RWTH AACHEN UNIVERSITY

VirlWind

A mobile device for user-centered augmented reality

Diploma Thesis at the
Media Computing Group
Prof. Dr. Jan Borchers
Computer Science Department
RWTH Aachen University

by
Alexander Strauch

Thesis advisor:
Prof. Dr. Jan Borchers

Second examiner:
Prof. Christian H. Bischof, Ph.D.

Registration date: Jan 23th, 2008
Submission date: Sep 10th, 2008

I hereby declare that I composed this thesis entirely myself and that it describes my own work. All other sources and tools are marked accordingly.

_____

Alexander Strauch, Aachen September 9, 2008

Contents

List of Figures

List of Tables

Abstract

Due to the constantly increasing power and the decreasing weight and size of mobile devices they become more and more attractive for mobile Augmented Reality applications. But although there exist some approaches, they are not fully mobile (Tsang et al. [2002], McKenna [1992]), they lack the user-centered perspective (Henrysson et al. [2005], Fitzmaurice [1993]) or they are too expensive and require the user to wear special equipment (Gleue and Dähne [2001], Thomas et al. [2002]).

The aim of this thesis therefore was to combine the best parts of these techniques and create a

- mobile,
- robust and
- cost-saving

solution for intuitive three-dimensional navigation.

Therefore we utilized a conventional tablet computer with an internal camera and an additional external camera mounted to the backside. In this way we could then use the external camera to determine the position of the device inside the real world by the use of fiducial markers and a multi marker tracking approach. The internal camera however was used to keep track of the position of the user's eyes. After estimating the real world coordinates of the user's head, relative to the device, we then calculated the appropriate user-centered perspective based on those two results.

In the end we conducted a user study to evaluate our system and compared the normal Augmented Reality method to our approach with user-centered perspective. Although the hardware we used for the prototype was limited, the tests indicated that this technique could be very promising for intuitive 3D navigation devices.

Überblick

Durch die stetig wachsende Leistungsfähigkeit und den immer weiter reduzierten Baugrößen und Gewichten werden mobile Geräte immer attraktiver für Anwendungen die mobile erweiterte Realität erlauben. Doch obwohl einige Ansätze in diesem Gebiet existieren sind diese entweder nicht vollkommen mobil (Tsang et al. [2002], McKenna [1992]), bieten keine betrachterzentrierte Perspektive (Henrysson et al. [2005], Fitzmaurice [1993]) oder sind zu teuer und verlangen vom Benutzer das Tragen spezieller Ausrüstung (Gleue and Dähne [2001], Thomas et al. [2002]).

Ziel dieser Diplomarbeit war es deshalb die besten Teile dieser Techniken zu vereinen und eine

- mobile,
- robuste und
- kostensparende

Lösung für intuitive dreidimensionale Navigation zu bauen.

Um dies zu erreichen kam ein herkömmlicher Tablet Computer mit einer eingebauten Kamera zum Einsatz, den wir mit einer zusätzlichen externen Kamera für die Rückseite des Gerätes erweiterten. Auf diesem Weg konnten wir die externe Kamera dazu nutzen die Position des Gerätes innerhalb der realen Welt anhand von Markern und einer Multi Marker Verfolgung zu bestimmen. Die interne Kamera nutzten wir hingegen dazu die Position der Augen des Betrachters zu verfolgen. Nach der Abschätzung der relativen Position des Benutzerkopfes innerhalb der realen Welt konnten wir dann anhand dieser zwei Resultate die passende betrachterzentrierte Perspektive berechnen.

Zum Schluß führten wir eine Benutzerstudie durch um den herkömmlichen Ansatz der erweiterten Realität mit unserem, um die betrachterzentrierte Perspektive erweiterten, Ansatz zu vergleichen. Obwohl die Hardware die wir für den Prototyp verwendeten limitiert war, deuteten die Tests an, daß diese Technik sehr vielversprechend sein könnte für intuitive 3D Navigationsgeräte.

Acknowledgements

"Your friend is the man who knows all about you, and still likes you."

—Elbert Hubbard

First of all I would like to thank Prof. Dr. Jan Borchers for giving me the opportunity to conduct this work. Additionally, I thank Prof. Dr. Jan Borchers and Prof. Christian H. Bischof in advance for the survey of this work.

I especially want to thank Malte Weiss, who provided the topic for this thesis. Our many fruitful discussions had a great influence and were the basis for numerous design decisions of the practical part of this work.

I also want to thank all the reviewers of this thesis including Stefan Spieker, Malte Weiss and my father .

Last but not least I want to thank my family, my friends and – most important of all – my girlfriend Nicole for their patience and never-ending support. They were there for me to pick me up when things went wrong and I would not be at this point without them.

Thank you all!

Conventions

Throughout this thesis we use the following conventions.

Text conventions

Definitions of technical terms or short excursus are set off in orange boxes.

EXCURSUS:
Excursus are detailed discussions of a particular point in a book, usually in an appendix, or digressions in a written text.

Definition:
Excursus

Source code and implementation symbols are written in typewriter-style text.

`myClass`

The whole thesis is written in American English.

Chapter 1

Introduction

As a result of the constantly increasing power and decreasing size of mobile devices, they become more and more interesting as platform for interactive 3D applications. One of these applications is known as mobile Augmented Reality.

Azuma [1997] defined Augmented Reality (AR) as

- Combining real and virtual
- Interactive in real time and
- Registered in 3D

But most AR applications have the drawback of not incorporating the user's point of view and therefore tend to show the wrong perspective.

The idea behind our project therefore was to build a system, which behaves like a virtual window — hence we gave it the name VirlWind. It should adapt the shown perspective to the viewing position of the user and the position of the device itself. The hypothesis of our work is, that this leads to a more intuitive and natural possibility to experience three-dimensional environments.

Since it would destroy this natural feeling, we wanted to avoid special tracking hardware and any other auxiliary equipment. We also wanted to keep the software as independent of the platform as possible in order to maximize the group of possible users and to not be limited to any kind of hardware. Hence our goal was to build a mobile, robust and cost saving solution for intuitive 3D navigation.

Possible applications for such a system are

- *Navigate in computed tomography images*

  By moving the device forward and backwards one could navigate inside those images and mark all areas of interest. In this way a medical scientist could also point out certain suspicious spots and ask a colleague to have a look at them.

- *Support interior designers*

  Designer could show their customers the result of their work in place without altering the surrounding area. The customers then would be able to review the suggestions and leave comments on places they like or dislike.
  This would provide valuable feedback for the designers, because any critique would be directly connected to spatial positions.

- *Explore cultural heritage sites*

  The device would enable the user to see reconstruction of the original buildings similar to the solution Gleue and Dähne [2001] proposed. This approach could also be additionally extended to guide visitors to interesting spots by placing annotations at that positions.
  These comments would directly show the user where he has to go to see all attractions and could provide him with additional textual or multimedia information.

- *Games*

  As the Nintendo Wii recently proved the introduction of new input methods can positively influence the perception of games and can help open up new markets. Incorporating the spatial awareness and the user-centered perspective into mobile gaming devices could have a similar effect and add a whole new level to the mobile gaming experience.

1.1 Overview

In Chapter 2—"Fundamentals" we will give some theoretical background for a number of algorithms we used in the implementation of our system. The following Chapter — Chapter 3—"Related Work" — deals with other systems that exist in this research area and shows which applications they are used for.

Chapter 4—"System Design and Implementation" then presents our own system and shows in the first part how it was designed and which hardware was used, while the second part focuses on implementational details. Afterwards, in Chapter 5—"Evaluation", we will give an in-depth analysis of the performance of the system. In this context we will present the results of our user studies and analyze what can be concluded from them.

The last Chapter 6—"Summary and Future Work" then closes this thesis by summarizing our own work, pointing out the contributions to the research community and giving an outlook on what future research will be possible based on our results.

Chapter 2

Fundamentals

"If the brain were so simple we could understand it, we would be so simple we couldn't."

—Lyall Watson

Since great parts of this approach are based on image understanding and the analyses of pictures, we will first give a brief introduction into the functionality of the human visual system. Then we will show the algorithms, which form the basis for our tracking methods and are therefore essential for our work.

2.1 The Human Eye

Light passes through the pupil onto the retina

The human eye works by letting light pass through the pupil and then converting this light into electric impulses. This conversion into electric signals is done by the retina, which is a layer of light-sensitive cells situated at the rear of the eye. The impulses are then transferred to the visual part of the brain to be processed. Figure 2.1 shows the frontal view of a human eye.

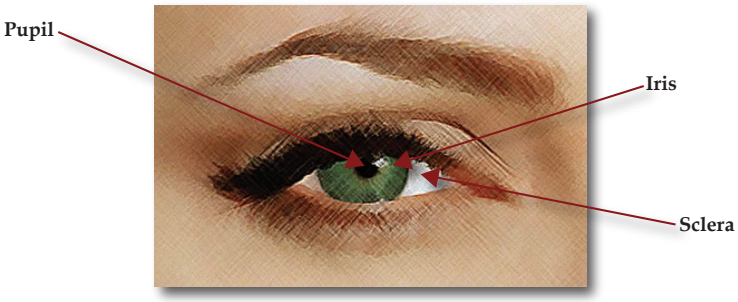

Figure 2.1: Frontal view of the human eye

Iris is used to adapt to amount of surrounding light

To vary the amount of light that passes through the pupil and to adapt to different light situations, the iris is used. The iris is the set of muscles which is located around the pupil. By relaxing or flexing these muscles the pupil can change its size and therefore control the amount of light.

Iris position used to improve tracking

Since the irides are prominent features in human faces, our eye tracking algorithm first tries to estimate possible eye regions and then to locate the position of the irides to stabilize the tracking result. If no iris is found in an area, we discard it to reduce the amount of false positives.

Distance estimation uses several methods

The eye is also used to estimate the distance of objects. To achieve this a number of methods are utilized

- `Stereo vision`
  Since the eyes are set a little bit apart from each other, they both return a different picture. The closer the object gets to the viewer, the more the two pictures differ.
- `Known objects`
  For objects the viewer has already seen before the size of the picture of the object on the retina is known.
- `Motion parallax`
  When we move our head, objects that are closer to us move at a greater speed than objects that are far away.

As we wanted to avoid that the user has to wear special glasses or a head-mounted display, we were not able to create stereo vision. So we resorted to simulating motion parallax to give the user the impression of distance and to make the displayed world feel more natural to the viewer.

2.2 Haar Classifiers

Haar classifiers are a statistical method to rapidly detect objects

The method used in our work has initially been proposed by Viola and Jones [2001] and was then extended by Lienhart and Maydt [2002]. It uses a statistical approach to rapidly detect objects. For this a classifier has to be trained with a set consisting of positive (images which contain the object class of interest) and negative (images which don't contain the object class of interest) samples of a fixed size. The training software searches for distinctive simple Haar-like features, which can be used to identify the object. Each of these features is described by one of 14 templates (see Figure 2.2), its position inside the sample and its size.

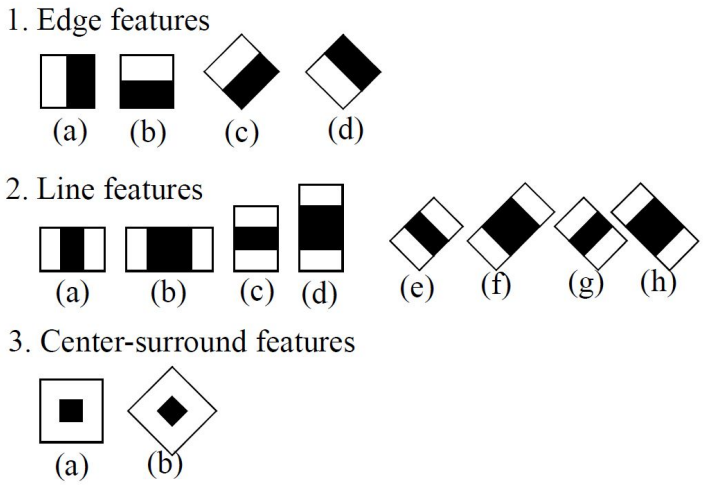

Figure 2.2: Extended set of Haar-like features which are used in the recognition process, from Lienhart and Maydt [2002]

Computation of the feature values

The feature values are hereby computed in the following way :

- compute the pixel sum over the black area and the pixel sum over the whole template,
- give these sums opposing signs and
- compute the weighted sum of both, with weight values inversely proportional to their area size

For example, the value of feature 2(a) of Figure 2.2 can be computed as

$$feature\_value = -1 \cdot Areasum_{whole} + 3 \cdot Areasum_{black}.$$

, where the weights are given by

$$w_0 = 1/\frac{\text{whole area}}{\text{whole area}} = 1$$
$$w_1 = 1/\frac{\text{black area}}{\text{whole area}} = 1/\frac{1}{3} = 3$$

and the minus sign is a result of the second rule.

Weak classifiers are combined into a cascade

In the last step these "weak" classifiers are combined into more complex boosted ones to form a cascade of classifiers with the simple ones at the beginning (see Figure 2.3).

Detection process would be slow without optimizations

The detection process then works by sliding windows with different sizes over the input image and determining for every step if an object is visible or not. Without optimizations this process would be too slow, so Viola and Jones [2001] proposed the use of an intermediate representation of the original image, the Integral Image (II).

$$II(X,Y) = \sum_{x \leq X,\, y \leq Y} I(x,y)$$

Here $I(x,y)$ stands for the pixel value of the original image at position (x,y).

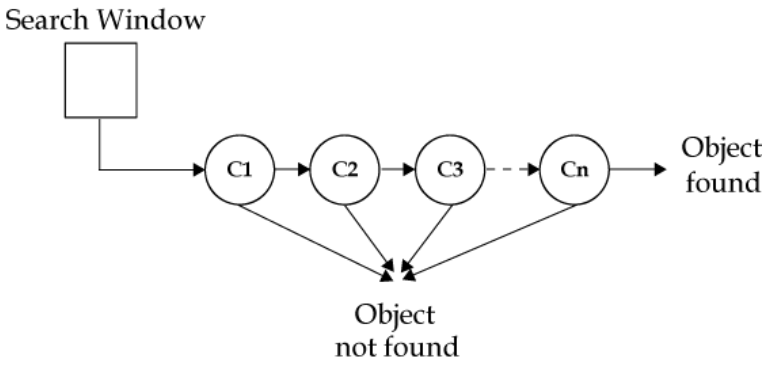

Figure 2.3: The cascade of boosted Haar classifiers speeds up the recognition by discarding false regions at a very early stage

With this representation it is possible to compute the sum over every rectangle with just four array references. For example, the sum of a rectangle at point (x_0, y_0) and size (w, h) can be computed as

$$II(x_0+w, y_0+h) - II(x_0+w, y_0) - II(x_0, y_0+h) + II(x_0, y_0).$$

Due to the cascades, false image regions are rejected at a very coarse scale. This, combined with the Integral Image representation, makes sure that this approach achieves real-time performance on modern day systems.

2.3 Template Matching

Template is matched based on similarity values

Template matching is an image processing technique which is used to find a predefined sub-image (a so-called template) in a test image. The basic principle is, to move the template over the test image and compute a measure of similarity for the underlying area of the image. Examples for similarity measures are squared differences and cross correlation.
Then the algorithm finds the maximal or minimal similarity value depending on which method was chosen. If the value is above or below a certain predefined threshold, the

algorithm presumes that the template was found inside the test image and the corresponding position is returned (see Figure 2.4).

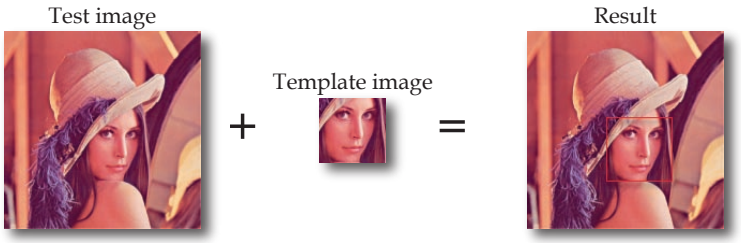

Figure 2.4: Example for one template matching pass. The input image is searched for the template in the middle and the region with the highest similarity value is marked in the output image.

Faster template matching method is used

Usually this technique is very slow, since it has to compute similarity values for the whole image. For that reason we chose to include an extended matching algorithm. The fast template matching method used in our work was implemented by Tristen Georgiou and works by first creating an image pyramid for the template and the test image.

Image pyramid helps speeding up the matching process

Here level 0, the base of the pyramid, represents the original image. Each subsequent level i is computed by applying Gaussian filtering with a window size of 5×5 to level i and then discarding even rows and columns. In this way the image resolution is halved in every step. 2.5 shows an example of this downscaling process.

Figure 2.5: Different resolutions of an image pyramid, which is used to speed up the matching algorithm

GAUSSIAN FILTERING:
The Gaussian filter is a low-pass filter, which is used to reduce noise and detail in images. It works by computing the weighted average of all samples within a predefined window around each pixel. In contrast to mean filtering, where the weights are chosen to be $\frac{1}{\#\text{ of samples}}$, in this case the weights are computed according to the two-dimensional Gaussian function, which is the density of the normal distribution.

$$G(x,y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}$$

Here x and y are the horizontal and vertical distance to the origin and σ is the standard deviation of the Gaussian distribution. Although the Gaussian function would be non-zero on every point of the image, points outside the window are not considered in the calculation.

Definition:
Gaussian filtering

The algorithm then proceeds by running template matching on the smallest template and the smallest test image. If a match is found, the position in the original image is computed and this area is again compared with the original template.

Detection works from the smallest to the biggest representation

By first searching the downscaled image many regions can be discarded very early and the running time can be reduced significantly.

2.4 Marker Tracking

This section describes the marker tracking algorithm by Kato and Billinghurst [1999], which is implemented in the ARToolkit library and can be downloaded from the Human Interface Technology Lab. We chose the ARToolkit library since it is fast, returns acceptable results, is free for non-commercial use and is available for Mac OS and Windows. Additionally it includes functions to capture pictures from connected cameras, which we used on Mac OS.

The marker tracking algorithm starts by thresholding the grayscale picture that should be searched. Grayscale pictures are used, because they can be processed faster.

Line fitting algorithm returns potential regions

The algorithm then searches for regions whose outline contours can be fitted by four line segments and saves their intersection positions. By normalizing the sub-images inside the found regions and comparing them to predefined marker template images, the algorithm can determine which markers were actually found and can discard falsely found regions. Figure 2.6 gives an overview of the various steps of the algorithm.

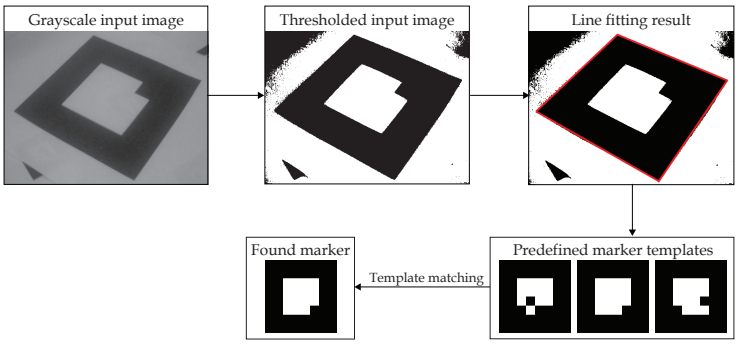

Figure 2.6: Example pass of the marker tracking algorithm used by ARToolkit. The outlines of the found marker are then used to estimate the relative camera position

To estimate the position of the camera (see Figure 2.7), the algorithm then has to know the size of the markers to build the transformation matrix T_{cm}.

Known marker size is used for estimation

$$\begin{bmatrix} X_c \\ Y_c \\ Z_c \\ 1 \end{bmatrix} = \begin{bmatrix} V_{11} & V_{12} & V_{13} & W_x \\ V_{21} & V_{22} & V_{23} & W_y \\ V_{31} & V_{32} & V_{33} & W_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} X_m \\ Y_m \\ Z_m \\ 1 \end{bmatrix} \tag{2.1}$$

$$= \begin{bmatrix} & V_{3\times3} & & W_{3\times1} \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} X_m \\ Y_m \\ Z_m \\ 1 \end{bmatrix} \tag{2.2}$$

$$= T_{cm} \cdot \begin{bmatrix} X_m \\ Y_m \\ Z_m \\ 1 \end{bmatrix} \tag{2.3}$$

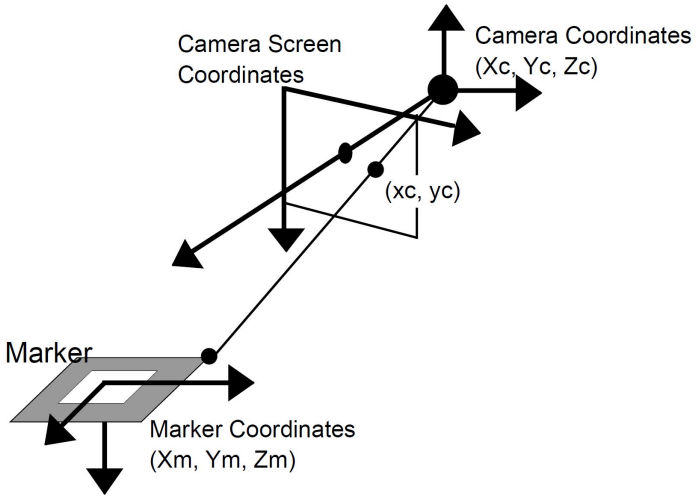

Figure 2.7: Relationship between the marker coordinates and camera coordinates in ARToolkit, from Kato and Billinghurst [1999]

We also need the perspective projection matrix P, which is obtained in the calibration process of ARToolkit.

$$P = \begin{bmatrix} P_{11} & P_{12} & P_{13} & 0 \\ 0 & P_{22} & P_{23} & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} f_x & 0 & x_0 & 0 \\ 0 & f_y & y_0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.4}$$

$$\text{so that} \quad \begin{bmatrix} hx_c \\ hy_c \\ h \\ 1 \end{bmatrix} = P \cdot \begin{bmatrix} X_c \\ Y_c \\ Z_c \\ 1 \end{bmatrix} \tag{2.5}$$

Here f_x and f_y are the focal lengths in x and y direction in pixel units and (x_0, y_0) is the principal point (see Appendix A—"The Pinhole Camera Model"). If now two parallel sides of a marker are projected to the image, they fulfill the following two equations

$$a_1 \cdot x + b_1 \cdot y + c_1 = 0 \quad a_2 \cdot x + b_2 \cdot y + c_2 = 0 \tag{2.6}$$

The parameters of these equations were already found previously in the line fitting process. By substituting x_c and y_c for x and y and combining equations (2.6) with equation (2.5) we get

$$a_1P_{11}X_c + (a_1P_{12} + b_1P_{22})Y_c + (a_1P_{13} + b_1P_{23} + c_1)Z_c = 0 \quad (2.7)$$

$$a_2P_{11}X_c + (a_2P_{12} + b_2P_{22})Y_c + (a_2P_{13} + b_2P_{23} + c_2)Z_c = 0 \quad (2.8)$$

Direction vectors have to be normalized to be perpendicular

The normal vectors of these two planes shall now be given as n_1 and n_2. That means, we can compute the direction vector as $u_1 = n_1 \times n_2$. Repeating this calculation for the other two sides of the marker yields u_2. To correct processing errors, we choose two perpendicular vectors v_1 and v_2 which lie in the same plane as u_1 and u_2 and which include both vectors (see Figure 2.8).

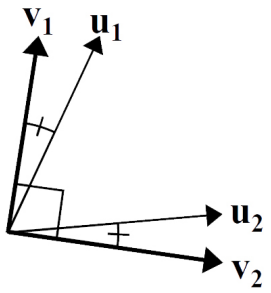

Figure 2.8: The two perpendicular unit direction vectors v_1 and v_2, which are computed from the vectors u_1 and u_2, from Kato and Billinghurst [1999]

The rotation component $V_{3\times3}$ of T_{cm} can then be computed as $({v_1}^T, {v_2}^T, (v_1 \times v_2)^T)$. By using equation (2.1) and equation (2.5) and inserting the coordinates of the marker vertices and their respective image coordinates, we obtain eight equations. From these equations we can compute the translation component $W_{3\times1}$.

Recursive optimization method improves the result

With the transformation matrix T_{cm} we can then transform the marker coordinates, compare the result to their image coordinates and optimize the rotation component in T_{cm}. This process is iterated a few times to improve the accuracy of the resulting transformation matrix. The translation component is reestimated in every iteration with the method explained above.

Chapter 3

Related Work

"Science never solves a problem without creating ten more."

—George Bernard Shaw

There are various approaches that try to solve the problem of navigating inside virtual environments.
At first we will take a look at Augmented Reality. In the last years AR has become a well researched field and some applications like, e.g., virtual advertisements in soccer broadcastings are now part of our everyday life. For this thesis we will focus on solutions that are mobile.

In the next section we will then show some devices that are aware of their own spatial position and use this knowledge to determine the content that is displayed. In this way they connect the real world with the virtual world and offer a window-like view. By translating the device the user can change the current view to reveal other parts of the virtual environment.

At last we will present a selection of systems that incorporate the position of the user to offer the appropriate perspective. Some of them use shutter glasses and compute a different picture for each eye to intensify the three-

dimensional effect. We will finish this chapter with a comparison of the different projects, which will also point out the differences to our approach.

3.1 Mobile Augmented Reality

Symbian port of ARToolkit determines the position of the device

As said in the introduction, there are many approaches for Augmented Reality on mobile devices. One of these approaches was made by Henrysson and Ollila [2004]. In their work they ported the ARToolkit to Symbian cell phones. This framework has then been used in several projects like AR Tennis [Henrysson et al., 2006], where two users play a game of virtual tennis with their cell phones and a set of markers is used to position the tennis court (see Figure 3.1).

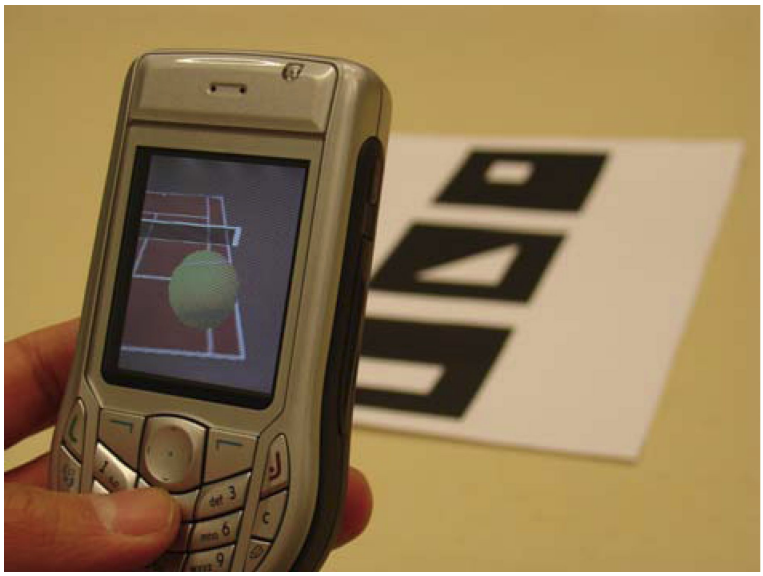

Figure 3.1: Scene from the AR Tennis project where a marker is used to determine the position of the mobile phone, from Henrysson et al. [2006]

Guidance systems popular application for mobile Augmented Reality

Guidance systems also are a field of application for mobile Augmented Reality systems. Schmalstieg and Wagner [2005] proposed such a system, which provides relevant in-

formation in dependence of the current context. To accomplish this they use a low cost PDA (Personal Digital Assistant) with an external camera and perform fiducial marker tracking. The 3D content that is displayed here is stored on a database server and is transferred to each device on demand. A picture of the device and a snapshot of the display can be seen in Figure 3.2.

Figure 3.2: Screenshot of the guidance system by Schmalstieg and Wagner [2005] which uses fiducial markers

Another approach was presented by Jung et al. [2007]. Their guidance system uses a markerless place recognition technique and scene structure awareness to determine the current position and to align the virtual content with the live video. The device can then present additional information about the places in sight and adjust the position of this information to the current camera picture. Figure 3.3 shows the result of this image-based modeling method.

Scene structure awareness used to determine position

LAMP3D (Location-Aware Mobile Presentation of 3D content) by Burigat and Chittaro [2005] is also a mobile device to guide tourists and provide additional information. This system offers three modes of operation. In `GPS-based navigation` the data delivered by the GPS module is used to estimate the position of the user and display the appropriate VRML (Virtual Reality Modeling Language) content (see Figure 3.4). The second mode, `manual navigation`, can be used to explore the city off-line or to

Virtual tour to show places of interest to visitors

plan a route and the last mode, `replayed navigation`, plays back a virtual tour based on previously recorded location and orientation information.

Figure 3.3: Input image after the structure is analyzed and the corresponding content is overlaid, from Jung et al. [2007]

Figure 3.4: Guidance system which shows 3D models and additional information based on the current location, from Burigat and Chittaro [2005]

Reconstructions are overlaid on live video

Gleue and Dähne [2001] tried a different approach for the ARCHEOGUIDE (Augmented Reality-based Cultural HEritage On-site GUIDE) project. Their system presents virtual reconstructions of buildings within the real environment. Therefore it uses a backpack with a mobile computer

and a HMD (Head-Mounted Display). The user position is tracked by a GPS (Global Positioning System) and an electronic compass is used to determine the orientation of the user's head. In Figure 3.5 a user with all required equipment is shown.

Figure 3.5: Equipment needed for the ARCHEOGUIDE project. The user has to wear a backpack and a HMD, from Gleue and Dähne [2001]

Guide displays auxiliary content on objects in view

Schiele et al. [2001] presented a similar attempt for a virtual museum guide. The system also consists of a HMD and a backpack with a mobile computer. In the first step a database is filled with audio and video content that should be played if a certain object is in the field of view. This can either be done by the user itself or, for example, by an museum employee. Whenever one of the trained objects comes into the sight of the camera, the previously stored content is overlaid on the current live image.

Virtual walls comply with real walls

For ARQuake, Thomas et al. [2002] extended the game Quake by id Software to an Augmented Reality application. Therefore the levels of the game where modeled after real locations so that walls or ceilings don't have to be drawn and the levels inside the game comply to the real environment. Augmented Reality content like enemies and the GUI is overlaid to give the user the impression of a real game (see Figure 3.6). Tracking in this system is implemented with a combination of a GPS, a compass (for outdoor tracking without buildings in sight) and fiducial markers (for outdoor tracking near buildings and for indoor tracking). The hardware configuration of ARQuake corresponds to the ARCHEOGUIDE system and is therefore too heavy to be used over a long period of time.

3.2 Spatially Aware Devices

Boom mounting alleviates the weight of the device

There are also some systems which incorporate the position of the device, but are not mobile. The Boom Chameleon by Tsang et al. [2002] is such a system. It consists of a boom mounted LCD (Liquid Crystal Display) panel with touchscreen functionality, a computer, a microphone and a pair of speakers (see Figure 3.7). From the angles of the various joints of the boom the system can determine the position and orientation of the LCD panel and display the appropriate view onto the scene. Then the user can make annotations or record whole review sessions for later playback.

Figure 3.6: Scene from ARQuake which is shown on the head mounted display, from Thomas et al. [2002]

Figure 3.7: With the Boom Chameleon the user can navigate around the 3D model and place comments, from Tsang et al. [2002]

Volume slices are displayed based on the orientation of the device

Another approach was made by Hirota and Saeki [2007]. The Cross-section Projector is used to view volume data in an intuitive way. Here multiple laser displacement sensors are used to determine the distance and orientation of the screen plane (see Figure 3.8). A liquid crystal projector is then used to generate the image on it. The system offers two different modes, depending on how the user moves the screen plane. If the screen plane is moved fast enough, a 3D image of the volume data is displayed, otherwise it shows single slices of the data.

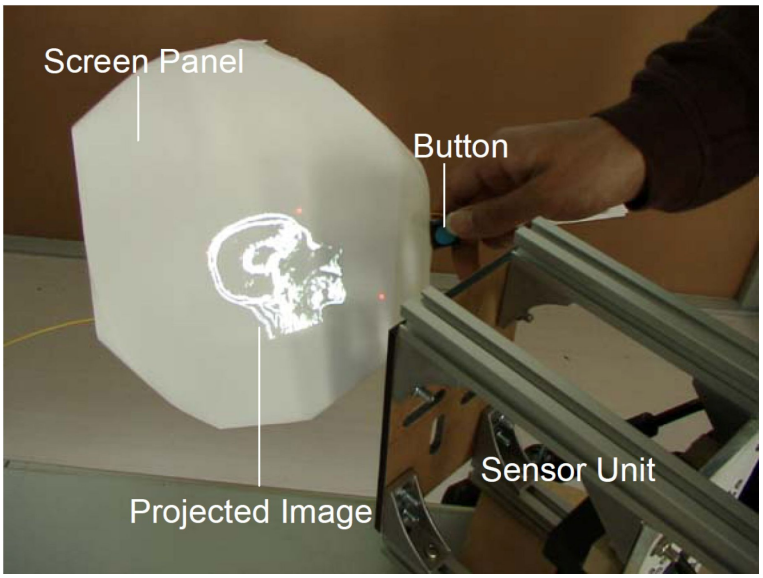

Figure 3.8: Screen panel showing a slice of the volume data according to the orientation of the screen plane, from Hirota and Saeki [2007]

Moving the device translates the window into the the virtual world

Chameleon, the prototype presented by Fitzmaurice [1993], was one of the first approaches for the so-called peephole metaphor. In this metaphor a spatially aware mobile device serves as connection between the real and the virtual world. The prototype consists of a small LCD monitor with an attached button and a six degree input device, a Silicon Graphics workstation and a camera, which captures that part of the large workstation screen which is currently visible on the mobile device. Moving the monitor horizontally or vertically adjusts the ”peephole” accordingly in the

virtual world, while a translation forwards or backwards equates to zooming in or out of the picture. The relationship between the virtual workspace and what is displayed on the screen is shown in Figure 3.9.

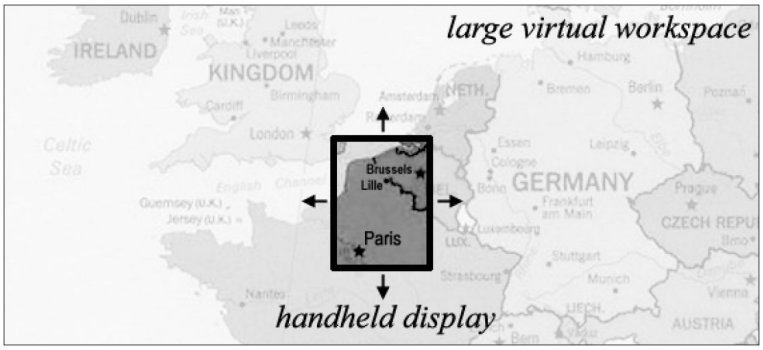

Figure 3.9: Relationship between the peephole and the virtual workspace, which is in the background, from Yee [2003]

Virtual workspace can be much larger than the current visible portion

Yee [2003] in his proposal presented two prototypes. The first prototype was built using a Handspring Visor and implements a two-dimensional "peephole" without zooming. An example application for it is the Peephole Doodle Pad, which can be seen in Figure 3.10. In this picture a circle larger than the display size is drawn by moving the Visor simultaneously. The second prototype was extended with a tracker which is able to track three dimensions.

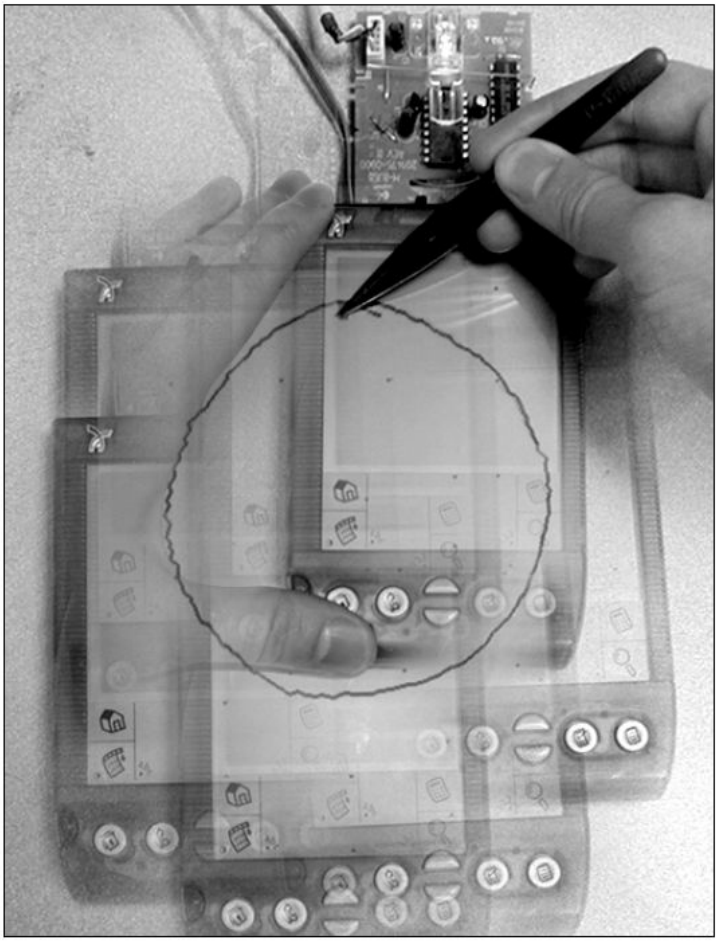

Figure 3.10: The Peephole Doodle Pad in this example is used to draw a circle which is larger then the display onto the virtual workspace, from Yee [2003]

3.3 User-centered Perspective

Movement of the head changes the camera view

In this section we will present some systems that incorporate the position of the user to provide the appropriate perspective. Cooperstock et al. [1995] proposed such a system to aid video conferences and give attendees the feel of fully participating rather than only watching the meeting. Hence the system tries to track the attendees head by comparing the current image to a reference image and then adjusts the remote camera via an attached motor.

Previous knowledge about the setup and tracking results used to determine correct perspective

The E-conic project was introduced by Nacenta et al. [2007]. It is an interface to dynamically correct the perspective of displayed content in multi-display environments. For this purpose the user's head is tracked with an ultrasonic three degree of freedom tracking system, while mobile devices are tracked with a combination of ultrasonic and inertial tracking. All information about static displays like location, size and orientation are stored in a geometry server, which then enables the system to calculate the perspective correction for every display and every mobile device (see Figure 3.11).
The correct perspective can now be determined by putting all objects that should be displayed onto a virtual plane that is perpendicular to the user's viewing direction and projecting this plane onto the displays.

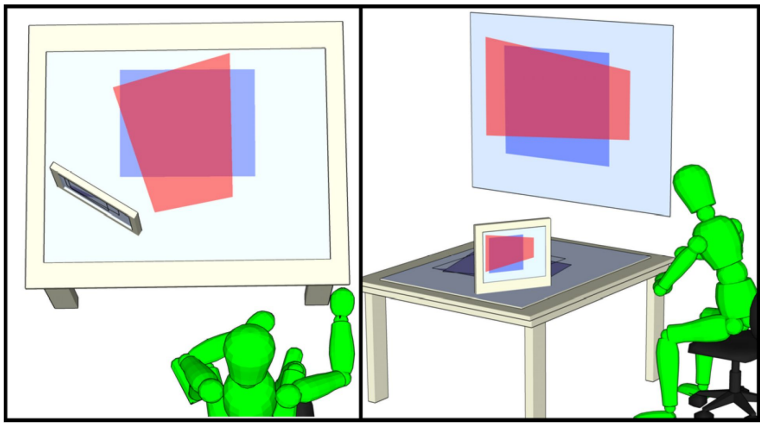

Figure 3.11: After the determination of the user's position relative to the monitors the content can be displayed in the correct perspective, from Nacenta et al. [2007]

Head-coupled display as replacement for real window

Another HCD (Head-Coupled Display) solution was proposed by Radikovic et al. [2005]. Their artificial window was intended for windowless environments as a substitute for a real window. The system uses a camera and head-tracking software to determine the position of the user in front of the screen and to adapt the perspective.

Creation of the picture is then done with image-based rendering methods to achieve a realistic view in real-time.

Shutter glasses provide stereo vision

At the 1992 SIGGRAPH the first CAVE (Cave Automatic Virtual Environment) was presented by Cruz-Neira et al. [1992] (also see [Cruz-Neira et al., 1993]). A CAVE is built up from five projection screens that form a room. The pictures are created by high-resolution projectors, which project the images onto mirrors from where they are reflected onto the screens. Tracking of the user's head is done with electromagnetic sensors that are mounted to Stereoscopic LCD shutter glasses. The glasses are needed to separate the images for the left and right eye. From the position of the user's head the correct user-centered perspective can be computed and the user can move freely inside the virtual world.

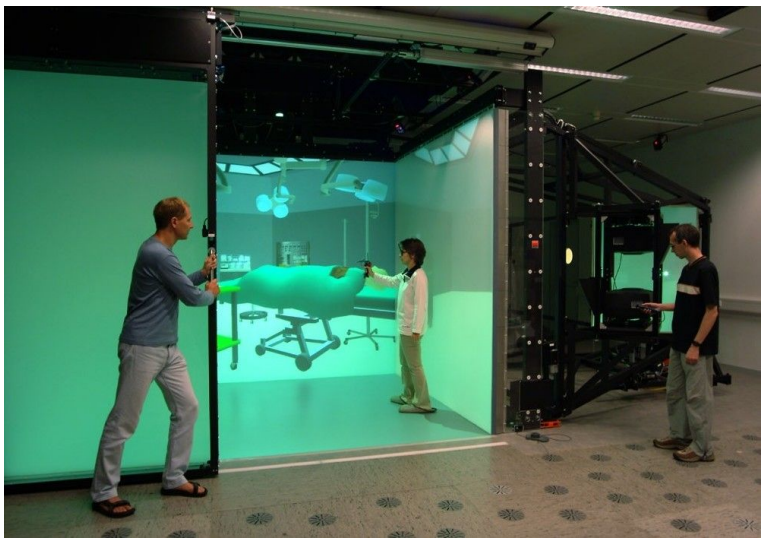

Figure 3.12: Photograph of the CAVE at the RWTH Aachen, which can be used to simulate virtual environments

More affordable systems with one projection screen

With the ImmersaDesk and the Infinity Wall Czernuszenko et al. [1997] presented two derivatives of the CAVE that are much more affordable. The ImmersaDesk consists of a rear-projected screen, which is mounted at an angle of 45 degrees. Because of the rotation the user is able to look forward and downward. To perceive three-dimensional pictures the user here also has to wear shutter glasses. The position and orientation of the head of the primary user are tracked and the perspective is adjusted accordingly.
In contrast to the ImmersaDesk the Infinity Wall was built with a much larger screen, as it was intended for presentations in front of large groups. Like the ImmersaDesk the perspective of the Infinity Wall can be controlled by either the primary shutter glasses with the attached tracking system or by a wand.

Wii remote used to track LED points

In 2007 Lee presented a head-tracking approach which utilizes two infrared LEDs (Light-Emitting Diode) which are mounted to a pair of glasses and the Nintendo Wii remote. The Wii remote is connected to a computer via Bluetooth and is placed above or below a screen. Tracking of the two LEDs is accomplished with the internal 1024×768 infrared camera of the Wii remote which also contains hardware to track four of these infrared dots at 100Hz simultaneously.

3.4 Comparison

Lightweight and inexpensive solutions for mobile AR

Most of the approaches we presented in the first section of this chapter utilize a mobile phone or a PDA to display the virtual content and to calculate the position of the device itself (Henrysson and Ollila [2004], Schmalstieg and Wagner [2005], Jung et al. [2007], Burigat and Chittaro [2005]). Those devices are very lightweight and therefore allow the usage over a long period of time. They are also cost saving, since most persons already own a mobile phone or a PDA. But due to the limited processing power they would still be too slow to achieve eye tracking at interactive frame rates without additional hardware. This means that they can not yet offer an user-centered perspective.

Head-mounted displays can cause cybersickness

The other projects in this section follow a different approach. They use an additional computer in a backpack for the processing part and display the resulting image on a head-mounted display (Gleue and Dähne [2001], Schiele et al. [2001], Thomas et al. [2002]). Due to the use of the HMD this systems offer an user-centered perspective but are also costly. Besides that, the usage of head mounted displays can lead to motion sickness or cybersickness (see McCauley and Sharkey [1992] or Joseph J. LaViola [1999]), which occurs if the visually perceived motion does not correspond to the motion observed by the other sense organs. The resulting symptoms of this are dizziness and nausea, which can significantly affect the usability of the device. Mobile devices that offer user-centered perspective are not affected by this phenomenon as much as head-mounted displays are, since the user is not forced to look at the screen all the time.

Viewing position does not influence the visable area

In the second section we showed some spatially aware devices that, in their own way, also create a window into a virtual world. None of them however offers an user-centered perspective, so the displayed content only depends on the position and orientation of the device.
The first two approaches we presented in this section were both not mobile and did use special hardware to achieve the tracking (Tsang et al. [2002], Hirota and Saeki [2007]). Therefore they did not meet our requirements. The other two projects which we showed in this section are representatives of the peephole metaphor (Fitzmaurice [1993], Yee [2003]). Both of them are mobile and do not require the use of special hardware but they do not offer an user-centered perspective either.

Almost all systems that provide the correct perspective are stationary

The third section then presented some projects which explicitly incorporate the position of the user to calculate the user-centered perspective (Cooperstock et al. [1995], Nacenta et al. [2007], Radikovic et al. [2005], Cruz-Neira et al. [1992], Czernuszenko et al. [1997], Lee). Unfortunately only one of them is mobile and most of them are very costly. Furthermore the system by Nacenta et al. [2007] is only partially mobile, since the tracking hardware has to be set up correctly. Table 3.1 shows a comparison of all different approaches.

Table 3.1: Comparison of the different approaches

Paper	Mobile	User-centered perspective	Cost saving	Uses special hardware
Henrysson and Ollila [2004]	√	×	√	×
Schmalstieg and Wagner [2005]	√	×	√	×
Jung et al. [2007]	√	×	√	×
Burigat and Chittaro [2005]	√	×	√	×
Gleue and Dähne [2001]	√	√	×	√
Schiele et al. [2001]	√	√	×	√
Thomas et al. [2002]	√	√	×	√
Tsang et al. [2002]	×	×	×	√
Hirota and Saeki [2007]	×	×	×	√
Fitzmaurice [1993]	√	×	√	×
Yee [2003]	√	×	√	×
Cooperstock et al. [1995]	×	√	×	√
Nacenta et al. [2007]	√	√	×	√
Radikovic et al. [2005]	×	√	√	×
Cruz-Neira et al. [1992]	×	√	×	√
Czernuszenko et al. [1997]	×	√	×	√
Lee	×	√	√	√

Chapter 4

System Design and Implementation

"Everything has beauty, but not everyone sees it."

—Confucius

In the previous chapter we introduced some research projects which deal with Augmented Reality and user-centered perspective. One of the problems combining these two areas is that the device can not just display the camera image from a static perspective. The perspective has to be readjusted every time the user changes his point of view.

As we did not see any way how to achieve this on our mobile device, we instead resorted to showing a complete virtual scene in which we tried to reproduce the real world as accurately as it was possible on this limited hardware. We also had to find a balance between visual realism and usability, as especially the eye tracking algorithm takes up a lot of resources and we thought that smooth interaction with the device was crucial for our approach.

The contents of this chapter show which hardware we used, provide a detailed view on how the software was implemented and give a description of the used methods. For this we concentrate on the essential parts of the software and omit trivial things like setting up windows or creating buttons.

4.1 Hardware Setup

Tablet computer with two connected cameras

As platform for our application we chose the Modbook by Axiotron (see Figure 4.1). The Modbook is a slate-style tablet Mac, which runs Mac OS X v10.5 Leopard. It is equipped with a 2.4GHz Intel Core 2 Duo CPU, 2GB of 667MHz DDR2 SDRAM, an Intel GMA X3100 graphics chipset and a built-in iSight camera that we used for eye tracking. The widescreen display of the device measures 13.3 inch in diagonal and has a native resolution of 1280 by 800 pixels. As the base system for the Modbook is a Apple MacBook computer, it unfortunately weighs 2.5kg.

Figure 4.1: The Modbook by Axiotron, which we used as basis for our prototype, from Axiotron

Modified notebook bag used as case for Modbook

For marker tracking we additionally connected an external iSight camera. Since we wanted to take the users the fear of dropping the device, we also built a special case with two straps at the side to provide a better grip. The external camera was then mounted perpendicular at the center of the backside of this case.

Mouse mounted to backside of device

Alongside the camera we placed a mouse, which gave the users the ability to freeze the current view and start an annotation without loosing the control over the device. Figure 4.2 shows a picture of this construction.

Change of markers improved the recognition rate

As markers we used the patterns that were provided in the examples of the ARToolkit Patternmaker by Johnson et al.. They achieved a higher recognition rate and less errors than the patterns that come with ARToolkit and could be used at higher distances. We used ten markers with the size of $135mm \times 135mm$ and five with the size of $67.5mm \times 67.5mm$. Then we glued each one of them to a piece of cardboard to avoid wrinkles and to give them more weight.

4.2 Software Implementation

C++ used for compatibility to multiple platforms

The software was meant to be multi-platform, so we used C++ as the programming language in conjunction with QT by Trolltech, to build the GUI (Graphical User Interface) and access OpenGL, which we used to render the virtual scene. For image processing we included OpenCV by Intel, because it offers many useful functions and its examples were a good starting point. Marker tracking is done with ARToolkit, which was originally developed by Kato and Billinghurst [1999] and can now be found at the Human Interface Technology Lab. We additionally extended it with a multi-marker recognition algorithm to improve stability.

Figure 4.2: The case we built for the Modbook. The straps on the sides should give users a good grip of the device

Camera capturing works platform dependent

The capturing of the camera images is handled differently depending on which platform the software runs on. On Macintosh we use the video functionality of ARToolkit and one of QT's built-in timers for camera capturing, whereas on Windows the videoInput library by Watson and a dedicated camera capture thread (`camthread.cpp`) are used. We restricted the size of the captured images on both platforms to 320×240 pixels to speed up the recognition process.

Operation modes for different application possibilities

VirlWind supports different operation modes, which can be initialized by calling the application with the corresponding command line parameter :

- `''-test''` : Measurement mode
  Displays a model of a box with pits and bulges, where we hid letters which the participants of the user studies had to find
- `''-ar''` : Augmented Reality mode
  Displays the same model as the "Measurement mode", but with exchanged letters and with the current picture of the marker tracking camera as background

- ''-annotate'' : Annotation mode
  Displays a faulty model of a car to test the annotation feature of the software
- ''-volume'' : Volume mode
  Displays a computed tomography volume of a head

If no parameter is given, the software starts into the "Measurement mode". At start the software switches to fullscreen and displays a virtual scene depending on which operation mode was chosen (see Figure 4.3).

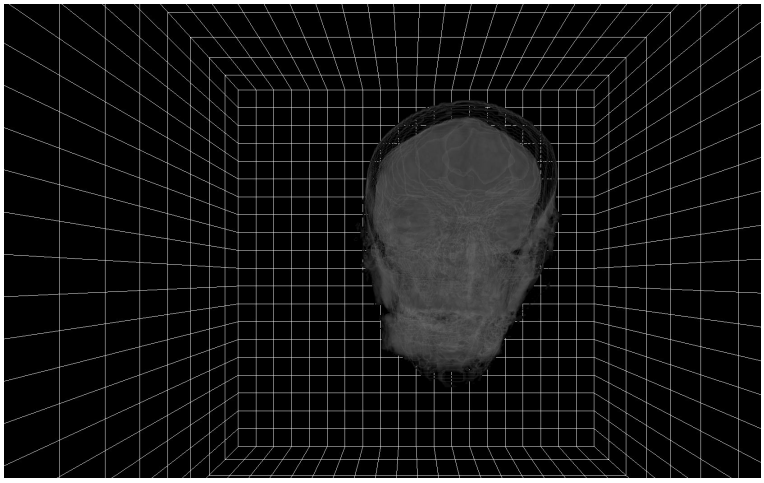

Figure 4.3: Screenshot of VirlWind in "Volume mode"

First two cameras used for tracking

During the initialization phase VirlWind searches for connected cameras and if two or more are found, the first two of them are used as input for the eye tracking algorithm and the marker tracking algorithm. Whenever there are new captured images, they are fed into the tracking algorithms. If no eyes are found in the captured image a warning sign is displayed at the top of the screen as can be seen in Figure 4.4.

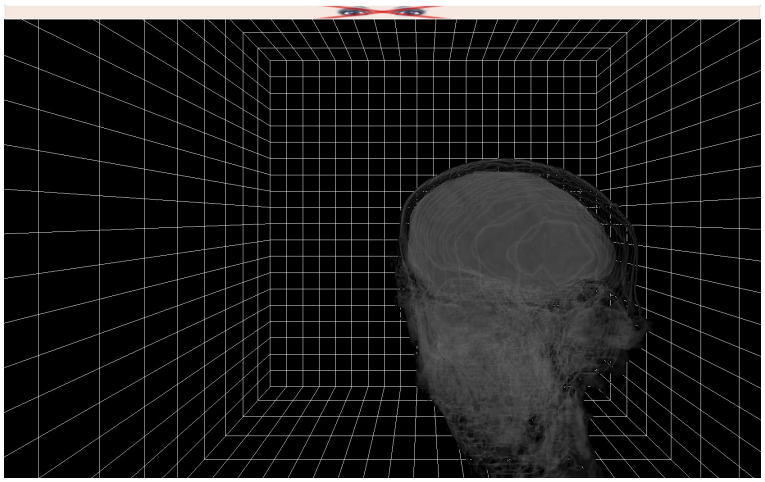

Figure 4.4: If no eyes are found, VirlWind displays a warning message on the top side of the window

Other displaymode used for configuration and information output

We also included a display mode which shows some additional information. Here the main window (`mainwindow.cpp`) is split up into an status area, the OpenGL rendering area (`glwidget.cpp`), the eye tracking window (`facetracker.cpp`) and a window for marker tracking (`markertracker.cpp`). This is shown in Figure 4.5. It can be accessed by clicking the red square in the upper left corner of the OpenGL window, which appears when you hover above it. The status area that is displayed consists of

- a label showing the eye tracking status,
- a label showing the estimated translation of the face to the camera,
- a label showing the number of currently captured images per second,
- a button to quit the program,
- a button to save all previously made annotations and
- a button for further options like disabling/enabling tracking or adjusting the number of slices for volume rendering.

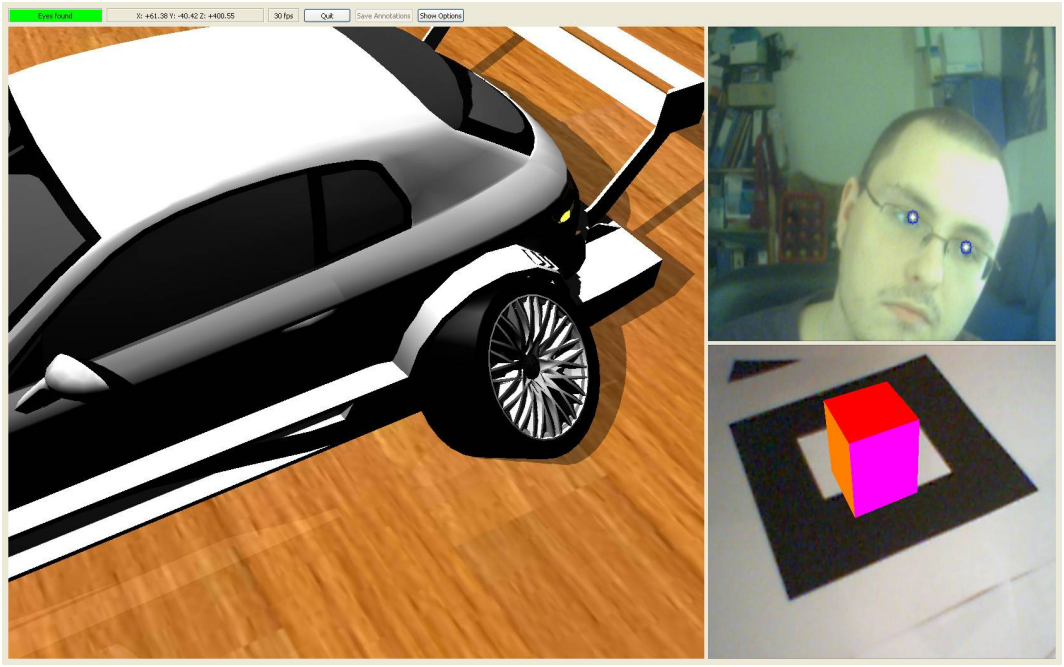

Figure 4.5: Screenshot of VirlWind with additional information, like the eye tracking status and the estimated translation of the user's head

This mode also gives the user the opportunity to switch camera inputs by clicking on one of the two camera windows.

In the following subsection we will give an in detail view of the various steps that our system performs to display the appropriate view.

4.2.1 Eye Tracking

At first we have to search for possible eye regions. In these areas we will later search for irides to stabilize the result and reduce the amount of false positives.

Finding the Eye Regions

Different classifiers for eye region detection

Eye tracking is done in the corresponding widget `facetracker.cpp`. It uses three different Haar classifiers in the process of finding the eye regions

- `haarcascade_frontalface_alt2.xml` : Face classifier, which is bundled with OpenCV
- `haarcascade_eye.xml` : Eye classifier, which was created by Shan
- `eyes22x5.xml` : Eye-pair classifier, which was created by Castrillón Santana et al. [2007]

Possible eye regions searched for irides

Whenever a new image `camimg` is captured

```
void detect_and_draw( IplImage* camimg )
```

is called. It first creates a smaller gray version of the image to speed up Haar detection. The algorithm then performs three steps to detect the eye regions. If one of these approaches succeeds the resulting areas are checked with the `findIris` function and, if an iris is found, new templates of these eye regions are created. Otherwise the algorithm proceeds with the next step.

1. First the algorithm uses the face classifier to detect the largest face in the image. If a face is found, it copies the upper left and upper right quarters to own buffers and searches these with the eye classifier.
2. If no face is found in the first step, but there exist templates for the eye regions, template matching is done.

3. If still no eyes were found, the eye pair classifier is used to identify the largest pair of eyes in the image.

Figure 4.6 depicts the whole process of finding the eye regions.

Finding the Iris

After a possible eye region is found, the algorithm tries to locate an iris inside this area. For this the function

```
bool findIris( IplImage *src, CvPoint
      *eyePoint, int rectSize )
```

is called. As arguments the function takes the `BGR`-format input image, the estimated eye position `eyePoint` and `rectSize` so that (`eyePoint.x` $\pm$ `rectSize`, `eyePoint.y` $\pm$ `rectSize`) is the search window. If an iris is found, `findIris` succeeds and returns the corresponding image point in `eyePoint`.

Equalize histogram to increase image contrast

The algorithm starts by separating the red channel of the input image's search window. We choose the red channel, because it proved to be the most responsive to skin color. Then the histogram of this image is equalized and a median filter with a window size of 15×15 is applied.

Definition: *Median filtering*

MEDIAN FILTERING:
The median filter is a spatial filter, which is normally used to reduce noise in images. Therefore a window with an odd number of samples is defined centered around each pixel. The pixel value is then replaced by the median value of all samples in this window.

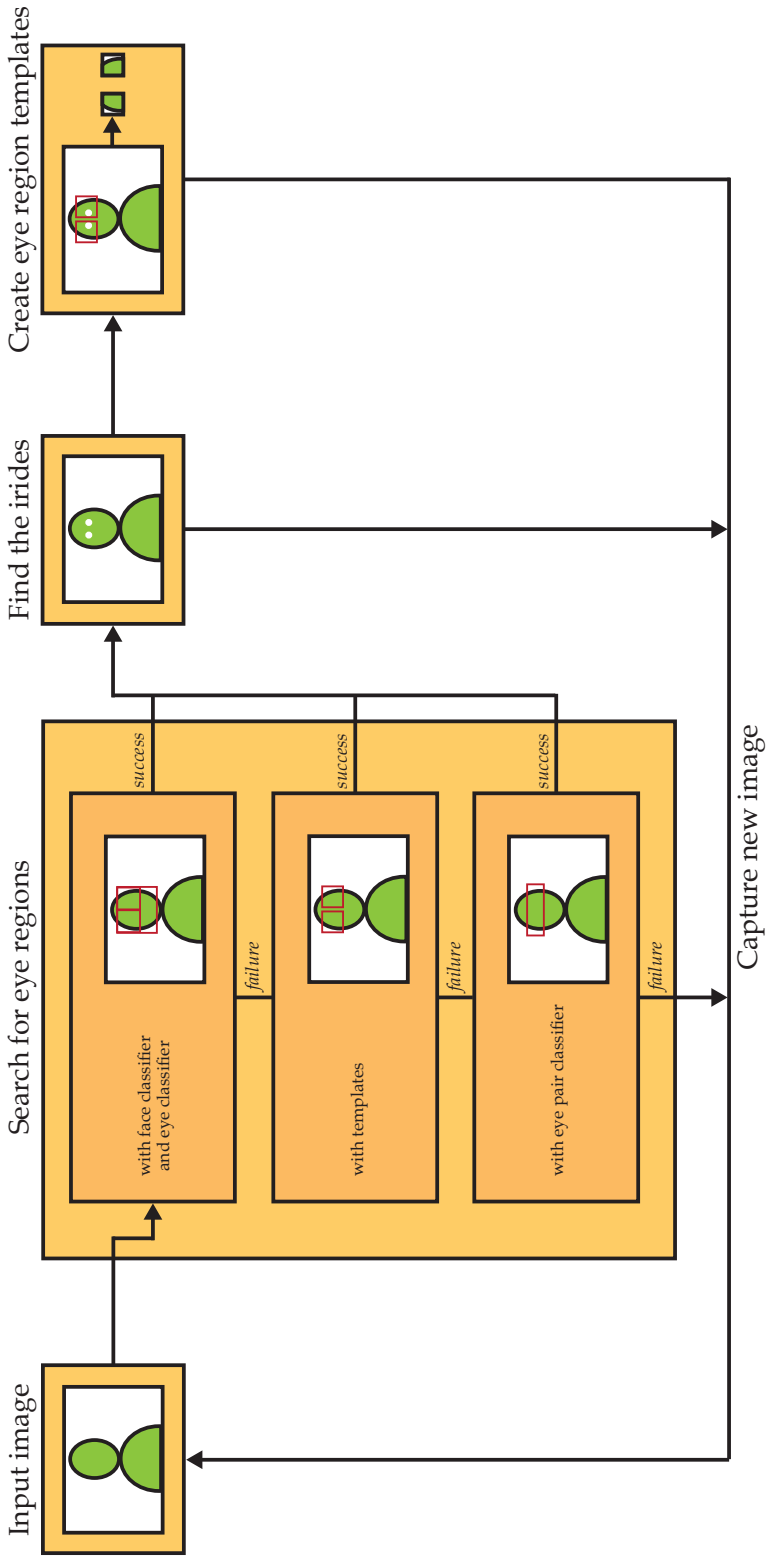

Figure 4.6: Eye tracking process : After searching the eye regions with three different approaches the algorithm starts to search for the irides and creates templates of the surrounding regions

Template matching improves eye positions

As last step the template shown in Figure 4.7 is searched inside the image with template matching. If the confidence value of the result is above a certain level, we presume that we found an iris and the corresponding point is marked with a small circle in the associated camera window.

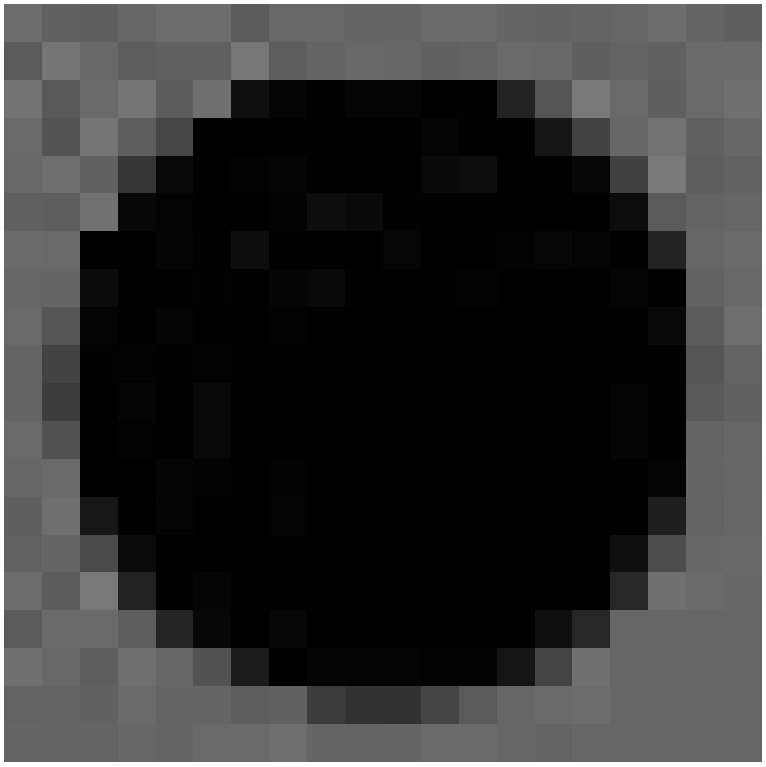

Figure 4.7: The template image we used to search for the iris. In most cases this template worked well and gave us the correct location

If both irides were found, the coordinates are passed on to the `calcEyePosition` function to estimate the real world coordinates of the user's head. The whole process of finding the eyes is depicted in Figure 4.8.

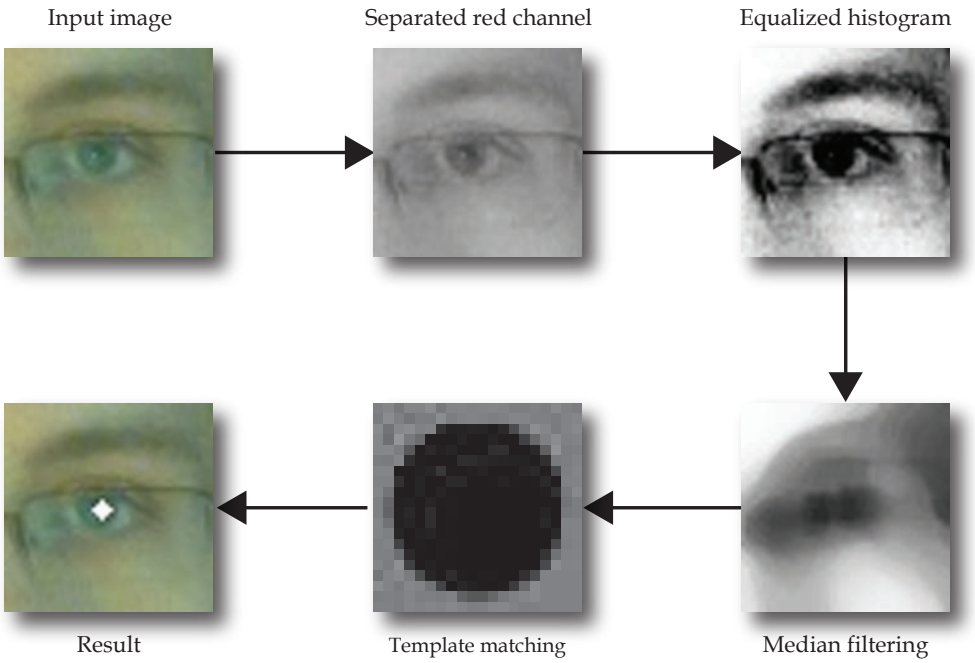

Figure 4.8: After equalizing the histogram of the red channel of our input image, we run a median filter on it and use template matching with the image of Figure 4.7 as template to find the iris coordinates

4.2.2 Marker Tracking

Multi-marker algorithm more stable

Marker tracking in our system is done with ARToolkit (see Chapter 2.4—"Marker Tracking" for an explanation on how it works). But tracking just one marker was not stable enough and the multi-marker approach of ARToolkit was too inflexible, so we added another multi-marker algorithm.

Initialization pass collects all marker positions

With this algorithm the user just has to specify which markers are used, but not their exact position relative to the origin. This can be done in the initialization file `marker.ini`. We define the origin to lie at the center of the first marker (marker 0) which is specified.

During the execution of the program we then memorize for every marker if it is visible in this pass and if it was found before. If a marker is recognized which has not been found before, its position can be calculated with the matrices that ARToolkit provides us.

There are two cases which have to be dealt with if a marker i that is recognized for the first time gets visible :

1. Marker 0 is visible
2. Marker 0 is not visible, but marker j, which was found before and whose position is therefore known, is visible

RELATIVE TRANSFORMATION:
If A is the transformation matrix for marker a and B is the transformation matrix for marker b, then we define the relative transformation from marker a to marker b as matrix B', so that
$A \cdot B' = B$.

Definition: *Relative Transformation*

In the first case we get the relative transformation for marker i as

$$\begin{aligned} & A \cdot B' = B \\ \Leftrightarrow\ & B' = A^{-1} \cdot B \end{aligned}$$

whereas A and B are the transformation matrices for marker 0 and i that are returned by ARToolkit and B' is the relative transformation matrix from marker 0 to marker i.

The second case also is not very difficult. Here we first calculate the relative transformation from marker j to marker i as

Relative transformations propagate to all newly found markers

$$B' = A^{-1} \cdot B$$

Then we get the final transformation B'' by multiplying B' by the relative transformation A' from marker 0 to marker j

$$B'' = A' \cdot B' = A' \cdot A^{-1} \cdot B$$

Positions saved for next execution

With these equations we can then calculate the relative position of every marker to the origin automatically. If the user quits the application, these positions are saved into `marker.ini.sav` and used for the next start.

Different matrices are combined for improved stability

Since in some cases we got more than one transformation matrix with corresponding errors estimates, we can use this additional information to stabilize the tracking result. For this we resorted to quaternions (see B—"Quaternions" for an explanation of quaternions) instead of matrices, because they are easier to interpolate.

Quaternions are used to combine the different transformations

The algorithm starts by calculating the transformation matrix B in the basis of marker 0 for all markers whose relative transformations are known and which are visible at the moment

$$B = A \cdot A'^{-1}.$$

Here A represents the transformation matrix of the corresponding marker i and A', as before, represents the relative transformation from marker 0 to marker i. Then this matrix is converted into a quaternion q and translational components $p[0]$, $p[1]$ and $p[2]$.

In the next step all of these quaternions and translation values are summed up relative to their associated confidence values, which can be obtained from the `ARMarkerInfo` structure, and at last they are divided by the sum of all confidences to get an unit quaternion. This quaternion with translation values is then converted back to a 4×4 matrix and forwarded to the rendering widget for perspective calculations.

4.2.3 Calculating the User-centered Perspective

To calculate the user-centered perspective we first have to estimate the position of the user's head relative to the mobile device. For this we will resort to the pinhole camera model (for an explanation see Appendix A—"The Pinhole Camera Model").

Pinhole camera model approximates mapping from 3D to 2D

Finding the Real-world Coordinates

In order to determine the real world position of the head relative to the mobile device, we need the focal length of the used camera in millimeters. Unfortunately, the camera calibration algorithm of ARToolkit only gives us f_x and f_y, the horizontal and vertical focal lengths in pixel units.

Focal length needed for computation

$$f_x = \frac{focal\ length[mm]}{pixel\_width[\frac{mm}{Pixel}]}$$
$$f_y = \frac{focal\ length[mm]}{pixel\_height[\frac{mm}{Pixel}]}$$

Therefore we need to know the pixel size of the camera, to calculate the focal length in millimeters and convert the image plane coordinates to millimeter units. For our device we used iSight cameras, which have a pixel size of $2.2\mu m \times 2.2\mu m$.

In Figure 4.9 the scene is shown from the top view. From the Pythagorean theorem we can conclude that

Interpupillary distance assumed to be 63 millimeters

$$\begin{aligned} & \overline{AP}^2 = (\frac{\overline{AB}}{2})^2 + z^2 \\ \Leftrightarrow \quad & z^2 = \overline{AP}^2 - (\frac{\overline{AB}}{2})^2 \\ \Rightarrow \quad & z = \sqrt{\overline{AP}^2 - (\frac{\overline{AB}}{2})^2} \end{aligned}$$

, where $\overline{AB}$ specifies the distance from point A to point B.

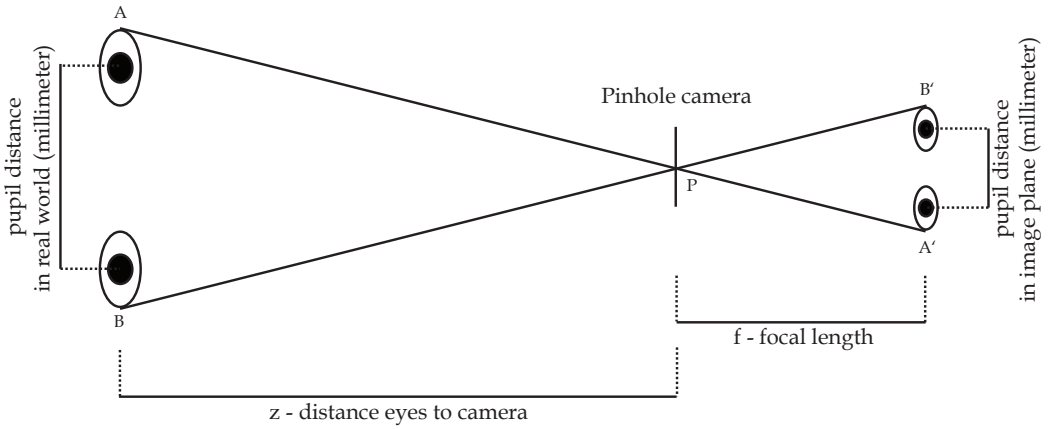

Figure 4.9: The pinhole camera model we used to estimate the distance of the viewer to the internal camera

This distance is also known as interpupillary distance and we estimate it to be 63 millimeters, which is the mean value for adults [Dodgson, 2004].

$\overline{AP} = \overline{PA}$ can then be calculated with the theorem on intersecting lines

$$\begin{aligned} &\overline{A'B'} : \overline{AB} = \overline{PA'} : \overline{PA} \\ \Leftrightarrow \quad &\overline{PA} = (\overline{PA'} \cdot \overline{AB}) : \overline{A'B'} \end{aligned}$$

Here we can again use the Pythagorean theorem to determine $\overline{PA'}$

$$\begin{aligned} \overline{PA'}^2 &= f^2 + (\frac{\overline{A'B'}}{2})^2 \\ \overline{PA'} &= \sqrt{f^2 + (\frac{\overline{A'B'}}{2})^2} \end{aligned}$$

so that $\overline{PA}$ can be written as

$$\overline{PA} = \frac{\sqrt{f^2 + (\frac{\overline{A'B'}}{2})^2} \cdot \overline{AB}}{\overline{A'B'}}$$

Inserting this partial result then yields the final solution

$$
\begin{aligned}
z &= \sqrt{(\frac{\sqrt{f^2 + (\frac{\overline{A'B'}}{2})^2} \cdot \overline{AB}}{\overline{A'B'}})^2 - (\frac{\overline{AB}}{2})^2} \\
&= \sqrt{\frac{(f^2 + \frac{\overline{A'B'}^2}{4}) \cdot \overline{AB}^2}{\overline{A'B'}^2} - \frac{\overline{AB}^2}{4}} \\
&= \overline{AB} \cdot \sqrt{\frac{f^2 + \frac{\overline{A'B'}^2}{4}}{\overline{A'B'}^2} - \frac{1}{4}} \\
&= \overline{AB} \cdot \sqrt{\frac{f^2}{\overline{A'B'}^2}} \\
&= \frac{\overline{AB} \cdot f}{\overline{A'B'}}
\end{aligned}
$$

Translation computed based on the midpoint of both eyes

After we have determined the distance of the viewer to the mobile device, we can calculate the horizontal and vertical translation based on this distance. For this we compute the x and y coordinates of the point (mid_x, mid_y), which lies in the middle between both eyes. This point is represented in Figure 4.10 as black dot.

The computation of the real world coordinates then works in a similar way to the computation of the distance

$$
\begin{aligned}
x^2 &= \overline{AP}^2 - z^2 \\
y^2 &= \overline{AP}^2 - z^2
\end{aligned}
$$

and again we can use the theorem on intersecting lines

$$
\begin{aligned}
2 \cdot \overline{A'B'} : 2 \cdot \overline{AB} &= \overline{PA'} : \overline{PA} \\
\Leftrightarrow \quad \overline{PA} &= (\overline{PA'} \cdot \overline{AB}) : \overline{A'B'}
\end{aligned}
$$

To eliminate the unknown, we then express $\overline{PA'}$ as

$$
\begin{aligned}
\overline{PA'} &= \sqrt{f^2 + \overline{A'B'}^2} \\
\Rightarrow \overline{PA} &= \frac{\sqrt{f^2 + \overline{A'B'}^2} \cdot \overline{AB}}{\overline{A'B'}}
\end{aligned}
$$

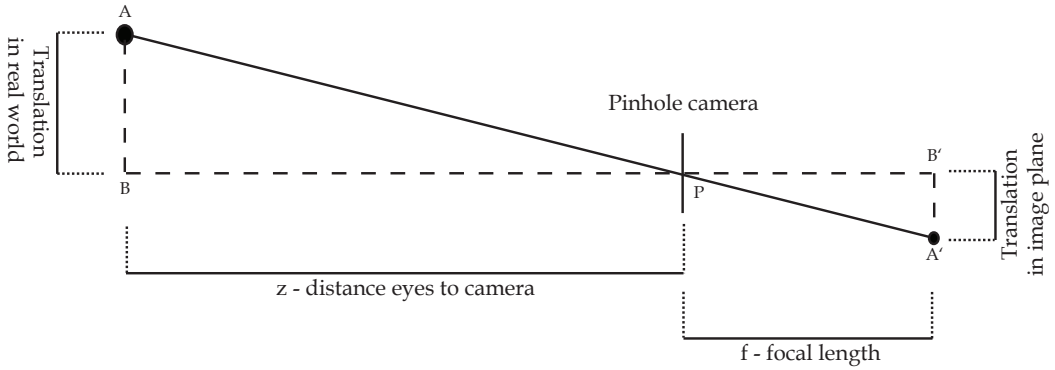

Figure 4.10: The pinhole camera model we used to determine the x and y translation based on the previously calculated distance

Inserting this leads to

$$\begin{aligned} x^2 &= (\frac{\sqrt{f^2 + {mid_x}^2} \cdot x}{mid_x})^2 - z^2 \\ &= \frac{(f^2 + {mid_x}^2) \cdot x^2}{{mid_x}^2} - z^2 \\ &= \frac{f^2 \cdot x^2}{{mid_x}^2} + \frac{{mid_x}^2 \cdot x^2}{{mid_x}^2} - z^2 \\ &= \frac{f^2 \cdot x^2}{{mid_x}^2} + x^2 - z^2 \\ \Rightarrow \quad x^2 &= \frac{z^2 \cdot {mid_x}^2}{f^2} \end{aligned}$$

Since the calculation of y works analogously, we can conclude

$$\begin{aligned} x &= \frac{z \cdot mid_x}{f} \\ \Rightarrow y &= \frac{z \cdot mid_y}{f} \end{aligned}$$

Creating the Matrices

Combination of tracking results yields user-centered perspective

In the last step the information we gained from marker tracking and eye tracking have to be combined into a model-view matrix and a projection matrix to setup the OpenGL rendering window. With the model-view matrix, every point is first transferred into the coordinate system of the device by multiplying it with `m_Matrix` and then it is transferred into the coordinate system of the viewer. This is done with the procedure `gluLookAt`, which creates a viewing matrix based on the

1. eye position,
2. the point where the camera is targeted at and
3. an up-vector, which defines the orientation of the camera.

The real world coordinates of the viewer in this case are represented by `trans_x`, `trans_y` and `trans_z`. Since in OpenGL the matrices are multiplied to the left side of the vector and concatenated transformations are applied from right to left, we need to reverse the order of the operations to yield the correct result.

```
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();

gluLookAt(  trans_x, trans_y, trans_z,
            trans_x, trans_y, 0.0f,
            0.0f, 1.0f, 0.0f );

glMultMatrixd(m_Matrix);
```

The viewing frustum specifies the field of view of the virtual camera

Then we set up the viewing frustum according to the position of the viewer and the width of the screen (`screenWidthMM`). For this we call `glFrustum`, which creates a perspective projection matrix with

1. the left and right vertical clipping planes,
2. the bottom and top horizontal clipping planes and
3. the near and far depth clipping planes.

The clipping planes for the four sides are furthermore multiplied by `nearPlane / trans_z` to keep the near plane at a fixed distance.

```
GLfloat nearPlane = 0.1f;
GLfloat hwidth = screenWidthMM / 2.0f;
GLfloat hheight = hwidth / aspect_ratio;

glMatrixMode(GL_PROJECTION);
glLoadIdentity ();

glFrustum(  nearPlane*(trans_x - hwidth)/trans_z,
            nearPlane*(trans_x + hwidth)/trans_z,
            nearPlane*(trans_y - hheight)/trans_z,
            nearPlane*(trans_y + hheight)/trans_z,
            nearPlane, 10000 );
```

After these matrices are set the OpenGL window is ready and we can start drawing the appropriate content based on the operation mode.

4.2.4 OpenGL Rendering

Associated mtl file defines the materials used in the model

We wanted to display three-dimensional models, so we included the GLM library by Robins to load and render Wavefront obj files. The GLM library was provided in the examples of the GLUT (OpenGL Utility Toolkit) source distribution. We choose the obj format, because the files have a simple structure and most modeling applications provide an export option for it.

To use textured models we extended the library with the

```
GLuint glmInitTexture(char* texName)
```

function, which loads the texture specified by `texName` with OpenCV's image functionality and returns a `GLuint`, which can be used by OpenGL. We also embedded the draw commands into Display lists to speed up the rendering process.

Shadows provide information about relation between objects

As shadows are a strong cue for the relative disposition of objects in images, we included a three pass shadow mapping approach into our rendering process (see Appendix C—"Shadow Mapping" for an explanation of shadow mapping). For this we used frame buffer objects to store the depth map, because in this way the size of the depth map was not limited by the resolution of the display.

GLee also used to access shaders

Since we needed some OpenGL extensions, we additionally included the GLee (GL Easy Extension) library by Woodhouse. It is a cross-platform library, which aids in accessing almost all OpenGL extensions. This way we did not have to struggle with the varying implementation details of the different platforms.

In the following we will describe for each operation mode of our software which settings and what models were used.

Measurement Mode

For the "Measurement mode" we included a model of a simple box with pits and bulges, where we hid letters which the users had to find in the user test (see Figure 4.11). As background for the box we choose a wooden texture that should resemble the table on which we placed the markers. To model the box we used Google SketchUp[1] and exported the result to obj file format.

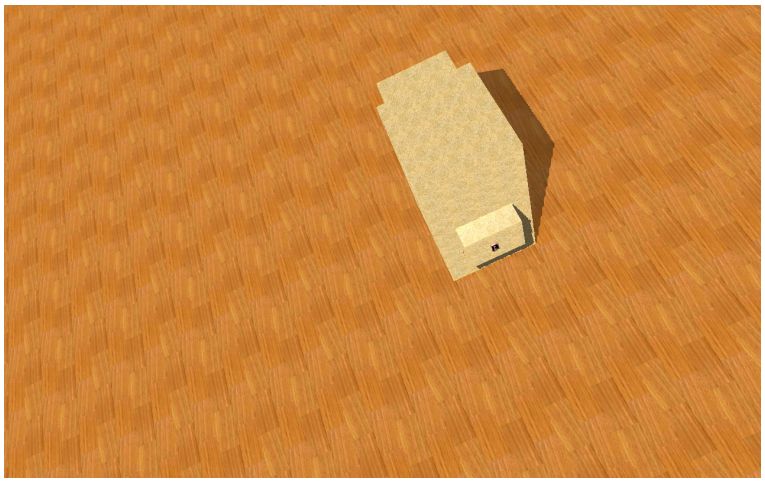

Figure 4.11: Screenshot of the "Measurement mode", where a simple model of a box is displayed

Augmented Reality Mode

Since we wanted to compare the normal Augmented Reality approach to our approach with user-centered perspective, we implemented an "Augmented Reality mode". In this mode the same model as in the "Measurement mode" is displayed, but with exchanged letters. The model is here rendered on top of the current picture of the backside camera. In Figure 4.12 this operation mode including one annotation, which is placed inside the world, can be seen.

[1] Free modeling software by Google `http://sketchup.google.com/`

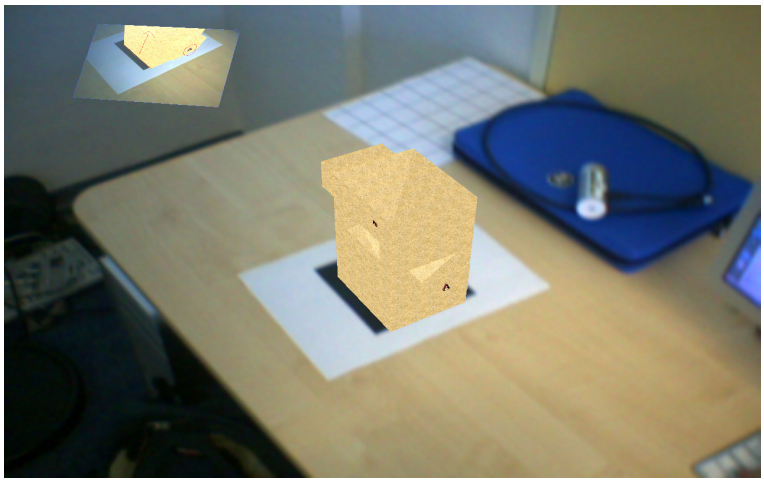

Figure 4.12: Screenshot of the "Augmented Reality mode", where the same model is displayed as in the "Measurement mode" but the background is exchanged with the current picture of the camera on the backside. An annotation can be seen, which was placed into the scene before

Annotation Mode

To test the annotation feature of our device, we implemented an own "Annotation mode". In this mode a defective model of a car is displayed, with the intention, that the user marks every flaw that he encounters. As can be seen in Figure 4.13 we choose to include only blatant errors, so that it is obvious for the user which points he has to mark. To deform the car we again used Google SketchUp. For this mode we additionally included simple reflections of the car on the ground by rendering the model again, but this time mirrored along the y-axis.

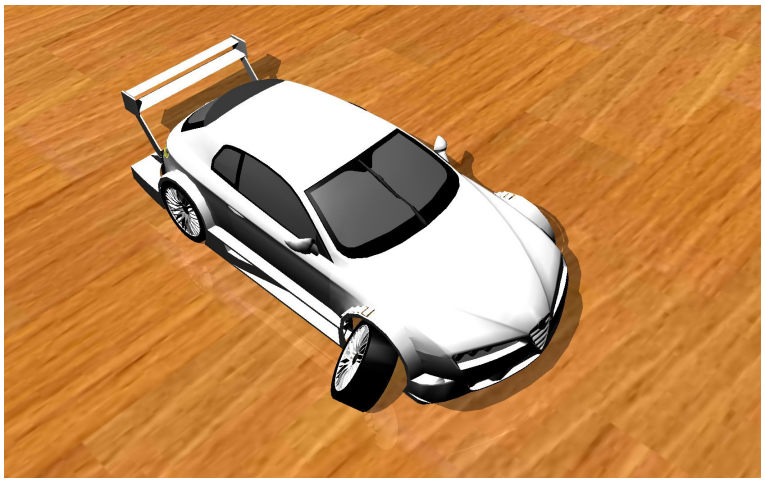

Figure 4.13: Screenshot of the "Annotation mode", where one can see clearly the wrong orientation of the tire and the right mirror

Volume Mode

The "Volume mode" shows one of the possible applications of our device. A computed tomography image (`CTheadsmall.raw`) is displayed in front of a grid and the user can navigate inside this volume, either by moving the device or by moving the head. For this we implemented the class `VolumeRenderer`, which loads a raw image file, interprets the content as three-dimensional texture with predefined measurements and displays it with the use of a vertex shader.

In the program options, which can be reached in the viewing mode with additional information, the user can vary the number of slices which are displayed simultaneously as can be seen in Figure 4.14.

For this mode we disabled the shadow mapping algorithm, since the displaying of the three-dimensional texture would otherwise have been too slow.

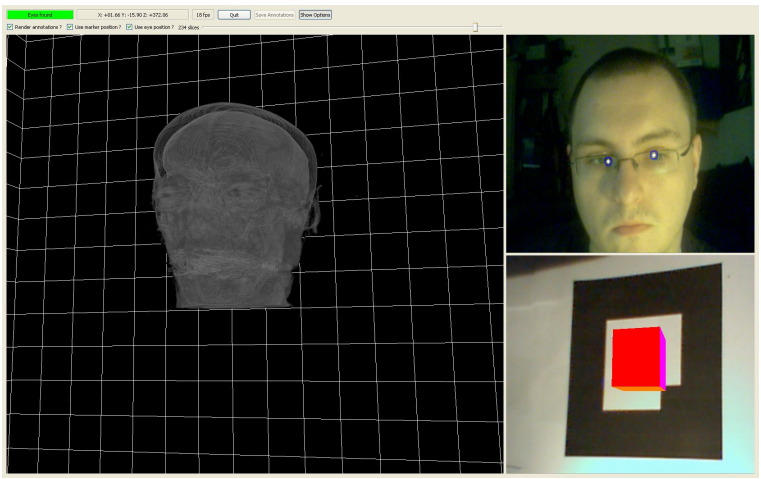

Figure 4.14: In the ”Options” section the user can control the number of slices that are displayed and turn on or off both tracking algorithms

4.2.5 Annotations

Annotations can be used to cooperate on a project

As said before, we also implemented an annotation feature to give users the ability to place comments inside the virtual environment. A new annotation can be created by clicking the left mouse button inside the rendering window and finished by clicking the right mouse button. The user can now draw freely and for example mark bad parts of a model as can be seen in Figure 4.15.

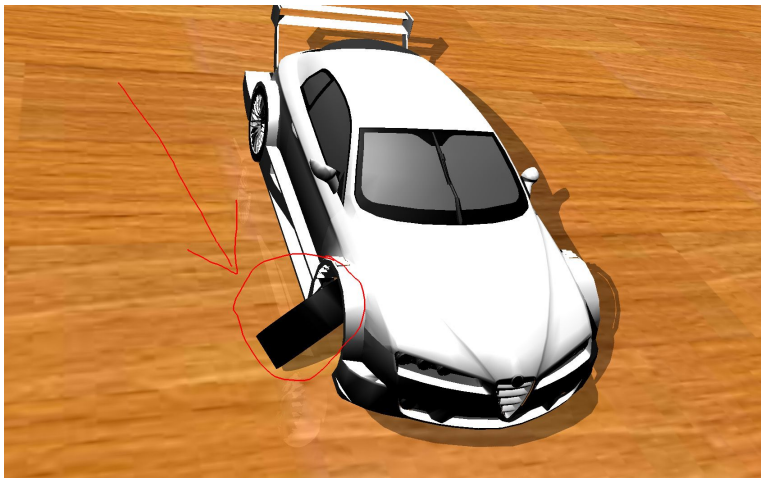

Figure 4.15: To start an annotation the user has to click the left mouse button, or with a touchscreen, just begin to draw onto the desired locations

Perspective is frozen during creation

While the annotation mode is active any changes to the user's perspective or the position of the device will be discarded, so the user does not have to remain frozen for the time of creation. After the creation is finished, the annotation is placed inside the virtual world onto the position where it was started (see Figure 4.16) and a bitmap file with the current date and time is created inside the `/pic` folder.

Annotations can be minimized

Since the creation of many annotations could obscure the object and confuse the user annotations can be minimized in our system. This can be achieved by right-clicking onto the designated comment. Figure 4.17 shows the same scene as Figure 4.16, only with a minimized annotation.

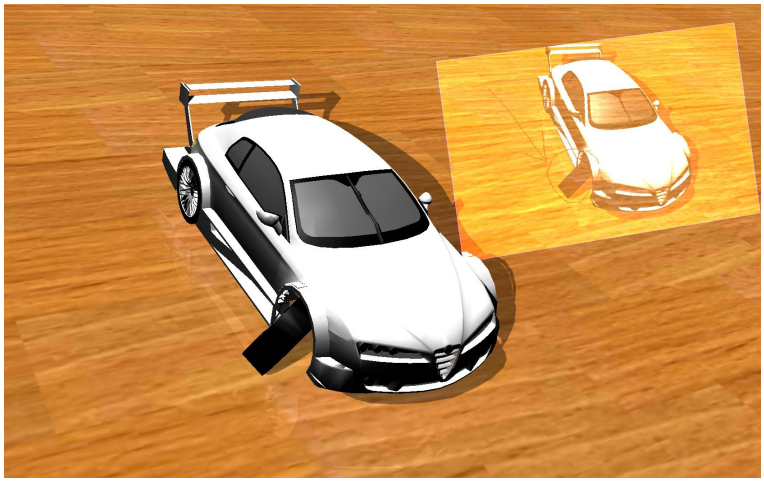

Figure 4.16: The previously created annotation is placed onto the correct location and is rotated accordingly

Figure 4.17: By clicking with the right mouse button onto an annotation the user can minimize it

Chapter 5

Evaluation

"No amount of experimentation can ever prove me right; a single experiment can prove me wrong."

—Albert Einstein

Users had to walk around a virtual box

To evaluate our work we conducted two user studies. The tests should show if the approach with user-centered perspective brought advantages in comparison with the normal Augmented Reality method or if it just confuses the users. For this we placed the markers on a table, which was situated in the middle of the room and calibrated our device to these marker positions by hovering the device over the markers until all their positions were known (also see Subsection 4.2.2—"Marker Tracking"). Figure 5.1 shows the setup we used in our tests.

In the first two of the following sections we will start with an analysis of the questionnaire we handed out after the tests and in the third section we will compare the times we measured. The contents of the last section will then be used to summarize the results.

Figure 5.1: We placed the markers on a table and the users had to walk around to find different letters

5.1 First User Test

Questionnaire should give qualitative feedback

Our first test was split up into two phases. For the first phase the device was operated consecutively in the normal Augmented Reality mode and then in our mode with eye tracking. For each test person we exchanged the order of the two approaches randomly and — as it was a single blinded trial — the users had to compare ″Method A″ to ″Method B″.

After a minute to adapt to the device, the users had to find letters that were affixed to a box inside the virtual world and say them aloud if they encounter one. Figure 5.2 shows the model of the box. They were then asked to fill out a questionnaire to provide qualitative feedback. For a quantitative comparison we measured the time the users needed to fulfill the task.

Flaws were placed obviously

With the second test phase we wanted to evaluate our annotation feature. To evaluate this feature a faulty model of a car was presented to the users and they had to mark all obvious flaws. We then asked them to give qualitative feedback about this feature, to find out if it was beneficial to place the comments as half-transparent billboards inside the virtual world (see Figure 5.3).

Our test group consisted of two females and six males. Each of them was a computer science student between the age of 24 and 30. Seven out of eight stated that they use the computer everyday, whereas only one of them stated that he uses 3D modeling programs more often than once a week.

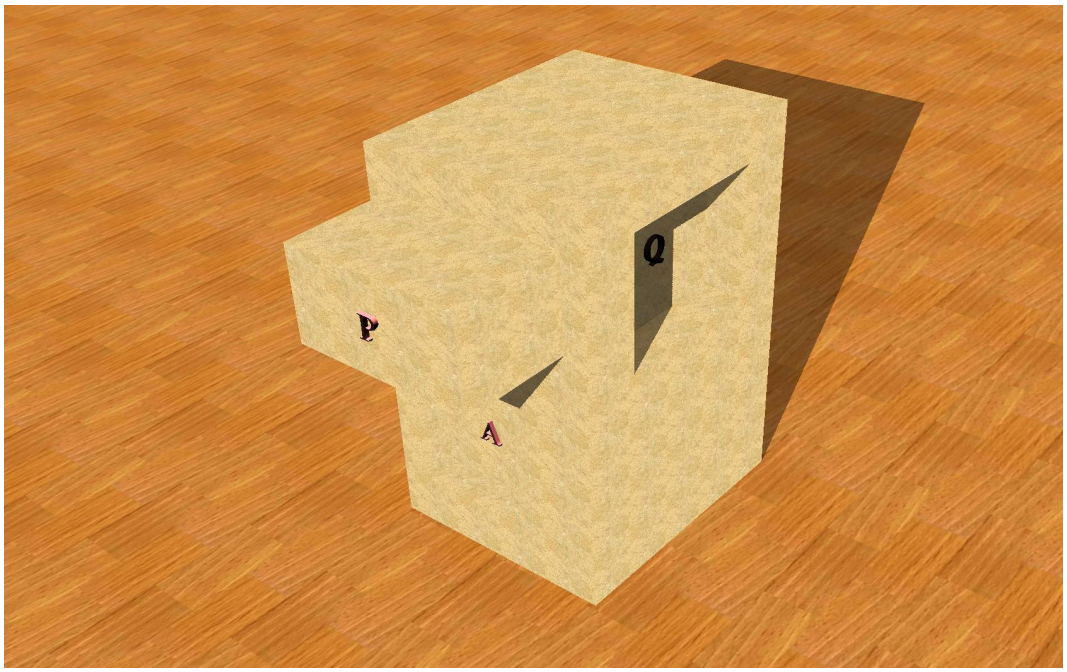

Figure 5.2: The model we used in the first user test. The letters were placed on each side of the box, so the user had to walk around to find all of them

Figure 5.3: Car with an annotation, which points out the mistake in the model

5.1.1 First Phase

The first test phase showed that the mean time the users needed to find the letters were higher with user-centered perspective. Only two participants were faster with our approach and interestingly one of them was the person who stated that he often uses modeling programs. All participants had no problems in finding all letters.

"Real" background more intuitive for users

The questionnaire we handed out after the tests yielded that, except one person, all of them favored the Augmented Reality approach. As the answers to the other questions pointed out, this can be lead back to the "real" background in the Augmented Reality approach and the, sometimes, faulty eye tracking results. Every time a user moved out of the field of view of the camera or moved too fast, the tracking got lost. On the one hand this was due to the opening angle of the camera, and on the other hand it was due to tracking errors forced by an remaining uncertainty in the underlying tracking algorithm.

Participants did not notice that their eyes were tracked

This also explains why some participants noted that the Augmented Reality approach felt "smoother", although the tracking results were the same in both cases and both approaches ran at the same rate of frames per second. As it was a single blind trial we could not tell them that eye tracking was used and, unfortunately, most persons used the device the same way they did in the Augmented Reality version.
An overview of the results of the questionnaire for the first test can be found in Table 5.1.

		Strongly disagree	Disagree	Neither agree nor disagree	Agree	Strongly Agree	N/A
I would use method	Augmented Reality		1		2	5	
	Virtual Reality with eye tracking		4	1	3		
Requires more attention	Augmented Reality	2	6				
	Virtual Reality with eye tracking				5	3	
Navigation always behaved like I expected	Augmented Reality			1	6	1	
	Virtual Reality with eye tracking		6	2			

Table 5.1: The complete results of the first test

5.1.2 Second phase

The questionnaire the users filled out after they found all faults of the car model showed that half of the users liked the idea of taking notes and placing them at the current position. They stated, that it was a good feature for group discussions and that additional features like zooming or highlighting of the marked area could be added.

	Strongly disagree	Disagree	Neither agree nor disagree	Agree	Strongly Agree	N/A
Displaying the annotations inside the virtual world helped me		2	2		4	
The annotation method always behaved like I expected		2	2		4	

Table 5.2: Results of the annotation test

Some users did not see annotations

After the questionnaire was finished, we asked the other half of the group what they did not like about this feature and it showed, that they had not seen any of the annotations they made previously. Some just noticed a small flash up when they passed one, but they thought that it was an error of the device.

5.2 Second User Test

Most users in the first test preferred the Augmented Reality mode

Since most users preferred the Augmented Reality approach because of the "real" background and we could not provide one for our approach, we modified the conditions for the second user test. Instead of using our mode and the Augmented Reality mode, we just used only the first one. The eye tracking was then randomly disabled for the first or second run. We additionally modified the model that was used to include a bigger ledge under which we also hid a letter as can be seen in Figure 5.4. In this way we wanted to motivate the users also to use their head movement to change the current view.

For the second user test the user group again consisted of eight persons between the age of 24 and 30. Seven of them were male computer science students and one of them was a female sales assistant. The tasks the users had to fulfill remained the same as in the first test and once again we measured the time the users needed.

As the measured times suggest, the second test was harder to cope with. The mean times the participants needed were more than twice as large as in the first test.

Changed conditions also changed outcome of the questionnaire

Due to the changed conditions now all of the users discovered that they also could use their head to yield a certain view on the scene. This resulted in a slightly changed outcome of the questionnaire. Now four of the eight participants preferred our approach, while the other four chose the approach with only marker tracking.

Instability of eye tracking confused many users

As we, after the questionnaire was finished, asked those four persons why they preferred the approach without eye tracking, they replied that they would have chosen our approach, if it would have been more stable. The instability was, as in the first test, partially a result of the small opening angle of the camera and partially of our eye tracking algorithm, which, in some cases, produced false positives.

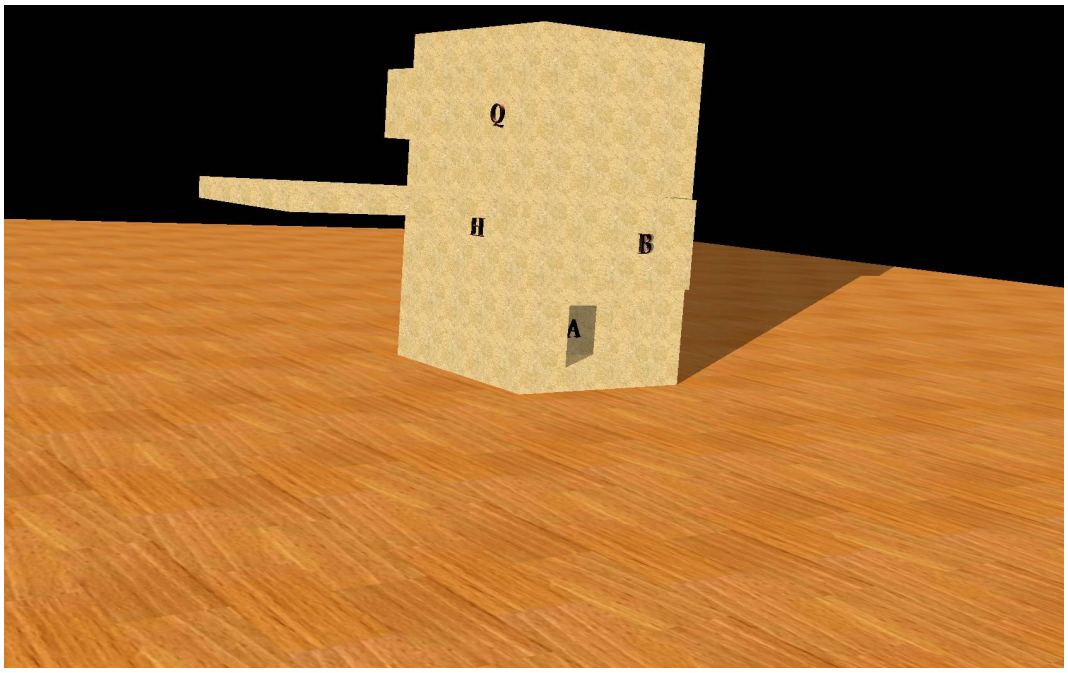

Figure 5.4: The big ledge on the left side was added to make the user aware of the eye tracking

In contrast to the first test, where most of the users only employed the device itself to alter the viewpoint, this time they also used the position of the head as can be seen in Figure 5.5. To find the letter under the big ledge, for example, some participants held the device fixed at a low position and then tried to adjust the view by moving the head up and down.

Table 5.3 summarizes all results of the questionnaire for the second test.

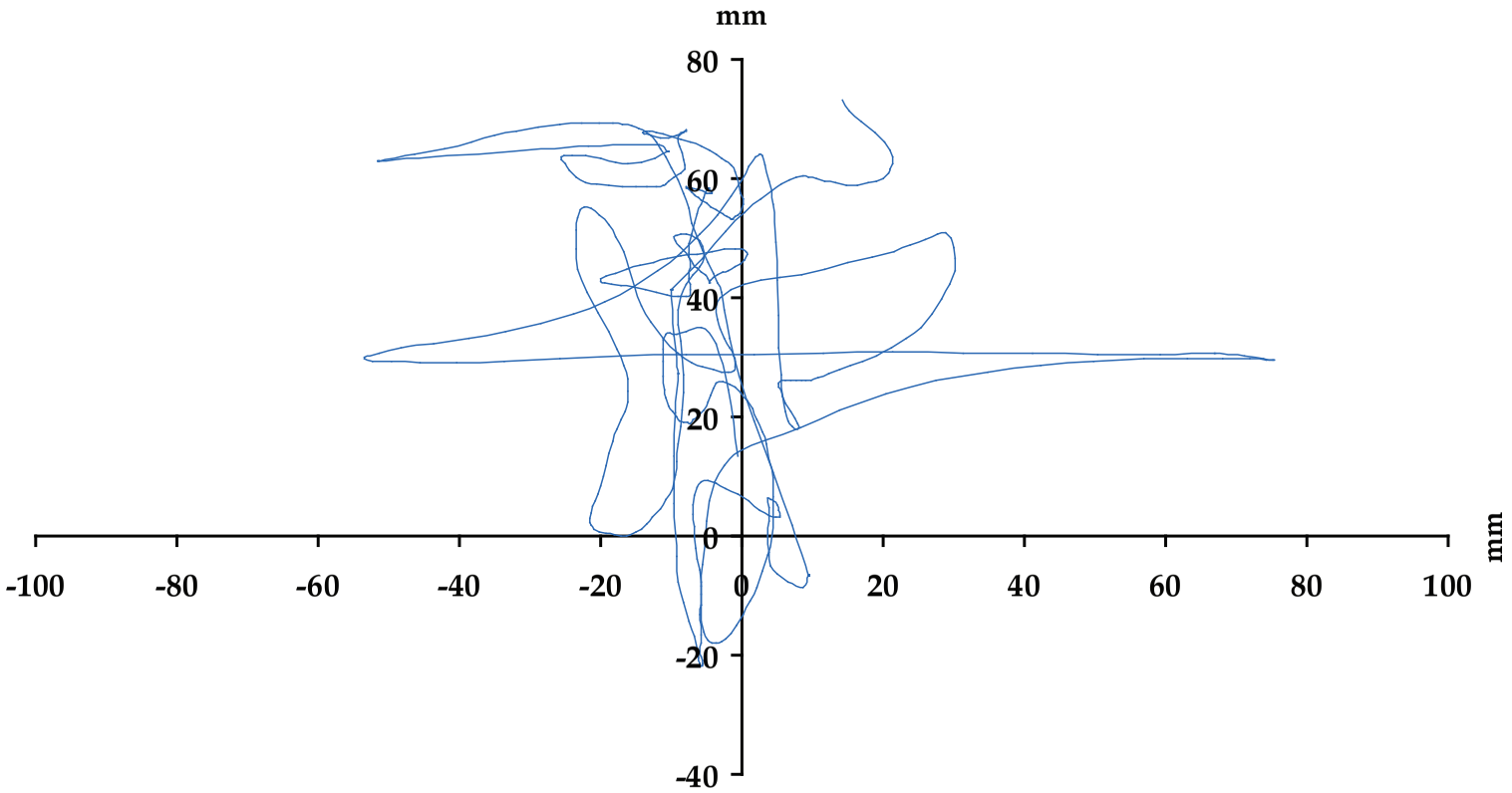

Figure 5.5: This diagram shows the movements of one user, who used the eye tracking to change the viewpoint of the device. Because of the way the user held the device the whole diagram is translated in y direction

			Strongly disagree	Disagree	Neither agree nor disagree	Agree	Strongly Agree	N/A
I would use method	Without tracking	eye	1	3	2	2		
	With tracking	eye		1	3	2	2	
Requires more attention	Without tracking	eye		1	3	3		1
	With tracking	eye		3	2	2		1
Navigation always behaved like I expected	Without tracking	eye		5	1	1	1	
	With tracking	eye	1	2		4	1	

Table 5.3: The complete results of the second test

5.3 Comparison of Time Measurements

For both user tests we measured the times the users needed to fulfill the tasks. The mean times and the corresponding standard deviations that emerged from this can be found in Table 5.4 and Table 5.5.

	Augmented Reality mode	Virtual Reality with eye tracking
Mean time	12.6 sec	14.1 sec
Standard deviation	3.3 sec	4.0 sec

Table 5.4: The results of the time measurement for the first test

	Without eye tracking	With eye tracking
Mean time	29.4 sec	33.9 sec
Standard deviation	7.9 sec	8.5 sec

Table 5.5: The results of the time measurement for the second test

Student's t-test used to test for statistical significance of results

Based on these results we now perform a paired Student's t-test. For this we will assume that the differences of the measured times for each participant are realizations of independent, normally distributed random variables with mean value μ and unknown variance σ.

Our null hypothesis is, that the users need the same time for both approaches, or formal : $\mu = 0$. As alternative, the measured data suggests, that the users are slower with our method, which means $\mu > 0$.

For our data the corresponding one-sided t-test returns

- $t = 0.9608$ for the first test and
- $t = 0.9967$ for the second one.

The critical value however, which the Student-distribution for $7 = (n-1)$ degrees of freedom and a level of significance of 0.05 returns, is 1.89446 (for a level of significance of 0.1 it returns 1.415).

This on the one hand means, that for the given level of significance it is not statistically safe to refuse the null hypothesis in favor of the alternative and on the other hand, that due to the asymmetry of the t-test the null hypothesis can also not be verified.

5.4 Discussion

The tests we did indicate that user-centered perspective could be a helpful addition to mobile 3D navigation devices. As soon as the participants found out that they could change the perspective with their head movement, they tried to utilize it to fulfill the given task. Unfortunately the eye tracking method was not robust enough to allow a completely natural movement in front of the camera. This confused some of our test persons, which made the system less intuitive to use for them.

As the first test pointed out, the combination of virtual objects and real background was the most accessible operation mode for the majority of our participants. Even if no marker was found this method still showed the view "through" the device and therefore made it easier to re-establish the tracking.

We also tested our annotation feature and the results were positive. Half of the users stated that the comments inside the virtual world helped them and that they behaved as expected. But to make the annotation feature more useful the weight of the device has to be reduced. This would allow the user to hold the system with one hand, while marking interesting points with the other one.

Chapter 6

Summary and Future Work

"Prediction is very difficult, especially about the future."

—Niels Bohr

The following chapter contains a summary of our research, points out the contributions to the field of mobile Augmented Reality and provides some ideas for future research to improve the device.

6.1 Summary and Contributions

At first we started with an introduction to the area of mobile Augmented Reality. Then we showed, what current approaches can be used for and what their limitations are.

Currently no mobile device with user-centered perspective

It became clear that all approaches are either

- not mobile,
- do not offer a user-centered perspective,

- are too expensive or
- require additional equipment

We then developed our own system that should overcome these limitations and provide the user with an intuitive solution for 3D navigation. As basis for our device we chose a tablet computer with touchscreen functionality and an internal camera. This was extended with another external camera for marker tracking.

Tracking determines position of device and user

In the next step we implemented our own eye tracking algorithm based on Haar classifiers combined with template matching and extended the marker tracking algorithm of ARToolkit. These tracking results were then used to calculated the appropriate user-centered perspective. Since we could not provide the camera picture for this perspective, we instead implemented a virtual scene which should imitate the real view.
To test the system we defined four different operation modes, which all display different scenes. Additionally an annotation feature was presented, which enables the users to place half-transparent comments inside the virtual world.

Device has to be improved to be more useful

As last step we evaluated our system in two user tests. The tests showed that our approach would be beneficial, if the tracking results would be improved and the weight of the device could be reduced. It also exemplified, which relevance the ”real” background of Augmented Reality applications has on the perception of the usability of the device.

As main contributions to the research community there are

- the extended eye tracking algorithm, which is fast, needs no user calibration and, in most cases, works with acceptable results

- the multi-marker approach, which permits the use of a set of markers without defining their exact positions in advance
- the test results, which showed that the users generally would prefer the system with user-centered perspective, if it worked flawlessly

6.2 Future Work

Prototype was too heavy for long time use

Since most users complained about the weight of our prototype, the first way to improve the system would be to find a lighter device. It should feature a more powerful graphics processor, to enable a more realistic presentation and improve the overall performance of the system. As said in the introduction, mobile devices become smaller and more powerful with every new generation, so the next version of tablet computers should be able to satisfy the requirements.

Get rid of the markers

Another limitation was that the device only recalculates its own position if one or more markers are in sight. For use in a natural environment it would be more convenient to set the markers aside and use other techniques. One possibility would be, to use a set of predefined reference images and calculate the position and orientation based on a selection of significant points. This would enable the usage of the device in environments where no markers could be placed or are not allowed.

Improve eye tracking

In the current prototype the range in which our eye tracking algorithm works sufficiently well is limited. Since the distance of the user to the device during the process is rather small, the built-in camera can only capture a fraction of the space the user could move to. Furthermore our eye tracking algorithm sometimes returns false positives and starts tracking the wrong points. These limitations could be attenuated by using a wide-angle lens camera or multiple cameras and improving the eye tracking algorithm.

Introducing, for example a calibration phase for every user and using this prior knowledge of the face would speed up the tracking and improve the estimation of the real world coordinates.

Use the device itself to draw notes

One participant of our user test suggested a different method to draw annotations. Since our prototype required the user to hold it with both hands because of the weight, he suggested to use the movements of the device itself to control the drawing. In this way the users could place simple marks like circles without even taking the hands off the device.

Multiple Users at the same time

For certain applications it would also be useful, if multiple users could work concurrently. Then one user would see the annotations by everyone else and they could collaborate inside the virtual world. This could be realized by defining one device as host, which gets the information about the position and orientation of all other devices and shares this information. The different users could then be represented by avatars in the virtual room.

Exchange virtual scene against camera picture

As pointed out in Chapter 4—"System Design and Implementation" a big drawback of our approach is that a virtual scene instead of the camera picture of the real world is drawn. A sphere camera or a motorized camera could for example be used to calculate the appropriate view. Another possibility would be, to use an array of cameras for the backside and a lumigraph (see Gortler et al. [1996]) in order to determine the correct background picture.

Appendix A

The Pinhole Camera Model

The pinhole camera model is widely used in the area of computer vision. It ignores many effects like geometric distortions but thereby it is very simple.

A point $P = (X, Y, Z)^T$ is projected onto the image plane, by calculating the intersection $Q = (x, y)^T$ of the projection line from P through the origin with the image plane (see Figure A.1). It is easy to see, that there are two similar triangles which share the projection line as their hypotenuses. Therefore the following equation holds

$$\begin{aligned} & \frac{y}{f} = \frac{Y}{Z} \\ \Leftrightarrow \quad & y = \frac{f \cdot Y}{Z} \end{aligned}$$

A similar result can be found for the x coordinate

$$\begin{aligned} & \frac{x}{f} = \frac{X}{Z} \\ \Leftrightarrow \quad & x = \frac{f \cdot X}{Z} \end{aligned}$$

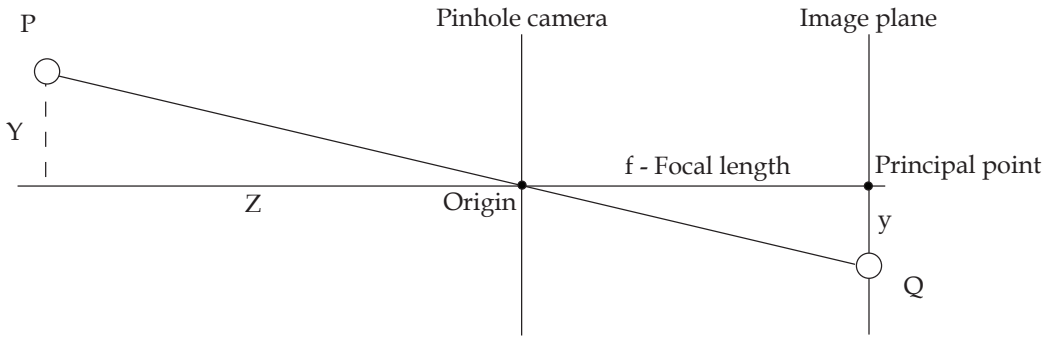

Figure A.1: Geometry of the pinhole camera model as seen from the side

But these equations only hold if the principal point, the intersection of the optical axis and the image plane, is in the middle of the image plane. Otherwise we have to add a translation (c_x, c_y). This yields as final result

$$\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} \frac{f \cdot X}{Z} + c_x \\ \frac{f \cdot Y}{Z} + c_y \end{pmatrix} = \frac{f}{Z} \cdot \begin{pmatrix} X \\ Y \end{pmatrix} + \begin{pmatrix} c_x \\ c_y \end{pmatrix}$$

Appendix B

Quaternions

Quaternions are an extension to complex numbers and are named $\mathbb{H}$, after their inventor Sir William Rowan Hamilton [1844]. Since they are more efficient and more numerically stable than rotation matrices, quaternions are often used in many fields like computer graphics and navigation.

A quaternion always can be expressed by

$$q_0 + q_1 \cdot i + q_2 \cdot j + q_3 \cdot k \quad q_0, q_1, q_2, q_3 \in \mathbb{R}$$

where i,j and k satisfy

$$i^2 = j^2 = k^2 = i \cdot j \cdot k = -1.$$

The imaginary part here behaves like a vector $(q_1, q_2, q_3) \in \mathbb{R}^3$, whereas the real part q_0 acts as a scalar $a \in \mathbb{R}$.

In the following we will give definitions of the essential operations for working with quaternions :

- `Sum of two quaternions` $Q, R \in \mathbb{H}$

  $Q + R := q_0 + r_0 + (q_1 + r_1) \cdot i + (q_2 + r_2) \cdot j + (q_3 + r_3) \cdot k$

- Conjugation $\overline{Q}$ of $Q \in \mathbb{H}$

$$\overline{Q} := q_0 - q_1 \cdot i - q_2 \cdot j - q_3 \cdot k$$

- Product of two quaternions $Q, R \in \mathbb{H}$ (not commutative)

$$\begin{aligned} Q * R := & \, q_0 \cdot r_0 - q_1 \cdot r_1 - q_2 \cdot r_2 - q_3 \cdot r_3 \\ & + (q_1 \cdot r_0 + q_0 \cdot r_1 - q_3 \cdot r_2 + q_2 \cdot r_3) \cdot i \\ & + (q_2 \cdot r_0 + q_3 \cdot r_1 - q_0 \cdot r_2 + q_1 \cdot r_3) \cdot j \\ & + (q_3 \cdot r_0 + q_2 \cdot r_1 - q_1 \cdot r_2 + q_0 \cdot r_3) \cdot k \end{aligned}$$

- Dot-product of two quaternions $Q, R \in \mathbb{H}$

$$Q \cdot R := q_0 \cdot r_0 + q_1 \cdot r_1 + q_2 \cdot r_2 + q_3 \cdot r_3$$

- Norm of $Q \in \mathbb{H}$

$$\|Q\| := \sqrt{Q \cdot \overline{Q}} = \sqrt{{q_0}^2 + {q_1}^2 + {q_2}^2 + {q_3}^2}$$

A unit quaternion, which we will use to represent rotations, has a norm of one. One desirable characteristic of it is, that its inverse equals its conjugate. Any quaternion $Q \neq 0$ can be transformed to a unit quaternion Q' by dividing it by it's norm

$$Q' = \frac{Q}{\|Q\|} = \frac{q_0}{\|Q\|} + \frac{q_1}{\|Q\|} \cdot i + \frac{q_2}{\|Q\|} \cdot j + \frac{q_3}{\|Q\|} \cdot k$$

As said before, quaternions can be used to easily represent rotations in $\mathbb{R}^3$. For this we first embed a point $x = (x_0, x_1, x_2)^T$ into $\mathbb{H}$

$$X = 0 + x_0 \cdot i + x_1 \cdot j + x_2 \cdot k \quad \in \mathbb{H}$$

Multiplying this with a unit quaternion

$$Q = cos(\tfrac{\theta}{2}) + a_x sin(\tfrac{\theta}{2}) \cdot i + a_y sin(\tfrac{\theta}{2}) \cdot j + a_z sin(\tfrac{\theta}{2}) \cdot k$$

with ${a_x}^2 + {a_y}^2 + {a_z}^2 = 1$ according to

$$X' = Q * X * Q^{-1} = Q * X * \overline{Q}$$

rotates the point x along the axis (a_x, a_y, a_z) by an angle θ.

Using quaternions instead of Euler angles has several benefits :

- `Gimbal lock is avoided`
  Because the final rotation matrix with Euler angles depends on the order of multiplications, rotations over one axis sometimes get mapped to another axis or it is even impossible to rotate around a specific axis. This phenomenon is commonly called Gimbal lock.
- `Easier to read-off rotation axis and angle`
  As you have seen, it is very easy to construct a quaternion for a rotation along a specific axis and angle. Likewise it is also not very hard to read-off the angle and axis by transforming this equation a little bit.
- `Easy to create a smooth rotation`
  While it is hard to calculate rotation matrices to rotate smoothly from one point to another for quaternions there exist methods like Slerp (spherical linear interpolation), which can be used to compute the intermediate quaternions.

At last we will show how to convert a quaternion $Q = q_0 + q_1 \cdot i + q_2 \cdot j + q_3 \cdot k \in \mathbb{H}$ back to a rotation matrix $M \in \mathbb{R}^{3\times 3}$

$$M = \begin{pmatrix} {q_0}^2 + {q_1}^2 - {q_2}^2 - {q_3}^2 & 2q_1q_2 - 2q_0q_3 & 2q_1q_3 + 2q_0q_2 \\ 2q_1q_2 + 2q_0q_3 & {q_0}^2 - {q_1}^2 + {q_2}^2 - {q_3}^2 & 2q_2q_3 - 2q_0q_1 \\ 2q_1q_3 - 2q_0q_2 & 2q_2q_3 + 2q_0q_1 & {q_0}^2 - {q_1}^2 - {q_2}^2 + {q_3}^2 \end{pmatrix}$$

This can then be used again in, e.g., OpenGL.

Appendix C

Shadow Mapping

The shadow mapping approach we used in this thesis works in a three step process and is based on the original approach by Williams [1978]. In the first step the scene is rendered from the light's point of view. This step is only used to fill the depth map with the corresponding depth values for each pixel and therefore all additional things like lighting are disabled and only flat shaded backsides are drawn (Figure C.1 shows the content of the depth buffer after the first pass).

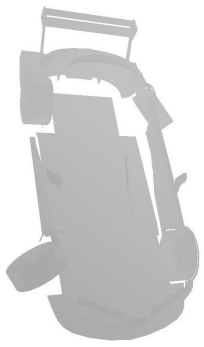

Figure C.1: Depth buffer content after the first pass

For the second step the scene is then rendered from the camera view with dim lighting. This step, which is depicted in Figure C.2, will later create the shadowed areas.

Figure C.2: Car model rendered with dim lighting for the shadowed areas

In the last step we draw the lit areas of the scene. To accomplish this, we have to convert each point we want to draw into light space coordinates, to compare it to the corresponding depth map value. For this we use the `ARB_shadow` extension of OpenGL, which takes care of the comparison. If the depth value of the current pixel now is smaller or equal to the depth map value, we draw it and otherwise it is discarded. The combination of lit areas and shadows then yields the final image which can be seen in Figure C.3.

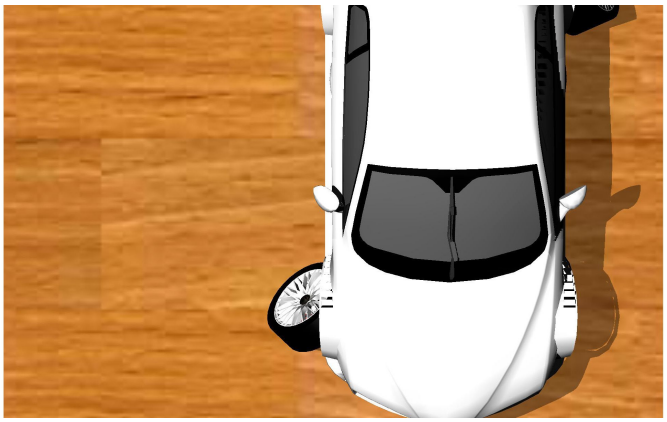

Figure C.3: The car model after combining all passes

Appendix D

CD Contents

The CD which comes enclosed with this thesis contains two folders "`VirlWind`" and "`Tests`" and a pdf file of this thesis. In the first folder you will find the source code including a Xcode and a Dev-C++[1] project file and the binaries for Windows and Macintosh systems. The second folder contains the log files of our user tests.

[1]Open source C/C++ IDE by Bloodshed Software : http://www.bloodshed.net/devcpp.html

Bibliography

Axiotron. Axiotron Modbook. URL http://www.axiotron.com/. [Online; accessed 31-August-2008].

Ronald Azuma. A Survey of Augmented Reality. *Presence: Teleoperators and Virtual Environments*, 6(4):355–385, 1997.

Stefano Burigat and Luca Chittaro. Location-aware visualization of VRML models in GPS-based mobile guides. In *Web3D '05: Proceedings of the tenth international conference on 3D Web technology*, pages 57–64, New York, NY, USA, 2005. ACM. ISBN 1-59593-012-4. doi: http://doi.acm.org/10.1145/1050491.1050499.

M. Castrillón Santana, O. Déniz Suárez, M. Hernández Tejera, and C. Guerra Artal. ENCARA2: Real-time Detection of Multiple Faces at Different Resolutions in Video Streams. *Journal of Visual Communication and Image Representation*, pages 130–140, April 2007.

Jeremy R. Cooperstock, Koichiro Tanikoshi, and William Buxton. Turning Your Video Monitor into a Virtual Window. In *Proc. of IEEE PACRIM, Visualisation and Signal Processing*, March 16 1995. URL http://citeseer.ist.psu.edu/30824.html;ftp://dgp.utoronto.ca/pub/jer/ieee95.ht.ps.Z.

Carolina Cruz-Neira, Daniel J. Sandin, Thomas A. DeFanti, Robert V. Kenyon, and John C. Hart. The CAVE: audio visual experience automatic virtual environment. *Commun. ACM*, 35(6):64–72, 1992. ISSN 0001-0782. doi: http://doi.acm.org/10.1145/129888.129892.

Carolina Cruz-Neira, Daniel J. Sandin, and Thomas A. DeFanti. Surround-screen projection-based virtual reality:

the design and implementation of the CAVE. In *SIGGRAPH '93: Proceedings of the 20th annual conference on Computer graphics and interactive techniques*, pages 135–142, New York, NY, USA, 1993. ACM. ISBN 0-89791-601-8. doi: http://doi.acm.org/10.1145/166117.166134.

Marek Czernuszenko, Dave Pape, Daniel Sandin, Tom DeFanti, Gregory L. Dawe, and Maxine D. Brown. The ImmersaDesk and Infinity Wall projection-based virtual reality displays. *SIGGRAPH Comput. Graph.*, 31(2):46–49, 1997. ISSN 0097-8930. doi: http://doi.acm.org/10.1145/271283.271303.

Neil A. Dodgson. Variation and extrema of human interpupillary distance. In *Proceedings of the SPIE Volume 5291, Stereoscopic Displays and Virtual Reality Systems XI*, 2004.

George W. Fitzmaurice. Situated information spaces and spatially aware palmtop computers. *Commun. ACM*, 36 (7):39–49, 1993. ISSN 0001-0782. doi: http://doi.acm.org/10.1145/159544.159566.

Tim Gleue and Patrick Dähne. Design and implementation of a mobile device for outdoor augmented reality in the archeoguide project. In *VAST '01: Proceedings of the 2001 conference on Virtual reality, archeology, and cultural heritage*, pages 161–168, New York, NY, USA, 2001. ACM. ISBN 1-58113-447-9. doi: http://doi.acm.org/10.1145/584993.585018.

GLUT. GLUT - The OpenGL Utility Toolkit. URL `http://www.opengl.org/resources/libraries/glut/`. [Online; accessed 31-August-2008].

Steven J. Gortler, Radek Grzeszczuk, Richard Szeliski, and Michael F. Cohen. The lumigraph. In *SIGGRAPH '96: Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pages 43–54, New York, NY, USA, 1996. ACM. ISBN 0-89791-746-4. doi: http://doi.acm.org/10.1145/237170.237200.

Sir William Rowan Hamilton. On a new Species of Imaginary Quantities connected with a theory of Quaternions. In *Proceedings of the Royal Irish Academy, Vol. 2*, pages 424–434, 1844.

Anders Henrysson and Mark Ollila. UMAR: Ubiquitous Mobile Augmented Reality. In *MUM '04: Proceedings of the 3rd international conference on Mobile and ubiquitous multimedia*, pages 41–45, New York, NY, USA, 2004. ACM. ISBN 1-58113-981-0. doi: http://doi.acm.org/10.1145/1052380.1052387.

Anders Henrysson, Mark Ollila, and Mark Billinghurst. Mobile phone based AR scene assembly. In *MUM '05: Proceedings of the 4th international conference on Mobile and ubiquitous multimedia*, pages 95–102, New York, NY, USA, 2005. ACM. ISBN 0-473-10658-2. doi: http://doi.acm.org/10.1145/1149488.1149504.

Anders Henrysson, Mark Billinghurst, and Mark Ollila. AR Tennis. In *SIGGRAPH '06: ACM SIGGRAPH 2006 Emerging technologies*, page 1, New York, NY, USA, 2006. ACM. ISBN 1-59593-364-6. doi: http://doi.acm.org/10.1145/1179133.1179135.

Koichi Hirota and Yuya Saeki. Cross-section Projector: Interactive and Intuitive Presentation of 3D Volume Data using a Handheld Screen. In *3D User Interfaces, 2007. 3DUI '07. IEEE Symposium on*, 2007. doi: http://doi.ieeecomputersociety.org/10.1109/3DUI.2007.340775.

Human Interface Technology Lab. ARToolkit. URL `http://www.hitl.washington.edu/artoolkit/`. [Online; accessed 31-August-2008].

id Software. id Software : Quake. URL `http://www.idsoftware.com/games/quake/quake/`. [Online; accessed 31-August-2008].

Intel. OpenCV : Open Source Computer Vision Library. URL `http://www.intel.com/technology/computing/opencv/`. [Online; accessed 31-August-2008].

David Johnson, Christopher Berthiaume, and Bryan Witkowski. ARToolkit Patternmaker. URL `http://www.cs.utah.edu/gdc/projects/augmentedreality/`. [Online; accessed 31-August-2008].

Jr. Joseph J. LaViola. A Discussion of Cybersickness in Virtual Environments. Technical report, Providence, RI, USA, 1999.

Eunsoo Jung, Sujin Oh, and Yanghee Nam. Handheld AR indoor guidance system using vision technique. In *VRST '07: Proceedings of the 2007 ACM symposium on Virtual reality software and technology*, pages 47–50, New York, NY, USA, 2007. ACM. ISBN 978-1-59593-863-3. doi: http://doi.acm.org/10.1145/1315184.1315190.

Hirokazu Kato and Mark Billinghurst. Marker Tracking and HMD Calibration for a Video-Based Augmented Reality Conferencing System. In *IWAR '99: Proceedings of the 2nd IEEE and ACM International Workshop on Augmented Reality*, page 85, Washington, DC, USA, 1999. IEEE Computer Society. ISBN 0-7695-0359-4.

Johnny Chung Lee. Head Tracking for Desktop VR Displays using the Wii Remote. URL `http://www.cs.cmu.edu/~johnny/projects/wii/`. [Online; accessed 31-August-2008].

R. Lienhart and J. Maydt. An extended set of Haar-like features for rapid object detection. volume 1, pages I–900–I–903 vol.1, 2002. doi: 10.1109/ICIP.2002.1038171. URL `http://dx.doi.org/10.1109/ICIP.2002.1038171`.

Michael E. McCauley and Thomas J. Sharkey. Cybersickness: perception of self-motion in virtual environments. *Presence: Teleoper. Virtual Environ.*, 1(3):311–318, 1992. ISSN 1054-7460.

Michael McKenna. Interactive viewpoint control and three-dimensional operations. In *SI3D '92: Proceedings of the 1992 symposium on Interactive 3D graphics*, pages 53–56, New York, NY, USA, 1992. ACM. ISBN 0-89791-467-8. doi: http://doi.acm.org/10.1145/147156.147163.

Miguel A. Nacenta, Satoshi Sakurai, Tokuo Yamaguchi, Yohei Miki, Yuichi Itoh, Yoshifumi Kitamura, Sriram Subramanian, and Carl Gutwin. E-conic: a perspective-aware interface for multi-display environments. In *UIST '07: Proceedings of the 20th annual ACM symposium on User interface software and technology*, pages 279–288, New

York, NY, USA, 2007. ACM. ISBN 978-1-59593-679-2. doi: http://doi.acm.org/10.1145/1294211.1294260.

Adrijan S. Radikovic, John J. Leggett, John Keyser, and Roger S. Ulrich. Artificial window view of nature. In *CHI '05: CHI '05 extended abstracts on Human factors in computing systems*, pages 1993–1996, New York, NY, USA, 2005. ACM. ISBN 1-59593-002-7. doi: http://doi.acm.org/10.1145/1056808.1057075.

Nate Robins. The GLM Library. URL `http://www.xmission.com/~nate/`. [Online; accessed 31-August-2008].

RWTH Aachen. CAVE at the RWTH Aachen. URL `http://www.rz.rwth-aachen.de/ca/c/nsi/lang/de/#snapshots`. [Online; accessed 31-August-2008].

Bernt Schiele, Tony Jebara, and Nuria Oliver. Sensory-Augmented Computing: Wearing the Museum's Guide. *IEEE Micro*, 21(3):44–52, 2001. ISSN 0272-1732. doi: http://dx.doi.org/10.1109/40.928764.

Dieter Schmalstieg and Daniel Wagner. A Handheld Augmented Reality Museum Guide. In *Proceedings of IADIS International Conference on Mobile Learning 2005*, 2005.

Ting Shan. Website of Ting Shan. URL `http://www.nicta.com.au/people/shant`. [Online; accessed 31-August-2008].

Bruce Thomas, Ben Close, John Donoghue, John Squires, Phillip De Bondi, and Wayne Piekarski. First Person Indoor/Outdoor Augmented Reality Application: ARQuake. *Personal Ubiquitous Comput.*, 6(1):75–86, 2002. ISSN 1617-4909. doi: http://dx.doi.org/10.1007/s007790200007.

Trolltech. Qt Cross-Platform Application Framework. URL `http://trolltech.com/products/qt`. [Online; accessed 31-August-2008].

Michael Tsang, George W. Fitzmaurice, Gordon Kurtenbach, Azam Khan, and Bill Buxton. Boom chameleon: simultaneous capture of 3D viewpoint, voice and gesture annotations on a spatially-aware display. In *UIST*

'02: Proceedings of the 15th annual ACM symposium on User interface software and technology, pages 111–120, New York, NY, USA, 2002. ACM. ISBN 1-58113-488-6. doi: http://doi.acm.org/10.1145/571985.572001.

P. Viola and M. Jones. Rapid object detection using a boosted cascade of simple features. 2001. URL `citeseer.ist.psu.edu/viola01rapid.html`.

Theo Watson. videoInput - a free windows video capture library. URL `http://muonics.net/school/spring05/videoInput/`. [Online; accessed 31-August-2008].

Lance Williams. Casting curved shadows on curved surfaces. *SIGGRAPH Comput. Graph.*, 12(3):270–274, 1978. ISSN 0097-8930. doi: http://doi.acm.org/10.1145/965139.807402.

Ben Woodhouse. GLee (GL Easy Extension library). URL `http://elf-stone.com/glee.php`. [Online; accessed 31-August-2008].

Ka-Ping Yee. Peephole displays: pen interaction on spatially aware handheld computers. In *CHI '03: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 1–8, New York, NY, USA, 2003. ACM. ISBN 1-58113-630-7. doi: http://doi.acm.org/10.1145/642611.642613.

Index

Typeset September 9, 2008