CONSTRUCTION OF A VIRTUAL REALITY ENVIRONMENT BASED ON POSITION TRACKING COMBINED WITH A HEAD-MOUNTED DISPLAY

Bachelor's Thesis

to obtain the academic degree Bachelor of Science submitted
to the Faculty Physics, Mathematics and Computer Science
of the Johannes Gutenberg University Mainz

22/09/2014

by
Michael Aloys Hedderich

Submission date: 22/09/2014

First reviewer: Prof. Dr. Elmar Schömer
Algorithmic Geometry and Computer Graphics
Johannes Gutenberg University Mainz

Second reviewer: Prof. Dr. Ulrich Schwanecke
Computer Vision and Mixed Reality
University of Applied Sciences RheinMain

Erklärung

Hiermit versichere ich, dass ich die vorliegende Arbeit selbstständig und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt habe. Die aus anderen Quellen oder indirekt übernommenen Daten und Konzepte sind unter Angabe der Quelle gekennzeichnet. Die Arbeit wurde bisher weder im In- noch Ausland in gleicher oder ähnlicher Form in einem Verfahren zur Erlangung eines akademischen Grades vorgelegt.

Declaration

I hereby declare that I have written the present thesis on my own and without use of other than the indicated means. I also declare that to the best of my knowledge all passages taken from published and unpublished sources have been referenced. The paper has not been submitted for evaluation to any other examining authority nor has it been published in any form whatsoever.

Mainz, 22/09/2014

Michael Aloys Hedderich

ABSTRACT

Virtual reality offers fascinating possibilities to experience computer generated worlds. We present a virtual reality system that enables the user to naturally walk through the virtual environment. He or she can crouch, jump, turn around, look upwards and downwards, etc. The user wears an active marker which is tracked by a camera and a special tracking software. The information about its position and rotation is sent to a head-mounted device which has a smartphone as a display. The smartphone renders a stereoscopic scene from the user's perspective. We developed a first prototype that demonstrates the feasibility of our concept. Our implementation aims at museums or exhibitions that want to bring virtual worlds to life. Nevertheless, this technique can be extended to a variety of applications from scientific visualization to augmented reality. In this thesis we explain all the parts in detail. As background a short description of the theory of stereoscopic projection is given. We conclude with an extensive evaluation explaining the problems that still exist and showing promising improvements for future work.

GERMAN ABSTRACT

Virtuelle Realitäten eröffnen spannende Möglichkeiten, um computergenerierte Welten zu erleben. Wir präsentieren ein System, das es dem Benutzer ermöglicht, sich auf natürliche Art und Weise durch virtuelle Umgebungen zu bewegen. Er oder sie kann springen, sich umdrehen, nach oben und unten schauen, etc. Der Nutzer trägt einen aktiven Marker, der von einer Kamera und einer speziellen Software beobachtet und analysiert wird. Die so erhaltenen Informationen über seine Position und Rotation werden an ein „head-mounted display“ geschickt. Dieses besteht aus einer am Kopf getragenen Halterung und einem Smartphone als Bildschirm. Das Smartphone berechnet dann ein stereoskopisches Bild der Szene aus der Perspektive des Benutzers. Wir haben einen ersten Prototypen entwickelt, der die Machbarkeit unseres Konzeptes demonstriert. Unsere Umsetzung richtet sich an Museen oder Ausstellungen, die virtuelle Welten zum Leben erwecken wollen. Diese Technik lässt sich aber auch für eine Vielzahl anderer Anwendungen einsetzen von der wissenschaftlichen Visualisierung bis hin zur erweiterten Realität. Im Folgenden wird detailliert auf alle Bestandteile des Systems eingegangen. Zum besseren Verständnis wird ein kurzer Abriss über die Theorie der stereoskopischen Projektion gegeben. Den Abschluss bildet eine ausführliche Auswertung, bei der wir auf vorhandene Probleme eingehen und Verbesserungsmöglichkeiten für die Zukunft aufzeigen.

ACKNOWLEDGEMENTS

First and foremost, I want to thank Prof. Dr. Elmar Schömer and Prof. Dr. Ulrich Schwanecke who supervised my thesis and helped me to finish it in time. Henning Tjaden explained me the use of the tracking technique and helped me to optimize the marker.

A special thanks to the Algorithmic Geometry and Computer Graphics group at the Johannes Gutenberg University Mainz, the Computer Vision and Mixed Reality group at the University of Applied Sciences RheinMain and Durovis Dive who supported me with technical equipment. Cyberith squeezed me in their full time schedule allowing me to test the Virtualizer.

Lena, Josh, Jonas and my family gave me moral support and proofread the text. Thank you! All mistakes that still remain in the text are mine.

CONTENTS

1 INTRODUCTION 2

2 TRACKING 4

3 MARKER 8
3.1 LEDs . . . 8
3.2 Circuit . . . 9

4 NETWORK 12
4.1 Network Connection . . . 12
4.2 Discovery Phase . . . 13
4.3 Transmission Phase . . . 14

5 DISPLAY 15
5.1 Display Device . . . 15
5.2 Software . . . 17

6 STEREOSCOPY 19
6.1 Definition . . . 19
6.2 Theoretical Calculation of the Projection . . . 20
6.3 Implementation With OpenGL . . . 24

7 EVALUATION AND FUTURE WORK 30
7.1 Marker . . . 30
7.2 Tracking . . . 30
7.3 Network . . . 32
7.4 Display . . . 33
7.5 Simulator Sickness . . . 33
7.6 Comparison to the Cyberith Virtualizer . . . 34
7.7 Conclusion . . . 35

A APPENDIX 37
A.1 CD . . . 37
A.2 Programs . . . 37
A.3 Compiling . . . 38

REFERENCES 39

1 Introduction

In this work we want to present a virtual reality environment based on position tracking combined with a head-mounted display. While definitions of virtual reality (VR) vary between a narrow focus on the technical devices used and a more general focus on the user's experience[1], most authors describe it as the physical immersion into a virtual world that allows interaction and gives sensory feedback[2, pp. 6-13.]. Lately this topic has been of increased interest, especially because of the efforts of the Oculus Rift developers to create a consumer version of a virtual reality device[3]. This trend is supported by competing companies that announced their own development of such devices, e.g. Sony[4] and Microsoft[5].

Most of these VR systems limit the physical interaction to the user's head movement. The orientation of the head is tracked using gyroscopes or positional trackers allowing the change of the camera's orientation in the virtual environment. All other interactions with the virtual world have to be done with classical input devices, e.g. keyboards or game controllers. As Slater et al. argue[6], the physical activity of walking increases the subjective presence of the user in the virtual world. This means that the user perceives the environment more intensely if he or she can really walk through the virtual world or the video game level in the same way as if he or she walks in the real world. Several projects try to achieve this by constructing an omnidirectional treadmill that measures the user's movement (e.g. Virtuix Omni[7] and Cyberith Virtualizer[8]). The advantage of these systems is that the physical space can be much smaller than the virtual space since the user only walks on the same spot. However, these constructions are relatively complex and expensive and they still limit the user in its natural movement since the user has to be fixed in the construction.

We therefore want to present a virtual reality system that allows the user to walk through the virtual environment without any other physical limitations than the room he or she walks in. The user's pose is tracked using an active infrared marker and a camera filming the room (see figure 1.1). This information is then sent to a Durovis Dive, a head-mounted device working with a smartphone as a display, therefore not needing any cables that limit the user's walk. A user is thus able to walk through a virtual room by walking through a real room. This system could be applied in museums and exhibitions allowing to walk through virtual reconstructions and to gain a more vivid experience of e.g. a Roman street or a

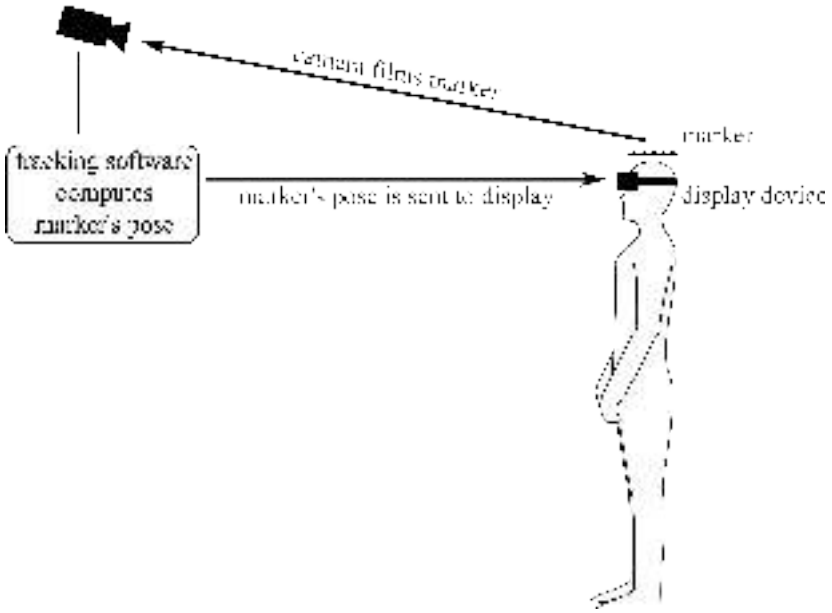


Figure 1.1: An overview of the system's parts.

medieval hall. It could also be combined with augmented reality applications or used as a physical extension to video games.

In this text we present every part of the system: the optical pose tracking, the tracking device/marker, the data transfer from the tracking software to the smartphone, the display device and the stereoscopic rendering. All parts are going to be explained in detail, also explaining the choice for this particular implementation and possible alternatives. The chapter about stereoscopy is extended by a theoretical background. We conclude with an evaluation of the system, a comparison to the Cyberith Virtualizer and possible improvements for future work.

2 Tracking

A key element to a virtual reality environment is the knowledge of the pose of the user's head, i.e. its position and rotation. This information is later used to generate an image of the virtual world from the user's point of view. Through the pose, the user's perspective in the virtual room should be recorded as exact and up-to-date as possible. A difference between the physical pose and the pose in the virtual world can cause a decline in immersion and even nausea (see chapter 7.5). This would be the case e.g. if the user steps to the left and the movement is only translated to the virtual reality with a certain delay.

We used a tracking system developed by Tjaden et al.[9]. It consists of three parts: a tracking device called marker, a camera and the software that calculates the position of the marker based on the images created by the camera.

The marker is fastened on the user's head. It consists of a circuit board with seven infrared light-emitting diodes (IR-LEDs). They are positioned in a cross shape with one light elevated above the others (see figure 3.4). Their distance to each other can be chosen arbitrarily as long as the cross form is kept intact. However, the exact distances of the LEDs must be known for calibrating the tracking software. To reduce measuring errors only one distance is measured by hand. All other distances can be calculated automatically using a tool that is part of the tracking software[9]. The construction of the marker is explained in chapter 3.

The camera is put on the edge of the room on an elevated position. It is a black-and-white, high-frequency USB 3 camera (Point Grey FL3-U3-13Y3M-C, figure 2.1)

Figure 2.1: Point Grey FL3-U3-13Y3M-C camera.

Figure 2.2: One of many calibration images to calculate the intrinsic parameters of the camera.

with 1280 x 1024 pixels at 150 frames per second[10]. A special C++ library, that is shipped with the camera, has to be used to access and capture the camera's images.

The tracking software requires the camera's intrinsic parameters (focal length, principal point and lens distortion). We measured these using the GML C++ Camera Calibration Toolbox[11]. This program computes the intrinsic parameters based on photos of chessboard patterns of known sizes (see figure 2.2). To minimize the measuring error, we took over 250 images of a composition of two patterns from a variety of perspectives.

The infrared LEDs appear as white dots on the camera's image (see figure 2.3a). Only these dots are relevant for the tracking process. The tracking performance is highly dependent on the light blobs' condition. The tracking software uses a binary threshold keeping only pixels with a high intensity. If the light blobs are not bright enough, the software might mistake them for noise and remove them (see figure 2.3b). Then again, if the lights are too bright, they show up in a star form that is not recognized either (see figure 2.3c). The blobs' condition depends among other factors on the distance between the camera and the marker and on the type of LEDs used. It can be controlled via the camera by opening or closing the diaphragm and by increasing or reducing the exposure time. If the diaphragm aperture is very small, the interference of other light sources and ambient light diminishes. The same effect can be reached using an infrared filter which reduces the passage of non-infrared light. The advantage of this approach is that the diaphragm can be opened wider (since less other light interferes) and the marker becomes visible at larger distances. The exposure time can be controlled directly using the camera's C++ library. In a future step, this value could be set by the software. This would also allow to adjust it automatically; increasing it when the user moves away from

(a) (b) (c)

Figure 2.3: Camera images of the marker at a distance of 0.5 m without infrared filter using an exposure time of 2 ms (a), 0.5 ms (b) and 6 ms (c). If the tracking software detects a marker, it automatically adds the digits corresponding to the LEDs. Only the conditions in (a) allow the successful detection.

the camera (and thus the lights get darker) and reducing it when he or she comes closer again, avoiding an overlap of light blobs.

The tracking software was provided as a C++ library. First it has to be configured with the characteristics of the camera (intrinsic parameters) and of the marker (position of the LEDs). It then accepts an image and returns the marker's pose in the real world. The pose P is encoded as a matrix in the form of

$$P = \begin{pmatrix} R & t \\ 0 & 1 \end{pmatrix} \tag{1}$$

with $R \in \mathbb{R}^{3x3}$ being the rotation matrix and $t \in \mathbb{R}^3$ being the translation vector.

We wrote a C++ program that connects the camera with the tracking library. Since the resulting pose is relative to the (unknown) pose of the camera, a correspondence between the physical and the virtual world is needed. The marker is therefore placed on a position that represents the starting point or origin in the physical world. By pressing the key 'o' in our program, the current pose is stored as origin O and its inverse O^{-1} is computed. Every pose P' is calculated as

$$P' = O^{-1} \cdot P \tag{2}$$

with P being the pose delivered by the tracking software. The origin in the physical world corresponds with the origin in the virtual world. The new pose P' is relative to the defined origin. It can be later multiplied by the projection matrix in the

virtual environment to translate and rotate the camera into the correct position. If the marker is placed for example at the origin ($P = O$), the resulting pose is

$$P' = O^{-1} \cdot P = O^{-1} \cdot O = E \tag{3}$$

and the camera also stays in the origin of the virtual world.

Once the pose is obtained, our program sends the information to the display device. Since the last row of the pose matrix is always the same, only the matrix R and the vector t have to be transmitted (12 values). The accuracy of the values is reduced from double to single precision. This reduces the amount of data that has to be transmitted. The quality will not suffer, as the software that renders the virtual environment only works with single precision values. The rendering software has to rotate the received pose matrix, because the coordinate system of the tracking library is different to the one of the rendering library.

3 Marker

3.1 LEDs

The tracking device or marker is based on the same paper as the tracking software[9]. As mentioned above, it consists of seven infrared light-emitting diodes (IR-LEDs). Since this marker was especially developed for the tracking software, it cannot be bought. Instead we soldered one ourselves. A detailed explanation will follow on how to build such a device starting with the choice of the LEDs.

Miniature LEDs exist in various shapes and sizes and we have experimented with several of them. Most commonly, they have a round or cylindrical shape, a diameter between three and five millimeters and a height between five and ten millimeters (figure 3.1). The polarity can be easily checked since the shorter leg and the flat edge of the casing indicate the position of the cathode (negative lead). It is important to find the correct polarity since the LED only emits light if the current is of the right polarity. Otherwise it will stop the current of the complete path (i.e. work as a diode) or even break.

We first used Osram SFH 409 LEDs. The size of these standard LEDs is big enough to wire them relatively easily into a circuit. Unfortunately, their viewing angle is small (20°). Rotating the tracker only a little, leads to one or several lights that no longer shine into the direction of the camera.

We also tried Harvatek HE3-1100AC LEDs. They have a viewing angle of 100°. The tracker can therefore be rotated without disappearing from the camera view. But

Figure 3.1: Infrared LEDs from left to right: Osram SFH 409, Harvatek HE3-1100AC and OptoSupply OSI3NAS1C1A (SMD).

Forward Current (I_F)	0.1 A
Forward Voltage (U_F)	1.6 V
Peak Wavelength	850 nm
50% Power Angle	120°

Table 3.1: Electrical and optical characteristics of the IR-LED OptoSupply OSI3NAS1C1A[12] .

each of these LEDs emits several light dots. This makes them useless as markers since the tracking software can only handle one light dot per LED. The other disadvantage of these LEDs is their height. Rotating the tracker leads to some diodes occluding others. The software cannot work with this either.

In the end, we worked with surface-mount LEDs (SMD-LEDs, see figure 3.1). They are flat and often used in industrial production. Their small height reduces the problem of occlusion. Their big disadvantage for "amateur users" is their small size and the missing legs or leads. This makes the soldering process very difficult. We tried two different products and in the end, we used OptoSupply OSI3NAS1C1A. This is the same product the authors of the tracking software used. The characteristics of this product can be seen in table 3.1.

3.2 Circuit

The two basic forms of connecting elements of an electrical circuit are series and parallel. In a series circuit, all components are placed along one closed path (see figure 3.2a). The voltage needed is the sum of the needed voltages of all components. The forward voltage U_F indicates the voltage a LED needs to shine correctly. Lowering the voltage would lower the brightness of the light. Increasing it would increase the brightness, but it would also shorten the life expectancy of the LED or could even break it immediately.

In our case U_F equals 1.6V. If we connect all 7 LEDs in a series circuit, we would need $7 * 1.6V = 11.2V$. A normal 9V battery would not be able to power this circuit.

Another possibility is to create a parallel circuit by placing each component on its own path parallel to the others (see figure 3.2b). If all components have the same characteristics, the necessary voltage is the same on each path. We would only need 1.6V to power the circuit. But there does not exist an ordinary battery that delivers this exact voltage. The solution is to place a resistor on each path to lower the voltage.

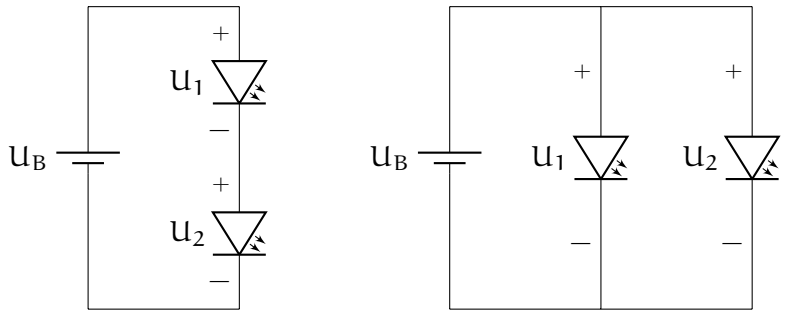


Figure 3.2: Two circuits with a battery (U_B) and two LEDs (U_1, U_2). In the series circuit (a) the voltages must be adjusted so that $U_B = U_1 + U_2$. In the parallel circuit (b) the voltages are $U_B = U_1 = U_2$.

If the supply voltage U_B equals 9V, the necessary resistance R can be calculated as

$$R = \frac{U_B - U_F}{I_F} = \frac{9V - 1.6V}{0.1A} = 74\Omega \tag{4}$$

with I_F as the forward current of the LED. Hence, the circuit could be powered with a 74Ω resistor. But we would have to place a resistor on each path making the circuit more complicated. It is possible to use only one resistor and unite it with all the paths, but this is not recommendable, because the failure of only one component in one path could then cause the destruction of the whole circuit.

We combined both concepts arriving at the final circuit that is illustrated in figure 3.3. It is powered by a 9 V battery and consists of two parallel paths. One path contains three LEDs and the other 4 LEDs. On each path, a resistor is placed. The resistance for each path is calculated similarly as before:

$$R_1 = \frac{9V - 3 \cdot 1.6V}{0.1A} = 42\Omega \tag{5}$$

$$R_2 = \frac{9V - 4 \cdot 1.6V}{0.1A} = 26\Omega \tag{6}$$

Resistors usually have standardized values. The nearest higher resistances that can be bought are 74Ω and 27Ω. Since the difference between the calculated and the actual resistance is not the same, the voltage and therefore the brightness of the lights could be different. This difference is not visible in the actual tracker.

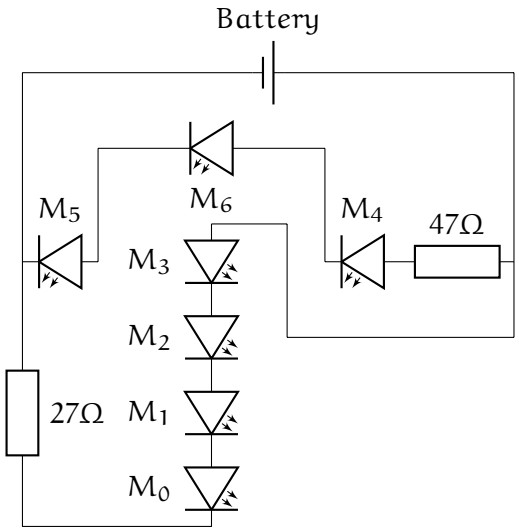
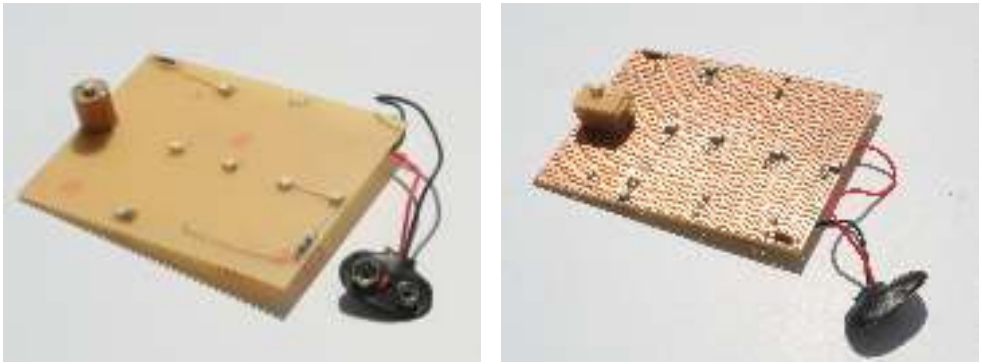


Figure 3.3: The circuit of the marker.



(a) **(b)**

Figure 3.4: Two markers using SMD infrared LEDs.

The total current is the sum of the paths' currents. This raises no problems in this case since each LED and thus each path only needs a current of 100 mA adding up to a current of 200 mA.

We constructed two markers which can be seen in figure 3.4. For marker (a), we used a circuit board fixing the LEDs with wire. Since the wire does not adhere to the board, it is difficult to properly fasten the SMD elements. The marker (b) is built on a stripboard. This type of board has copper strips attached to it. The elements are soldered directly to the strips. This makes it easier to mount the SMD-LEDs accurately.

4 Network

4.1 Network Connection

One fundamental requirement of our system is that the user can move through the room without any physical limitations by the technical equipment. This is achieved by using a head-mounted device that does not need any cables connecting it to a computer (see chapter 5). The pose that is calculated by the camera and the computer has to be transferred wirelessly to the smartphone used in the head-mounted device. Two very common wireless communication techniques in smartphones are Bluetooth and WLAN. Both are suitable for this application and work with the same concept.

The network connection consists of two phases. First the smartphone has to find the computer that runs the tracking software. During this discovery phase, the smartphone contacts all devices in the network (Bluetooth or WLAN) until it finds the correct partner. During the second phase, the two devices create a fixed network connection between each other. The computer is then going to use this connection to send the information about the current pose each time it receives and analyzes a new picture from the camera.

If WLAN in infrastructure mode is used, all devices in a network connect to a wireless access point that takes care of transferring the information between the devices (see figure 4.1a). If the access point is an Ethernet bridge, it is even possible to use a computer that only has a wired Ethernet connection. We chose this communication technique, because, in our experience, this is currently the most widely used one on both smartphones and computers.

WLAN in ad hoc mode as well as Bluetooth have the advantage that they do not need a central access point. Instead, the smartphone and the computer can

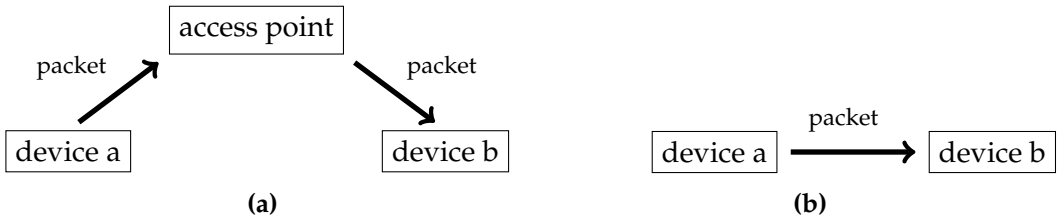


Figure 4.1: Schema of a wireless LAN communication in infrastructure mode (a) and ad hoc mode (b).

communicate directly with each other (peer-to-peer, see figure 4.1b). This can be useful if no access point infrastructure or no AC power plugs are available, e.g. outdoors or at a temporary exhibition site. The tracker, the smartphone and the computer (if a laptop is used) are all equipped with their own battery and the camera is powered by the USB connection to the computer. The access point is the only device that usually needs a power plug. It would not be needed anymore using a peer-to-peer technique.

Since these examples of peer-to-peer communication are special cases, we did not implement this technique. It can be done, however, by just changing the actual soft- and hardware responsible for the network communication. All the other components and also the general communication concept remain the same.

We will now explain the implementation in more detail. As pointed out above the network connection consists of two parts: the discovery and the transmission phase. In both phases, the computer acts as a server and the smartphone as a client. This arrangement would make it possible in a further step to connect several smartphones to one computer and camera.

4.2 Discovery Phase

The aim of the discovery phase is for the client to obtain the server's IP address. This is visualized in figure 4.2. It starts with the server opening a socket for the UDP protocol on port 55003 and listening for incoming packets. When the smartphone application (i.e. the client) is started, it opens a socket on the same port and sends a broadcast message on the same port to all devices in the network. Upon receiving the client's message, the server extracts the sender's address and answers it with a predefined response. The client receives the server's address through its response. The client has now obtained the server's IP address which is stored and used for the second phase. All other devices in the network will probably not answer to the broadcasted message since they usually do not listen to this specific port. Even if the port is used by any other application in the network, the response message will not be the same. We used the UDP protocol for this phase, because it supports broadcast messages which is not the case with TCP.

Instead of using the discovery phase, the server's address could be entered manually in the client application. This might be necessary if the network infrastructure forbids broadcast packets or if the server forms part of another network. Usually, this is not the case. Although the discovery phase is not necessary, it makes the use of the system more comfortable. Otherwise the user would need

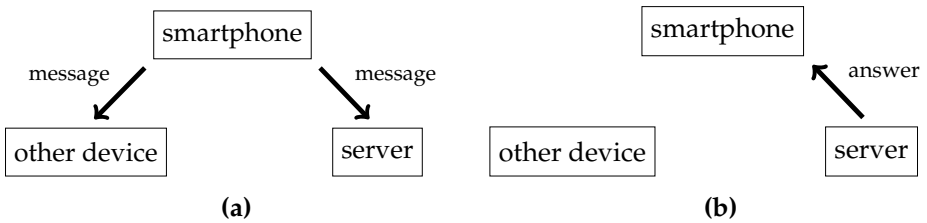


Figure 4.2: Schema of the network discovery phase. The smartphone sends messages to all devices that form part of the network (a). Only the server replies to this message with the correct answer (b).

to check the server's address and then enter it manually into the smartphone's application.

4.3 Transmission Phase

During the transmission phase, the actual information about the user's pose is sent. It starts with the server opening a socket for the TCP protocol on port 55004 and listening for incoming packets. Given that the server's address might have been entered manually into the client (and therefore the server's discovery function would not have been used), the server's transmission phase program starts at the same time as the discovery phase program and runs parallelly.

When the client has obtained the server's IP address (either using the discovery phase or having it entered manually), it starts a TCP connection with the server. Once this connection is established, the server starts sending a pose each time it calculates a new one. For the transmission, the pose is encoded in 12 consecutive float values. Before the floats are sent, their byte order has to be reversed, because the computer usually runs on an little-endian architecture whereas the Java language (on which the Android platform is based) uses a big-endian system[13]. For this phase, we chose the TCP protocol, because it ensures that the segments arrive in the correct order.

5 Display

5.1 Display Device

Once the user's current pose is retrieved, it can be used to render the virtual world from the user's viewpoint. This rendered image must then be displayed to the user. In our opinion, the main criteria that influence the choice of an adequate display device are:

- The user should always see the display independent of his or her position or rotation.
- The device should not limit the user in his or her movement.
- The device should support the immersion into the virtual environment, e.g. using stereoscopic techniques (see chapter 6).
- The device should be relatively inexpensive and easy to obtain.

The first two points are basic requirements for free movement, whereas the third one improves the experience. The fourth one is not necessary from a technical point of view, but important if such an approach should successfully be adopted.

The most widespread display devices are TV or computer screens and projectors. Their severest impediment is that the user cannot turn around without losing sight of the image. This can be solved by projecting images on all walls of the room and - if possible - even on the ceiling and the floor. Certain projectors also support stereoscopy. Such an approach was taken by the developers of the CAVE at the Electronic Visualization Laboratory of the University of Illinois at Chicago[14]. They constructed a theater with three 3 x 3 m walls, three rear-projectors for the walls and one top-down projector for the floor. The generated images are stereoscopic and spread across the walls. The concept was also developed further, e.g. into the CAVE2[15]. The downside of this approach are the very high costs and its complex construction.

An alternative are head-mounted displays. These devices often resemble big goggles. But instead of looking through glass, the user looks onto a screen. They also block all or most of the external light which increases the immersion. Since the display is fixed to the user's head, it is always visible. Usually a separate image is presented to each eye which allows the use of stereoscopic techniques.

(a) (b)

Figure 5.1: The head-mounted device Durovis Dive without display (a) and with an inserted smartphone (b).

The probably most well known head-mounted display on the market is currently the Oculus Rift. Its disadvantage is the cable that connects the device to the computer that renders the images. Although the cable could probably be extended to a certain length, it still limits the user's movement. The user would not be able to turn around without wrapping him- or herself in the cable. To avoid this, a construction would be necessary that guides the cable, e.g. on top of the user. Unfortunately, the development kit was not available for purchase when we started constructing our system, so we were not able to test this possibility.

In the end, we decided to use a Durovis Dive (see figure 5.1). Instead of having a built-in display, this device works as a holder for a smartphone. A lens for each eye increases the image. Since all the rendering can be done on the smartphone, no wired connection is needed. Instead, only the current pose is sent wirelessly and not the rendered images as with the Oculus Rift. This limits of course the graphical complexity to the smartphone's capabilities. Alternatively, the images could be rendered on a high-performance computer and then streamed to the smartphone. This could, however, increase the delay between the tracking and the display (see chapter 7.5).

The Durovis Dive itself is inexpensive (under 50 €) and the models have been released for non-commercial use, so it can even be printed directly on a 3D-printer. The producing company was kind enough to grant us a free device for the purpose of this work. The image quality depends on the smartphone. We used a Nexus 5 with a resolution of 960 x 1080 pixels per eye[16] (which is the same as that of the current developer version of the Oculus Rift[17]).

Google Cardboard[18] is a project similar to the Durovis Dive. Instead of using a 3D-printer, the holder is constructed out of cardboard; the lenses, however, are the same. The disadvantage is that the lenses cannot be adjusted, whereas the Durovis Dive has small handles that allow to adjust the eye distance and the focus.

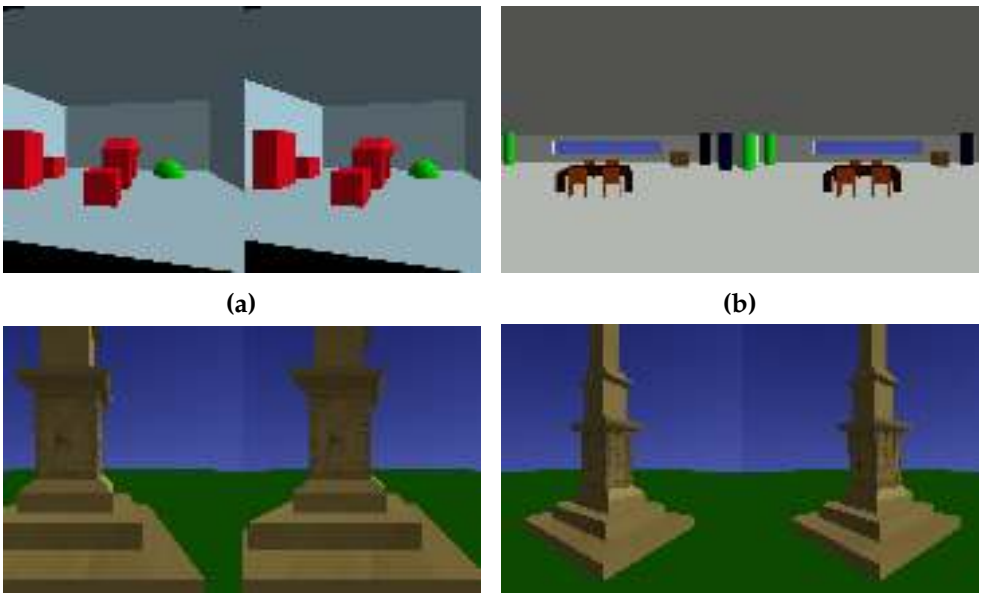


Figure 5.2: Different rooms rendered for a stereoscopic display device. Figures (a) and (b) are preliminary environments. Figures (c) and (d) show details of the "Große Mainzer Jupitersäule".

5.2 Software

The Durovis Dive offers a plug-in for the Unity development environment. It is possible to create applications both for Android and iOS. To better understand how stereoscopy works, we did not use this plug-in. Instead, we extended the libGDX framework adding stereoscopic functionality. LibGDX[19] is a game development framework for Java. It has a strong focus on cross platform development allowing to use the same code for Linux, Windows, Mac, Android and iOS. On the one hand, it is - in contrast to Unity - completely free and open-source under the Apache 2.0 license and it is also possible to use the development software on Linux. On the other hand, the features do not match that of Unity. For developing a prototype, the functionality of libGDX was adequate. It allowed us to extend the code without any problem when this was necessary. For a bigger project, a reevaluation of the used framework might be necessary.

Google Cardboard offers a free toolkit for Android. The source code of this development kit might be published in the future. Since the Cardboard device is so similar to the Durovis Dive and the toolkit is based directly on OpenGL, it might be interesting to compare their code to the one used here. The Cardboard API also indicates additional features like lens distortion[20] that are not implemented in our solution.

(a) (b)

Figure 5.3: A real reconstruction of the "Große Mainzer Jupitersäule" in Mainz (a) and our virtual version (b).

Figure 5.4: Photos of the real reconstruction were used for textures for the reliefs .

We created the virtual environments using the open-source 3D graphic software Blender[21]. A couple of rooms were modeled, exported to the fbx-format and then converted to libGDX's g3db-format. Using libGDX, our program reads these files, adds light, translates and rotates the camera based on the user's pose and then renders the scene (see figure 5.2). For illustrating the use of our system in exhibitions, we made a simple model of the "Große Mainzer Jupitersäule"[1](see figures 5.3 and 5.4).

1 The "Große Mainzer Jupitersäule" is an important monument from Roman times dedicated to the Roman god Jupiter. A reconstruction can be seen in Mainz, Germany.

6 Stereoscopy

6.1 Definition

The human brain obtains information about the distance of objects from a variety of cues. Some of these depth cues are[2, pp. 116-121]:

- Interposition: Objects in the foreground occlude background objects.
- Lightning and shading: Reflections of light and shadows help revealing the shape of an object.
- Linear perspective: Parallel lines are observed as converging in a distant vanishing point.
- Motion: Closer objects seem to move more quickly.

They all can be used by the observer to form the perception of depth in an image[22] and they can be found both in real life and in visual representations (e.g. paintings, photographs or movies).

Since the two human eyes have a different horizontal location, their view is not identical and they see two slightly different images of their surroundings (binocular disparity). This can be easily tested by looking at an object, closing one eye and then switching the viewing eye: the observed object seems to move a bit. This "displacement of objects viewed from different locations"[2, p. 119] is called parallax and it is used as another depth cue called stereopsis. It provides one of the highest degrees of precision[23, p. 2] and is usually dominant when in conflict with other depth cues[2, p. 120].

In spite of the importance of stereopsis, most display devices lack the possibility to create it. Although stereoscopic techniques have existed for a long time[1], they only obtained a wider distribution in recent years with the success of 3D movies and TV-screens. The principal idea behind most of these methods is to show a different image to each eye. This can be obtained by projecting the two images with differently polarized light. The user wears glasses that contain a polarization filter. The left part of the glasses blocks the polarized light of the image for the right eye and vice versa. Another possibility is the use of shutter glasses. Hereby, the images for the left and for the right eye are alternated rapidly on the display. The shutters

1 E.g. the display presented by Roese et al.[24] in 1979 as an example for a modern stereoscopic device.

in the glasses are synchronized so that the left shutter is closed while the image for the right eye is displayed and vice versa[22].

In the system we implemented, a head-mounted device is used. A display is placed closely to the user's eyes showing the two images side by side. Since the distance between the eyes and the display is so small, each eye can only see its corresponding image.

6.2 Theoretical Calculation of the Projection

We will now explain the geometrical theory on how to create stereoscopic images. Several publications about stereoscopy use this concept (or similar ones) when writing about projection theory. It differs somewhat from the way it can be implemented with the popular graphics library OpenGL. We will afterwards indicate on how to transform the concept so it can be implemented in OpenGL. The methods to create the stereoscopic images are independent of the technique that is used to display them and the same images can be used for all three display types we presented.

A non-stereoscopic, perspective projection is often defined by its projection plane and its center of projection (COP). The projection plane is also called viewing plane or image plane. It lies in the x-y plane and the projection of the three-dimensional world onto the projection plane is the basis for the rendered image that is shown on the screen. Although the plane itself has an infinite width and length, only the part that can be seen on the screen later on is of interest. Therefore, one can envision the projection plane as a bounded area in the virtual world the size of the screen in the physical world. The center of projection is often identified with the camera or the eye. It is located on the z-axis at $(0, 0, -d)$. Thus d is the distance between the COP and the projection plane.

To project an arbitrary point $p = (x, y, z)$ onto the projection plane, one calculates the intersection between the plane and the line that goes through p and COP as visualized in figure 6.1. The line could be pictured as the trajectory of the light from p to the viewer's eye. The intersection (x_p, y_p) can be calculated using the intercept theorem [25, pp. 276-277] :

$$\frac{x_p}{d} = \frac{x}{d+z} \Rightarrow x_p = \frac{x \cdot d}{d+z} \tag{7}$$

$$\frac{y_p}{d} = \frac{y}{d+z} \Rightarrow y_p = \frac{y \cdot d}{d+z} \tag{8}$$

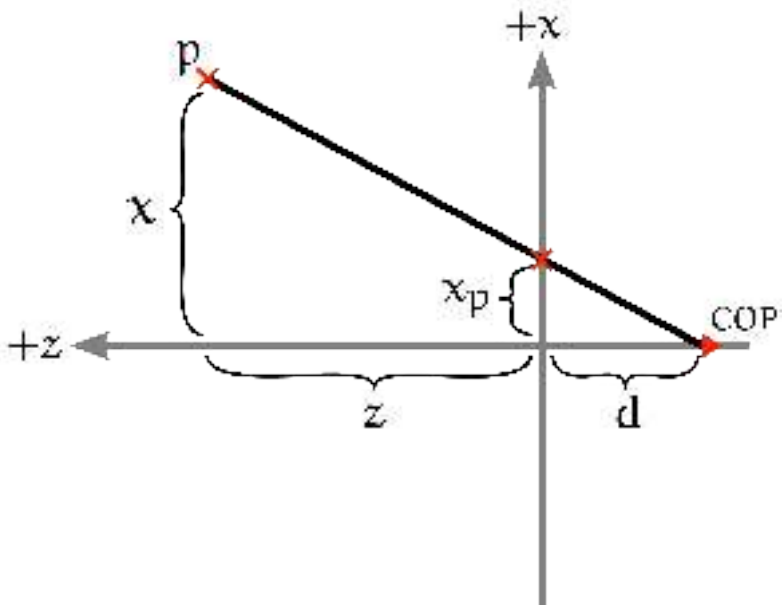


Figure 6.1: A non-stereoscopic projection.

For the stereoscopic projection called off-axis projection[2] the COP is moved along the x-axis to $(-e/2, 0, -d)$ for the left eye (COP_l) and $(e/2, 0, -d)$ for the right eye (COP_r, see figure 6.2a). The variable e represents the horizontal distance between the left and the right eye (interocular distance). We call the image for the left eye left image and the image for the right eye right image. The projected point's coordinate x_{pl} for the left image is derived using again the intercept theorem. For $x_{pl} \geqslant 0$ and $x_{pl} \leqslant -\frac{e}{2}$ this is outlined in figures 6.2b and 6.2c:

$$\begin{aligned} \frac{x_{pl} + \frac{e}{2}}{d} &= \frac{x + \frac{e}{2}}{d + z} \\ \Rightarrow x_{pl} &= \frac{(x + \frac{e}{2}) \cdot d}{d + z} - \frac{e}{2} \\ &= \frac{x \cdot d + \frac{e}{2} \cdot d}{d + z} - \frac{\frac{e}{2} \cdot (d + z)}{d + z} \\ &= \frac{x \cdot d - \frac{e}{2} \cdot z}{d + z} \end{aligned} \tag{9}$$

2 Derivation based on the formulas by Hodges [22].

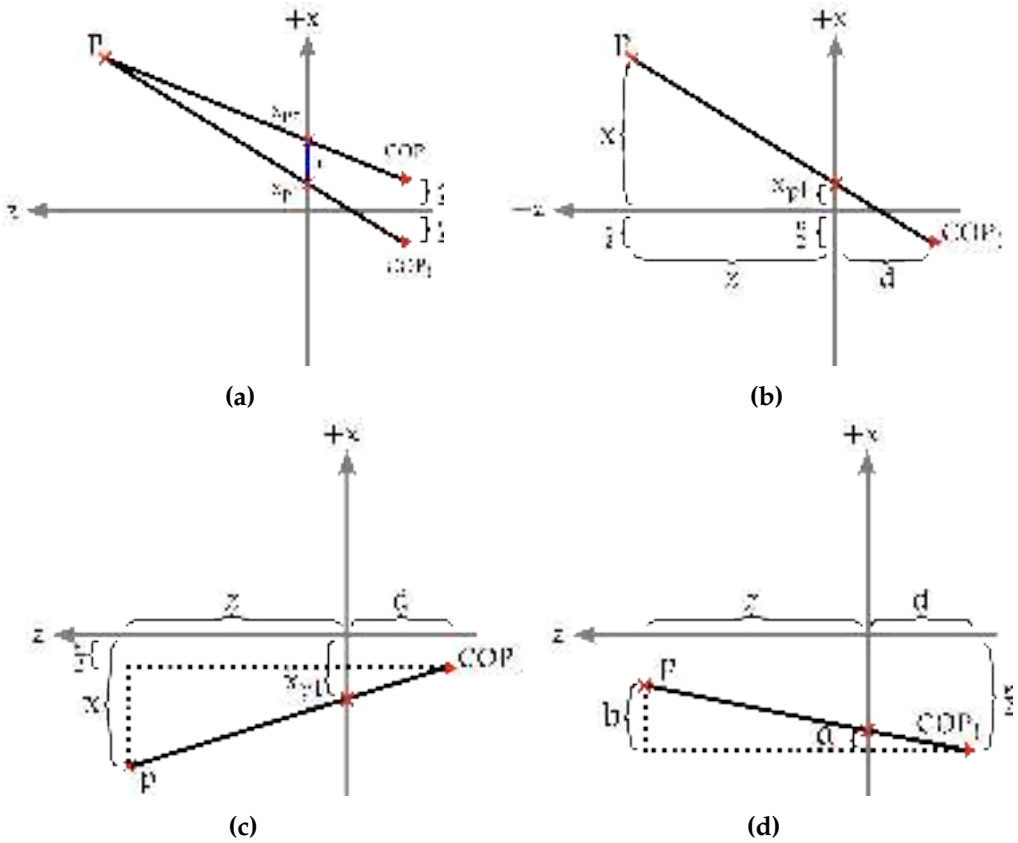


Figure 6.2: Illustrations of stereoscopic projections. In (a), point p is projected on the projection plane for the left and the right eye. The parallax r is marked blue. Figure (b) shows a projection for the left eye, if $x_{pl} \geqslant 0$. Figure (c) visualizes the case $x_{pl} \leqslant -\frac{e}{2}$ and (d) the case $-\frac{e}{2} < x_{pl} < 0$.

For $-\frac{e}{2} < x_{pl} < 0$ see figure 6.2d:

$$\begin{aligned}
a &= -\frac{e}{2} - x_p \\
b &= -\frac{e}{2} - x \\
\frac{a}{d} &= \frac{b}{d+z} \\
\Rightarrow &\frac{-\frac{e}{2} - x_{pl}}{d} = \frac{-\frac{e}{2} - x}{d+z} \\
\Rightarrow &\frac{x_{pl} + \frac{e}{2}}{d} = \frac{x + \frac{e}{2}}{d+z} \\
\Rightarrow &x_{pl} = \frac{x \cdot d - \frac{e}{2} \cdot z}{d+z}
\end{aligned} \tag{10}$$

The calculation for the right image is similar. Only the sign of the translation changes:

$$\begin{aligned}
\frac{x_{pr} - \frac{e}{2}}{d} &= \frac{x - \frac{e}{2}}{d+z} \\
\Rightarrow x_{pr} &= \frac{x \cdot d + \frac{e}{2} \cdot z}{d+z}
\end{aligned} \tag{11}$$

The formulas for y_{pl} and y_{pr} are equal to formula (8), because they do not depend on the x-value of the COP.

As stated above, we try to achieve a parallax, i.e. the position of an object on the left image should be slightly different from the one on the right image (see figure 6.2a). The actual horizontal parallax r can be calculated as

$$\begin{aligned}
r &= x_{pr} - x_{pl} \\
&= \frac{x \cdot d + \frac{e}{2} \cdot z}{d+z} - \frac{x \cdot d - \frac{e}{2} \cdot z}{d+z} \\
&= \frac{e \cdot z}{d+z}
\end{aligned} \tag{12}$$

When the viewer's brain fuses the left and the right image of a stereoscopic projection into one, an object with a positive parallax appears to be behind the screen. The smaller the parallax, the closer the object seems to be compared to the screen. An object with no parallax appears to lie in the plane of the screen and an object with a negative parallax seems to hover in front of the screen[26].

By analyzing formula (12), one can see that the projection we presented achieves this phenomenon. The further away an object is from the COP, the greater is the value of z (assuming of course that the object is in front of the viewer and thus visible). Since e and d are fixed values, the parallax r increases with the same rate

as z. Therefore, the larger the object's distance to the COP on the z-axis, the further away behind the screen it appears to the viewer. If the object lies on the projection plane, the parallax is 0 and if the object is in front of the plane the parallax is negative.

Off-axis projection is not the only method to achieve a stereoscopic effect. Another popular technique is the toe-in-projection, where the two cameras are rotated around a vertical axis through the center of the scene[22]. This introduces, however, a vertical parallax that increases towards the corners of the screen and that can make it impossible for the brain to fuse the images[26] destroying the stereoscopic effect. This vertical parallax cannot occur in the off-axis projection because y_{pl} and y_{pr} are equal. Although the toe-in technique is easier to implement and the parameters can be adjusted so that the artifacts are within an acceptable limit[26], we chose to implement the off-axis method.

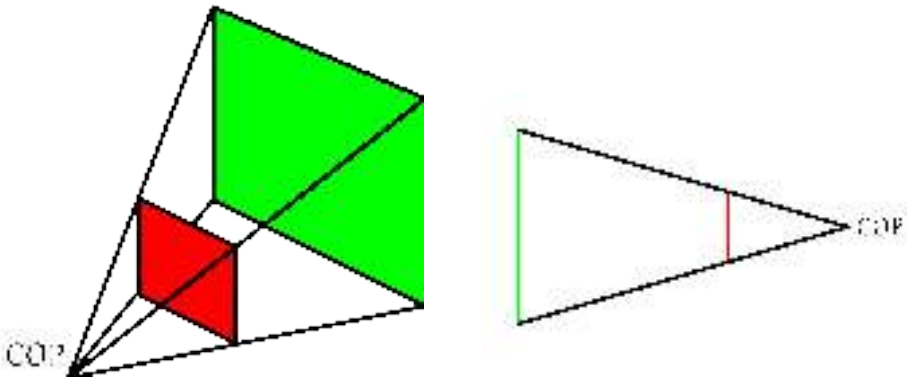
6.3 Implementation With OpenGL

When implementing stereoscopy using OpenGL, the concept has to be adapted[3], because the perspective projection in OpenGL works differently from the idea we presented above.

In OpenGL, the center of projection lies in the origin of the coordinate system. A non-stereoscopic, perspective projection is defined by the viewing frustum as illustrated in figure 6.3. All elements outside of the frustum are not drawn. All elements inside the frustum are projected onto the near plane, so the near plane is similar to the projection plane in the former concept. This causes a conflict for the implementation of a stereoscopic camera. On the one hand, the near plane would have to be placed at the distance d from the camera to represent the screen in the physical world. On the other hand, all objects between the camera and the near plane are not drawn. This is not acceptable, because a lot of objects would disappear that are close to the user (and thus usually especially interesting).

Instead, the near plane is placed in an arbitrary small distance n to the camera and a screen is introduced to the virtual world. This virtual screen is not visible and it does not even need to exist as an actual element in the environment. It is just a theoretical representation of the screen in the physical world. Its dimension and position are necessary to adjust the frustums of the two stereoscopic cameras so that the parallax effect works.

3 This adaptation is based on the implementation by Kooima[27]. He presents a more general system which we modified, because in our scenario the user's position relative to the screen is always fixed. He gives, however, no indication on how his implementation is connected to the theoretical concept or why the stereoscopic effect works. We added these parts ourselves.



(a) (b)

Figure 6.3: A viewing frustum in 3D (a) and 2D (b) as used in OpenGL. The near plane is colored red and the far plane green. The intersection point of the lines is the center of projection.

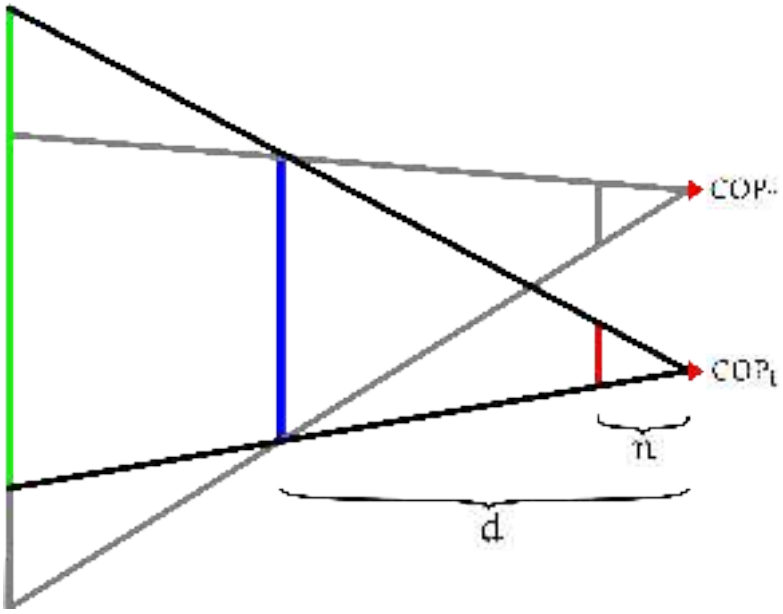


Figure 6.4: Illustration of two frustums for stereoscopic rendering in OpenGL. The screen is colored blue, the near and far planes of the left camera are red and green.

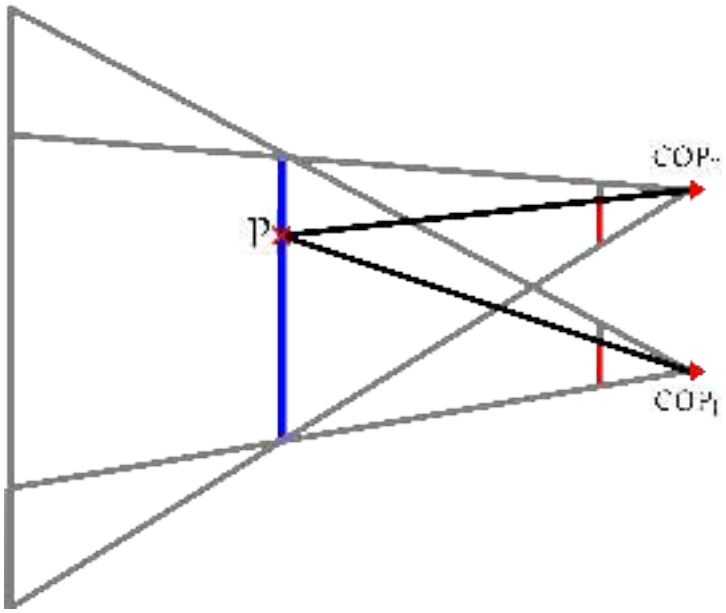


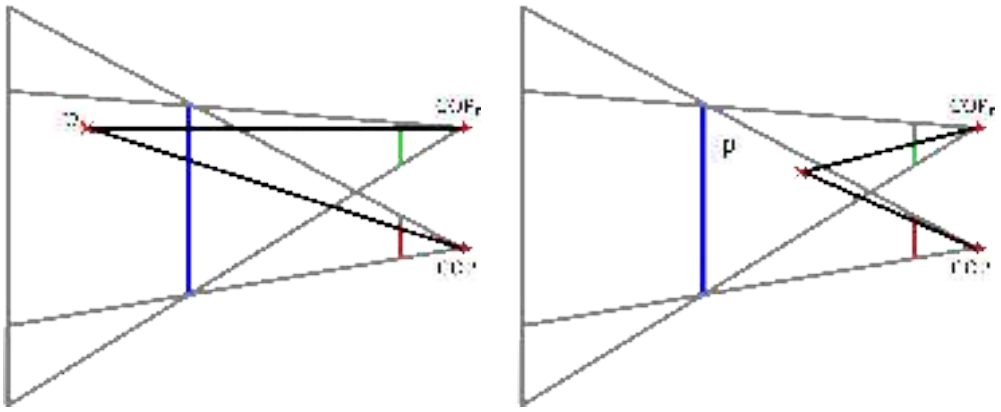
Figure 6.5: Projection of a point p onto the near plane of two frustums. Point p lies on the screen (blue line). The x-values of both projected points are equal (red lines).

Such a construction can be seen in figure 6.4. The frustums of the cameras are chosen so that they tangent the borders of the screen. Given the distance n, the position of the near plane follows directly. In order to evaluate if the stereoscopic effect can still work, we have to analyze the parallax. If a point p lies on the screen and is projected on the near planes of both cameras, the x-values of the projected points are equal (see figure 6.5). Thus the parallax $r = x_{pr} - x_{pl}$ is zero. If p lies behind the screen, the x-value of the projected point on the left image is always larger than on the right image and the parallax is positive (figure 6.6a). The opposite occurs when p lies in front of the screen (figure 6.6b). So the parallax acts in the same way as in the concept of 6.2 ensuring that objects behind the theoretical screen appear to lie behind the physical screen and objects in front of the theoretical screen also appear to hover in front of the physical one.

Figure 6.7 illustrates the proof that $x_{pl} < x_{pr}$ if p lies behind the screen. Since one frustum is the reflection of the other at the z-axis, it holds that $c = e$ and $C = E$. The intercept theorem can now be used to show that

$$\frac{B}{b} = \frac{D}{d} = \frac{B'}{b'} = \frac{E}{e'} = \frac{C}{c} = \frac{A}{a} \tag{13}$$

Since p lies behind the screen and thus $A < B$, it follows that $a < b$. The variable a is equivalent to x_{pl} and b to x_{pr}. The other two cases (p on screen and p in front of screen) can be proven similarly.



(a) (b)

Figure 6.6: Projection of a point p onto the near planes of two frustums. In (a) the point lies behind the screen (blue line) and the x-value on the left projection (red line) is smaller than the value on the right projection (green line). In (b) the point lies in front of the screen and the x-value on the left projection is greater than the value on the right one.

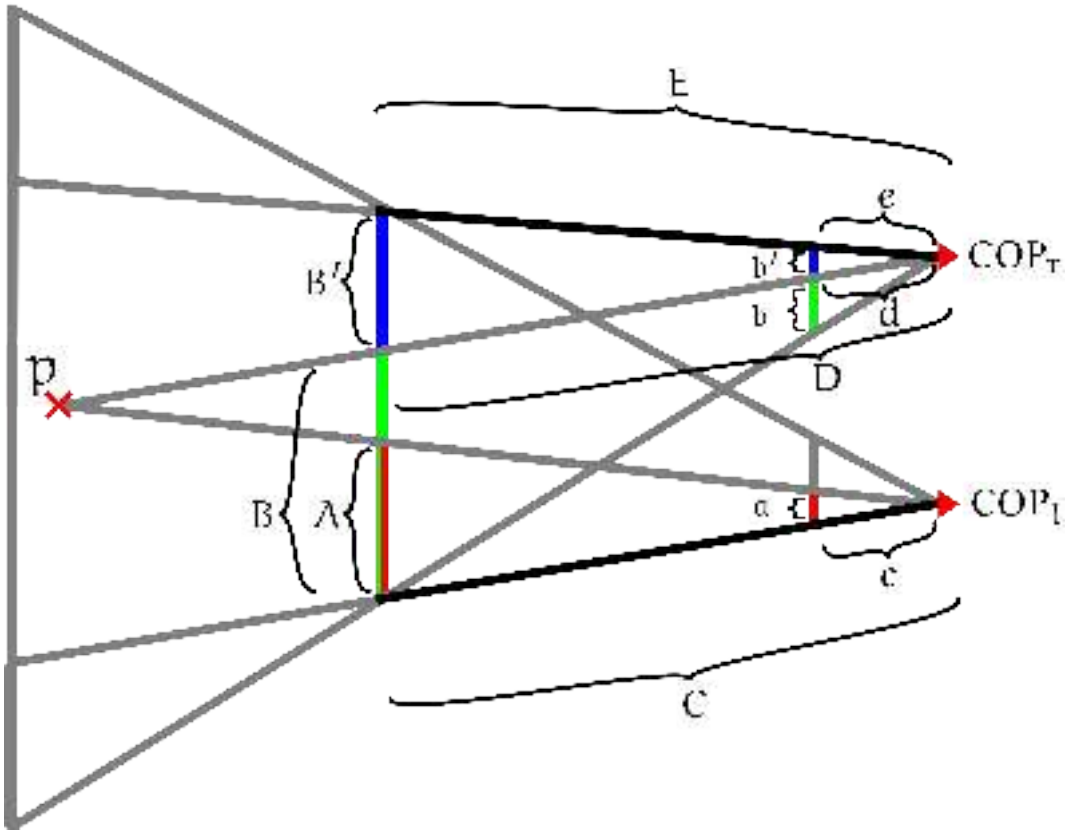


Figure 6.7: Illustration of the proof that $x_{pl} < x_{pr}$ if p lies behind the screen.

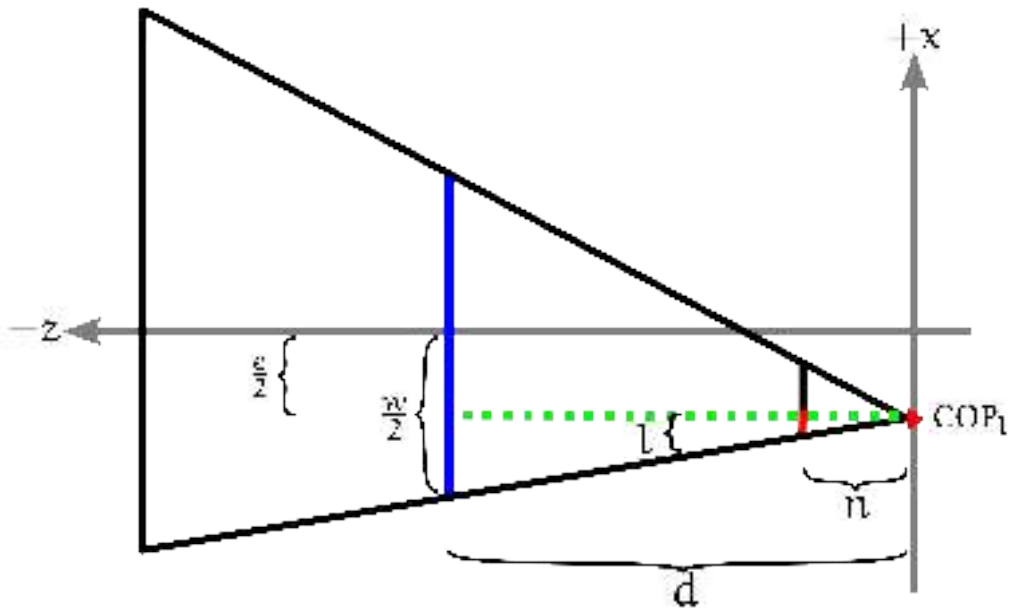


Figure 6.8: Calculation of the frustum's parameter left (l) using the intercept theorem.

The only aspect that is missing is the calculation of the characteristics of the near plane. OpenGL needs the parameters near, far, left, right, top and bottom. Based on these parameters the projection matrix is computed using the function "glFrustum". The value of near is n, the distance between the camera and the near plane. The value of far is the distance between the camera and the far plane. The stereoscopic camera puts no constraints on these values. Usually they are chosen based on the characteristics of the virtual environment, the capacity of the rendering device, etc. As it can be seen in figure 6.8, the intercept theorem can be used to calculate the value for left:

$$\begin{aligned} \frac{l}{n} &= \frac{\frac{w}{2} - \frac{e}{2}}{d} \\ \Rightarrow l &= \frac{\frac{w}{2} - \frac{e}{2}}{d} \cdot n \end{aligned} \tag{14}$$

with l being the parameter left and w the width of the virtual screen. The other three parameters right, top and bottom can all be calculated using the same approach. After computing the projection matrix with these parameters, it has to be translated by $-\frac{e}{2}$ on the x-axis since OpenGL assumes the COP in the origin and not in the actual position. Since the camera is immobile, this is done by translating all objects in the opposite direction.

We have implemented this concept using the libGDX framework and extending the class "Camera" to support stereoscopy. It was tested on the Durovis Dive and on a 3D-TV. The stereoscopic effect is visible on both devices. The parameters e and d have to be adjusted for both devices to allow comfortable viewing. Hodges[22]

assumes an average interocular distance of 6.35 centimeters. He indicates, however, that a smaller value should be used. Otherwise users could have problems fusing the two images. In our setup in different virtual environments, different values achieved good results. The values also depended on the used display device. A deeper analysis should be done to test whether the stereoscopic effect can be improved by optimizing the distance.

7 Evaluation and Future Work

After having presented every part of the system, we now evaluate them and suggest possible improvements for future work. We also compare our solution to an omnidirectional treadmill.

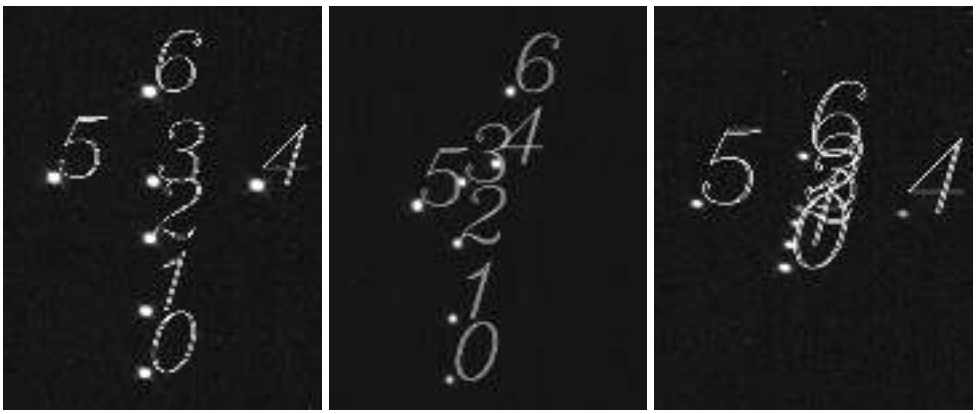
7.1 Marker

The tracking software recognizes our marker up to a distance of 4 meters using an infrared filter and an exposure time of 1 ms. This is less then the 7 meters achieved by Tjaden et al.[9]. The main reason probably is that their marker is larger (8,6 x 6,3 cm compared to 11,4 x 7,6 cm). If the size of the tracker is increased, this also allows to increase the exposure time and the diaphragm's aperture. The light blobs will be brighter and bigger (thus visible at greater distances) without overlapping.

Several factors exist that limit the marker's rotation: Lights can overlap inhibiting a separation of the blobs, parts of the circuit board can occlude the lights and the viewing angle can be too small so that the blob is not bright enough. Latter case can be seen in figure 7.1c, where the central light is barely bright enough to be detected. We achieved rotations around the x- and the y-axis between 60°and 80°(see figure 7.1). This seems consistent with the results of Tjaden et al.. The closer the distance to the camera the bigger gets the maximal possible rotation.

7.2 Tracking

The system by Tjaden et al. allows a very fast and accurate tracking which results in very smooth and natural movements in the virtual environment. Unfortunately, the camera costs over 400 €. In the current setup it is by far the most expensive element of our construction. In future work, ways could be examined to reduce the camera's cost. First, the frame rate of the camera is higher than the frame rate of the display device. Even in a simple 3D-world the rendering frame rate we measured never exceeded 70 images per second. This means that even in the best case about every second pose is dropped. Thus a camera with a lower frame rate could be used which would probably be cheaper. Second, the camera is targeted at professional use, e.g. in the industrial sector. A more economically priced camera would have less features and likely introduce more noise into the images. This would probably reduce the tracking accuracy. Since the accuracy is already very high, it might be



(a) (b) (c)

Figure 7.1: Camera images of the marker analyzed by the tracking software. The marker is at a distance of 1 meter and has no rotation (a), maximal rotation around x-axis (b) and maximal rotation around y-axis (c).

possible to achieve acceptable results even with a lower accuracy. An extreme case would be to try a normal webcam.

The main problem of the tracking procedure is that the marker is sometimes not recognized by the tracking software. The user notices this either because the image he or she sees jumps between two positions or the image freezes completely. Since the camera is placed at the edge of the room, it usually sees an inclined version of the marker (see figure 7.2a), assuming that the user mostly stands or walks upright. As analyzed above, an increase in rotation reduces the chances of a successful tracking. Therefore, it might be beneficial to place the camera - if possible - on a higher position or on the ceiling (see figure 7.2b). This should enhance the tracking quality, because the camera would then look directly onto the marker most of the time.

Tjaden et al. note in their publication that their tracking system can be expanded to use multiple markers or multiple cameras. Placing several markers around the user's head could be an easy way to ensure that at least one marker is always visible to the camera. Alternatively, several cameras could be distributed in the room. This is a more complicated approach, but it might be necessary for huge rooms where the distance to one camera might become too large. Multiple markers could also be used to let various users interact in the same physical and virtual environment. Since a camera can track several markers at the same time, the cost and effort for multiple users is relatively low.

(a) (b)

Figure 7.2: The camera's viewing angle of the marker if the camera is positioned at the edge of the room (a) or on the ceiling (b).

7.3 Network

We have tested the network connection at several locations. To estimate the latency we wrote a test program that measures the time difference between a message and its response (round-trip or ping time). The results varied between 1 ms to 7 ms per way. This value is smaller than the difference between two rendered images, so the network latency should not be problematic.

A small problem exists due to the fact that the smartphone's and the server's software are not perfectly synchronized. Each time the smartphone renders a frame or image, it queries the latest pose that has been received. It happens sometimes that no new pose has arrived since the last query. Instead in the next query, two poses will be available. One could either put all arriving poses in a FIFO queue and use the first item of the queue in each frame or one could ignore the old poses completely and always take the latest. We decided for latter approach, because it minimizes the discrepancy between the physical pose and the pose in the rendered image. This technique could result in small jumps due to discarded poses, but we could not detect them in our tests.

Once the connection is established, it has proven to fulfill its requirements on speed and reliability. Improvements could be done in the discovery phase. Currently if no answer is received from the server, the program is stuck. A better error handling could be implemented by resending the discovery signal after a time-out and/or allowing the manual input of the server's address.

7.4 Display

The Durovis Dive in combination with the Nexus 5 smartphone and the libGDX graphic library offer an acceptable image quality. The wearing comfort of the head-mounted device is good and the stereoscopic effect works well. However, some minor problems exist: the size of the screen does not fill the field of vision completely. Also the smartphone's buttons are visible and some light enters from below. All three reduce the immersion. A higher resolution would be desirable as well.

The simulation we implemented does not require any user control except for the user's movement. More advanced programs will probably need some additional form of user input. Some applications for the Durovis Dive and the Google Cardboard use buttons that exist in the virtual world. By looking for a certain time period on such a button, the user can "click" on them. Other applications use traditional, often wireless game controllers. Additional markers could be used to transform real world objects into virtual objects the user can control and interact with.

The smartphone's camera can be used while wearing the Durovis Dive. This could allow to create an augmented reality scenario. The user sees the real world in front of him or her through the camera's images. Based on the user's pose additional, virtual objects are added to these images as if they really existed in the physical world. Ideally, the camera would be mounted between the eyes. The smartphone's camera is positioned on the left side. This must be taken into account.

7.5 Simulator Sickness

A major problem many virtual reality applications face is simulator sickness. When using such a system over a prolonged time period, one may develop symptoms such as sweating, dizziness or nausea[28, pp. 27-28]. Simulator sickness is similar to motion sickness and one explanation may be the cue conflict theory[28, pp. 17-18 and 48-49]. It assumes that the brain collects input about the body's motion from several sensors, among them the eyes and the vestibular system. Based on learned movement patterns from the past, a certain motion is expected. If the expected motion does not match the actual motion, this can trigger both motion sickness and an adaptation that learns this new motion pattern. An example for this is seasickness. When sitting below deck of a ship, the eyes see no movement. The vestibular system, however, registers the ship's rocking. This can cause motion sickness. Most people adapt after a while and the symptoms disappear.

In virtual reality, this may occur when the physical motion and the visually displayed motion differ. Groen et al. analyzed experimental data that suggests this[29]. When testing our system, very slight symptoms of sweating and nausea could be detected. A reason could be the delay caused by the time needed to track, transmit and visualize the current physical pose. This causes the optical feedback to the brain to be always a little behind the other body sensors. While this discrepancy is not visible by just looking at the screen, it might be enough to trigger simulator sickness. Further study would be needed to analyze the symptoms and how they can be prevented. To reduce the difference between the physical and the visualized movement, it might be possible to combine the tracked pose with information from the smartphone's gyroscope and accelerometer.

7.6 Comparison to the Cyberith Virtualizer

The omnidirectional treadmill Cyberith Virtualizer was exhibited at the Gamescom 2014 in Cologne. It consists of a low-friction base plate, a pillar construction and a harness that holds the user (see figure 7.3). Sensors inside the pillars measure the user's vertical position and optical sensors in the base plate recognize the movement of his or her feet. An Oculus Rift is used as display device and headphones add audio to improve the immersion. It allows the user to walk, crouch and jump in all directions. They were so kind to let us test their device. In this test version, the user's height was fixed, so it was only possible to walk and rotate and not to jump or crouch. It has not recognize different walking velocities, yet. The walking was not completely natural since one had to lean forward with the upper part of the body and push against the holding ring. The game one could play was rather simplistic. One had to walk through a sinister maze avoiding monsters that hid in the corridors. It took about five to ten minutes to get used to the treadmill. After that a major immersion into the game was possible. Although very little actually occurred in the game it was thrilling and exciting.

The Virtualizer offers an interesting possibility for movement in a virtual environment. It overcomes a major disadvantage of our system which is the limitation of the physical space. Our virtual rooms must not be larger than the actual physical room the user walks in. Otherwise the user walks unsuspectingly into walls. If a larger environment should be visualized, barriers have to be introduced into the virtual world that indicate the borders of the accessible space. On the other hand, we see two major disadvantages of the Virtualizer. First, there is its high costs per unit. Second, the user needs to get accustomed to walking in the treadmill. When we tested our construction with other people, they instinctively put on the head-mounted display as if they were wearing sky goggles. And after adjusting the lenses of the Durovis Dive they orientated themselves very quickly.

(a) **(b)**

Figure 7.3: Two photos of the Virtualizer from Cyberith's press material[30].

Since the movements are no different from normal walking, no adaptation was necessary.

In our opinion, the Virtualizer and our system have two different purposes. The Virtualizer aims at video games. Here the physical limitation of the walking radius our system has would usually be unacceptable. The longer familiarization period is not problematic as a player would normally use the device regularly. But for a one-time use e.g. in exhibitions, a user-friendlier and more intuitive application like ours might be advantageous.

7.7 Conclusion

In this work, we demonstrated the possibility to combine optical position tracking and a head-mounted display to create a virtual reality environment. The user is able to walk naturally through a virtual world. He or she can crouch, jump, turn around, look upwards and downwards, etc. Both the transmission of the tracked pose to the display device and the rendering of the virtual scene from the user's perspective work fast and reliably. As it is in the nature of a prototype, the handling is still a bit crude. But it should be possible to fix this adequately in the future with a GUI for the software and a helmet that holds the marker. Unfortunately, the movement radius is still limited. A larger tracker and another camera position should solve this problem. In a more complex scenario, the use of several cameras or markers could be an interesting option.

Figure 7.4: Our system is tested in the virtual reality laboratory of the Johannes Gutenberg University Mainz. The marker still has to be hold by hand.

The areas of application of our system are manifold. We implemented a scenario that suggests the use in museums or exhibitions to bring past or future times to life. This is not limited to purely virtual worlds. It could be combined with augmented reality techniques, e.g. to virtually rebuild a Roman ruin allowing the user to walk through it, seeing both the original stones and the reconstructed architecture. Both in science and industry, 3D representations can be found and virtual reality has been used to improve the handling of these visualizations. One of many examples is the design of complex proteins which can be enhanced using the combination of manual work and algorithms[31]. Another example is the planning of an assembly process[32]. Although the mentioned limitations exist, a use for certain entertainment purposes is still possible. An interesting enhancement would be the incorporation of a multi-user environment.

A Appendix

A.1 CD

The appended CD contains the compiled programs we developed as well as the corresponding source code and additional material like 3D models and textures. In the following we will give some explanatory notes on how to run the programs and how to compile the code.

A.2 Programs

- Server: The programs were tested on a Windows 8 system with a 64-bit architecture. They were, however, compiled for 32-bit so the programs should run on most Windows computers. The firewall might need to be configured for all programs to work correctly. Unfortunatelly versions for other operation systems do not exist due to missing support by certain libraries (see notes on compilation below).

  - The UDP server is a standalone console program that can be used for both the TCP ping application and the tracking server. It waits for a broadcast message from the client and then sends a corresponding answer. It is necessary to run this program parallely to one of the other server programms so that the client can find the server's IP address.

  - The TCP ping server is a standalone console program. It can be used to test the latency of the network connection to the client.

  - The tracking server is a standalone console program which opens a window as soon as the tracking starts. This is the program used for the computation and transmission of the pose. It exists in a normal version with network capabilities and a version for testing the tracking that does not wait for a network connection.

  - To control the camera the FlyCapture SDK must be installed (tested version 2.7 Release 8, 32-bit).

- Android: Both apps were tested on the Nexus 5 with Android 4.4. The Android version must be 2.3 or newer since certain functionalities are used

that only exist since API version 9. It is necessary to allow the installation of apps that are not from the Google Play Store and it might be necessary to active the developer mode. Rooting or similar changes are not necessary.

  - The TCP ping application is the counterpart to the TCP ping server. It must be started after starting the UDP and the TCP ping server.

  - The VRRoom application is the counterpart to the tracking server. It uses the poses the server sends to render the virtual room from the user's perspective. It must be started after starting the UDP and the tracking server.

- The alternative client VRRoomDesktop allows to demonstrate the rendering on a desktop computer. The Java Runtime Environment 7 is necessary. The program can be started by either clicking on the jar-file or by running the command line argument "java -jar VRRoomJupiter.jar". The camera can be controlled with the keys WASDQE. By pressing the key M automatic camera movement can be activated. The program also exists in a version without network capability. The client was tested on Xubuntu 14.04 and Windows 8. It should run on all current versions of Linux, Windows and Mac OS.

A.3 Compiling

- Server: The programs have been written in C++ and compiled using Microsoft Visual Studio 2013. The source code, the header files for the tracking library and a compiled version of the tracking library can be found on the CD. The FlyCapture SDK (version 2.7 Release 8, 32-bit) and the OpenCV library (version 2.44, 32-bit, exact version is needed) have to be installed additionally. The TCP ping server uses constructs from the C++11 standard to measure the time. If C++11 is not available these could be replaced by other time-measuring functions. The tracking library only works on Windows (32-bit) and the FlyCapture SDK is only available for Windows and Linux. In our code only the socket.cpp is limited to Windows.

- Client: The CD contains the working directories of the Android Developer Tools, a special version of the Eclipse IDE. Any other development tool for Android and/or Java can also be used. The VRRommDesktop project shares several classes with the VRRoom project.

- Models: The blend-, fbx- and g3db-files of all models are included on the CD along with all textures and the fbx-converter.

REFERENCES

[1] J. Steuer. Communication in the age of virtual reality. chapter Defining Virtual Reality: Dimensions Determining Telepresence, pages 33–56. L. Erlbaum Associates Inc., Hillsdale, NJ, USA, **1995**.

[2] W. R. Sherman and A. B. Craig. *Understanding Virtual Reality: Interface, Application, and Design*. Morgan Kaufmann Publishers, **2003**.

[3] P. Rubin. The inside story of Oculus Rift and how virtual reality became reality. http://www.wired.com/2014/05/oculus-rift-4/, **May 2014**. [accessed 19/09/2014].

[4] Sony computer entertainment announces "Project Morpheus". http://www.sony.com/SCA/company-news/press-releases/sony-computer-entertainment-america-inc/2014/sony-computer-entertainment-announces-project-morp.shtml, **March 2014**. [accessed 19/09/2014].

[5] New Xbox head Phil Spencer discusses evolving games industry. http://fortune.com/2014/04/25/new-xbox-head-phil-spencer-discusses-evolving-games-industry, **April 2014**. [accessed 19/09/2014].

[6] M. Slater, M. Usoh, and A. Steed. Taking steps: The influence of a walking technique on presence in virtual reality. *ACM Trans. Comput.-Hum. Interact.*, 2(3):201–219, **September 1995**.

[7] Virtuix Omni. http://www.virtuix.com. [accessed 19/09/2014].

[8] Cyberith Virtualizer. http://www.cyberith.com. [accessed 19/09/2014].

[9] H. Tjaden, F. Stein, E. Schömer, and U. Schwanecke. High-speed and robust monocular tracking. *Submitted to VISAPP2015*.

[10] Point Grey. Flea 3. http://www.ptgrey.com/products/flea3_usb3/datasheet_flea3_usb3.pdf. [accessed 19/09/2014].

[11] A. Velizhev. GML C++ camera calibration toolbox. http://graphics.cs.msu.ru/en/node/909. [accessed 19/09/2014].

[12] OptoSupply. OSI3NAS1C1A. http://www.optosupply.com/UploadFile/PDF/osi3nas1c1a.pdf. [accessed 19/09/2014].

[13] T. Lindholm, F. Yellin, G. Bracha, and A. Buckley. *The Java© Virtual Machine Specification: Java SE 7 Edition*, chapter The class File Format. Oracle America, Inc., **2013**.

[14] C. Cruz-Neira, D. J. Sandin, and T. A. DeFanti. Surround-screen projection-based virtual reality: The design and implementation of the CAVE. In *Proceedings of the 20th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '93, pages 135–142, New York, NY, USA, **1993**. ACM.

[15] K. Reda, A. Febretti, A. Knoll, J. Aurisano, J. Leigh, A. Johnson, M. Papka, and M. Hereld. Visualizing large, heterogeneous data in hybrid-reality environments. *Computer Graphics and Applications, IEEE*, 33(4):38–48, **July 2013**.

[16] Google. Nexus 5 tech specs. http://www.google.com/nexus/5/. [accessed 19/09/2014].

[17] Oculus VR. The all new oculus rift development kit. http://www.oculusvr.com/dk2/. [accessed 19/09/2014].

[18] Google. Cardboard. https://developers.google.com/cardboard/. [accessed 19/09/2014].

[19] M. Zechner. libGDX. http://libgdx.badlogicgames.com. [accessed 19/09/2014].

[20] Google. Cardboard API reference distortion. https://developers.google.com/cardboard/api/reference/com/google/vrtoolkit/cardboard/Distortion. [accessed 19/09/2014].

[21] Blender. http://www.blender.org. [accessed 19/09/2014].

[22] L. F. Hodges. Tutorial: Time-multiplexed stereoscopic computer graphics. *IEEE Comput. Graph. Appl.*, 12(2):20–30, **March 1992**.

[23] I. P. Howard. *Binocular Vision and Stereopsis*. Oxford University Press, **1995**.

[24] J. A. Roese and L. E. McCleary. Stereoscopic computer graphics for simulation and modeling. *SIGGRAPH Comput. Graph.*, 13(2):41–47, **August 1979**.

[25] A. Foley, James D. ; VanDam. *Fundamentals of Interactive Computer Graphics*. The Systems Programming Series. Addison-Wesley, Reading, Mass., repr. with

corr. edition, **1984**.

[26] L. F. Hodges and D. F. McAllister. Rotation algorithm artifacts in stereoscopic images. *Optical Engineering*, 29(8):973–976, **1990**.

[27] R. Kooima. Generalized perspective projection. Technical report, Louisiana State University, **June 2009**.

[28] D. M. Johnson. Introduction to and review of simulator sickness research. Technical report, DTIC Document, **2005**.

[29] E. Groen and J. Bos. Simulator sickness depends on frequency of the simulator motion mismatch: An observation. *Presence*, 17(6):584–593, **Dec 2008**.

[30] Cyberith Virtualizer press. http://cyberith.com/press/. [accessed 19/09/2014].

[31] A. Anderson and Z. Weng. VRDD: Applying virtual reality visualization to protein docking and design. *Journal of Molecular Graphics and Modelling*, 17(3 - 4):180 – 186, **1999**.

[32] A. Seth, J. M. Vance, and J. H. Oliver. Virtual reality for assembly methods prototyping: a review. *Virtual Reality*, 15(1):5–20, **2011**.