo[i].position.y());
                frameIndex = i; }
    }

else {
    if ((myR>1.5*PI-margin) && (myR<1.5*PI+margin))
        {if (myRotat==1)
            {
                if ((camPos.x()-frames_locationInfo[i].position.x()) > 0 )
                    {
                        // frame is in front of camera
                        if (camPos.x()-frames_locationInfo[i].position.x())<350)
                            {if ((fabs(frames_locationInfo[i].position.y()-camPos.y()))<min)
                                {min = fabs(camPos.y()-frames_locationInfo[i].position.y());
                                    frameIndex = i; }
                            }
            }
        }
    return frameIndex; }
void trackingStart()
{
    WORD wVersionRequested = MAKEWORD(2, 2);
    WSADATA wsaData;

    // Initialize WinSock and check version
    int nRet = WSAStartup(wVersionRequested, &wsaData);
    if (nRet != 0)
    {
        PrintError("Couldn't load winsock library (ws2_32.dll)");
        return;
    }
    if (wsaData.wVersion != wVersionRequested)
    {
        PrintError("Wrong version of winsock (ws2_32.dll)");
        return;
    }

    // By default assume you want to connect to QTM at the same machine - just for testing
    //CStdString tServer = "localhost";
    CStdString tServer = "137.195.27.252";

    // Example showing how to read data frame by frame
    // Example2(tServer, 22222);

    // Release WinSock
    //WSACleanup();
    return;
}

void Example2(CStdString tServer, int nPort)
{
    // Connect to the server
    SOCKET hSocket = Connect(tServer, nPort);
    if (hSocket == INVALID_SOCKET)
    {
        return;
    }

    SHeader sHeader;
    CStdString tData;
    CStdString tCmd;

    // Welcome message
    //if(RecvMessage(hSocket, sHeader, tData))
    {
        puts(tData);
// Set protocol version
Set protocol version
//
tCmd = "Version 1.0";
SendCommand(hSocket, tCmd);
if(RecvMessage(hSocket, sHeader, tData))
{
    puts(tData);
}

// Set byte order to machine native ordering for easier processing
//
tCmd = "ByteOrder LittleEndian";
SendCommand(hSocket, tCmd);
if(RecvMessage(hSocket, sHeader, tData))
{
    puts(tData);
}

// Request capture parameters
//
std::vector<CStdStringA> atLabels;
tCmd = "SendParameters All"; //"SendParameters 6dEuler";
SendCommand(hSocket, tCmd);
if(RecvMessage(hSocket, sHeader, tData))
{
    if (sHeader.nType == 2)
    {
        NBC_CMarkup oXML;
oXML.SetDoc(tData);
        // Read 6d marker labels
        //
        if (oXML.FindChildElem("The_6D") && oXML.IntoElem() &&
oXML.FindChildElem("Bodies")
        {
            int nLabels = atoi(oXML.GetChildData());
            printf("There are %d labeled bodies\n", nLabels);
atLabels.resize(nLabels);
            for (int iLabel=0; iLabel<nLabels; iLabel++)
            {
                if (oXML.FindChildElem("Body")
                {
                    oXML.IntoElem();
                    if (oXML.FindChildElem("Name"))
                    {
                        atLabels.at(iLabel) = oXML.GetChildData();
                        printf("Body %d: %s\n", iLabel+1, atLabels.at(iLabel).c_str());
                    }
                    oXML.OutOfElem();
                }
            }
        }
        else
        {
            // Error
            //
            puts(tData);
        }
    }
    //cout << tData;
    //getchar();
}
Variables for timing test

int nReceivedFrames = 0;
LARGE_INTEGER nBefore;
QueryPerformanceCounter(&nBefore);

// Retrieve one data frame

char* tCmd = "SendCurrentFrame All";  // <--"SendCurrentFrame 6dEuler"
//tCmd = "StreamFrames FrequencyDivisor:1 3dNoLabels";  //?????
SendCommand(hSocket, tCmd);
bool bHasData = true;
int nLastFrame = 0;
while (bHasData && RecvMessage(hSocket, sHeader, tData))
{
  switch(sHeader.nType)
  {
    case 0:
      // sHeader.nType 0 indicates an error
      //
      fprintf(stderr, "Error at SendCurrentFrame: %s\n", tData);
      bHasData = false;
      break;
    case 1:
      //
      // sHeader.nType 1 indicates successful completion of an operation and/or a string message
      //
      fprintf(stderr, "Message: %s\n", tData);
      break;
    case 3:
      //
      // sHeader.nType 3 means data
      //
      nReceivedFrames++;
      //
      // Do some processing here
      //
      //PrintFrameNumber(sHeader, tData, nReceivedFrames, nLastFrame);
      PrintData(sHeader, tData, atLabels);
      break;
    case 4:
      //
      // sHeader.nType 4 means no more data
      //
      bHasData = false;
      break;
    default:
      break;
  }
  if (bHasData)
  {
    //
    // Retrieve next data frame if last frame was ok.
    //
    SendCommand(hSocket, tCmd);
  }
}
// Disconnect from the server
std::cout << "This is the end ...
getchar();
Disconnect(hSocket);

```c
// Print data to console
void PrintData(SHeader sHeader, CStdString tData, std::vector< CStdString > atLabels)
{
    char *pData = const_cast<char*>(tData.c_str());
    SPacketHeader* pHeader = reinterpret_cast< SPacketHeader* >(pData);
    char* pComponent = pData + sizeof(SPacketHeader);
    //float fTime = pHeader->nTimeStamp*0.000001;
    //printf("Time \%3.2f\tfFrame \%d\n", fTime, pHeader->nFrameNumber);

    for (int iComponent=0; iComponent<pHeader->nComponentCount; iComponent++)
    {
        //S3dComponent* p3d = reinterpret_cast< S3dComponent* >(pComponent);
        S3dNoLabelsComponent* p3dNoLabels = reinterpret_cast< S3dNoLabelsComponent* >(pComponent);
        S6dEulerComponent* p6dEuler = reinterpret_cast< S6dEulerComponent* >(pComponent);

        if (p6dEuler->nType==cn6dEuler)
        {
            for (int i = 0; i < p6dEuler->nCount; i++)
            {
                // if the labeled body is my body (a_glasses)
                if (strcmp(atLabels.at(i).c_str(),"a_glasses")==0)
                {
                    if (trackHead)
                    {
                        if (typeid(p6dEuler->aSamples[i].fRl) == typeid(myHR))
                        {
                            // if there is a significant change with the previous position, update
                            if ((abs (p6dEuler->aSamples[i].fRl - myHR)) > 0.25)
                            {
                                myHR = p6dEuler->aSamples[i].fRl;
                                // limit rotation between +45 and -45 degrees
                                if (myHR>45) myHR=45;
                                else if (myHR<-45) myHR=-45;
                            }
                        }
                        else if (typeid(p6dEuler->aSamples[i].fR3) == typeid(myHR2))
                        {
                            // if there is a significant change with the previous position, update
                            if (abs(p6dEuler->aSamples[i].fR3 - myHR2) > 0.25)
                            {
                                myHR2 = p6dEuler->aSamples[i].fR3;
                                // limit rotation between +45 and -45 degrees
                                if (myHR2>45) myHR2=45;
                                else if (myHR2<-45) myHR2=-45;
                            }
                        }
                    }
                    else if(p3dNoLabels->nType==cn3dNoLabels)
                    {
                        // hand tracking is sent as 3D data
                        // I just need the position of the marker
                        for (int i = 0; i < 1; i++)
                        {
                            markerX = p3dNoLabels->as3dPoints[i].fx;
                            markerY = p3dNoLabels->as3dPoints[i].fy;
                            markerZ = p3dNoLabels->as3dPoints[i].fz;
                        }
                    }
                }
            }
        }
    }
}```
else {
  //cout<< "no right data received" <<endl;
}

pComponent += p6dEuler->nSize;

} // Receive a packet. Data is stored in a CStdString, so allocation and deallocation is made fairly easy

// Receive start of a new packet from the client. The first 8 bytes holds the header.

nRecved = recv(hSocket, &uHeader.c+nHeaderRecved, sizeof(rsHeader)-nHeaderRecved, 0);
if (nRecved == SOCKET_ERROR) {
  PrintError("Error at recv()");
  return false;
}
if (nRecved == 0) {
  // The server closed the connection
  return false;
}

nHeaderRecved += nRecved;

// Convert the header from network byte order to host byte order

rsHeader.nSize = htonl(uHeader.s.nSize) - sizeof(rsHeader);
rsHeader.nType = htonl(uHeader.s.nType);

// Allocate memory to hold the entire packet

ASSERT(rsHeader.nSize < 20000000);
rtData.resize(rsHeader.nSize);

// Receive more data until we have read the whole packet

while (nBodyRecved < rsHeader.nSize) {
  // As long as we haven't received enough data, wait for more
  nRecved = recv(hSocket, &{rtData[nBodyRecved]}, rsHeader.nSize-nBodyRecved, 0);
if (nRecved == SOCKET_ERROR || nRecved == 0)
{
    PrintError("Error at recv()");
    return false;
}

nBodyRecved += nRecved;

bool SendCommand(SOCKET hSocket, CStdString tCommand)
{
    if (tCommand.size() > BUFFER_SIZE)
    {
        fprintf(stderr, "Command string too long");
        return false;
    }

    USendBuffer uSendBuffer;

    // Commands are string which means type is 1
    // Header size + length of the string + terminating null char
    int nSize = sizeof(SHeader) + tCommand.size() + 1;
    uSendBuffer.s.nSize = htonl(nSize);
    memcpy(uSendBuffer.c + sizeof(SHeader), tCommand.c_str(), tCommand.size() + 1);

    int nSent = 0;
    int nTotSent = 0;
    while (nTotSent < nSize)
    {
        nSent = send(hSocket, uSendBuffer.c + nTotSent, nSize - nTotSent, 0);
        if (nSent == SOCKET_ERROR)
        {
            PrintError("Error at send()");
            return false;
        }

        nTotSent += nSent;
    }

    // Successfully sent the command
    return true;
}

SOCKET Connect(CStdString tServer, int nPort)
{
    SOCKET hSocket = socket(AF_INET, SOCK_STREAM, 0);

    sockaddr_in sAddr;

    // First check if the address is a dotted number "A.B.C.D"
    sockaddr.sin_addr.s_addr = inet_addr(tServer.c_str());
    if (sAddr.sin_addr.s_addr == INADDR_NONE)
    {
        // If it wasn't a dotted number lookup the server name

/ *psHost = gethostbyname(tServer.c_str());
  if (!psHost)
  {
    PrintError("Error looking up host name");
    closesocket(hSocket);
    return INVALID_SOCKET;
  }
  sAddr.sin_addr = *((in_addr*)psHost->h_addr_list[0]);
} sAddr.sin_port = htons(nPort);
sAddr.sin_family = AF_INET;

if (connect(hSocket, (sockaddr*)(&sAddr), sizeof(sAddr)) == SOCKET_ERROR)
{
  PrintError("Error at connect()");
  closesocket(hSocket);
  return INVALID_SOCKET;
}

// Disable Nagle's algorithm

char bNoDelay = 1;
if (setsockopt(hSocket, IPPROTO_TCP, TCP_NODELAY, &bNoDelay, sizeof(bNoDelay)))
{
  PrintError("Error at connect()");
  closesocket(hSocket);
  return INVALID_SOCKET;
}

// Successfully connected to the server

return hSocket;

void Disconnect(SOCKET hSocket)
{
  // Try to shutdown gracefully
  shutdown(hSocket, SD_SEND);
  int nRecved = 1;
  char pData[500];
  while (nRecved > 0)
  {
    // There shouldn't be anything left to receive now, but check just to make sure
    nRecved = recv(hSocket, pData, sizeof(pData), 0);
  }
  closesocket(hSocket);
}

void PrintError(char *t)
{
  char *tError = NULL;
  DWORD nError = GetLastError();
  DWORD nRet = FormatMessage(
    FORMAT_MESSAGE_ALLOCATE_BUFFER | FORMAT_MESSAGE_FROM_SYSTEM,
    NULL, nError, 0, reinterpret_cast<LPTSTR>(&tError), 0, NULL);
  fprintf(stderr,"%s: %d %s\n", t, nError, tError);
  LocalFree(tError);
}