

Diplomarbeit

Comprehensive Calibration Procedures for Augmented Reality

unter der Leitung von
Univ. Prof. Dipl.-Ing. Dr.techn. Werner Purgathofer,
Institut 186 für Computergraphik und Algorithmen,

unter Mitbetreuung von
Dipl.-Ing. Dr.techn. Anton L. Fuhrmann,
Forschungszentrum VRVis in Wien,

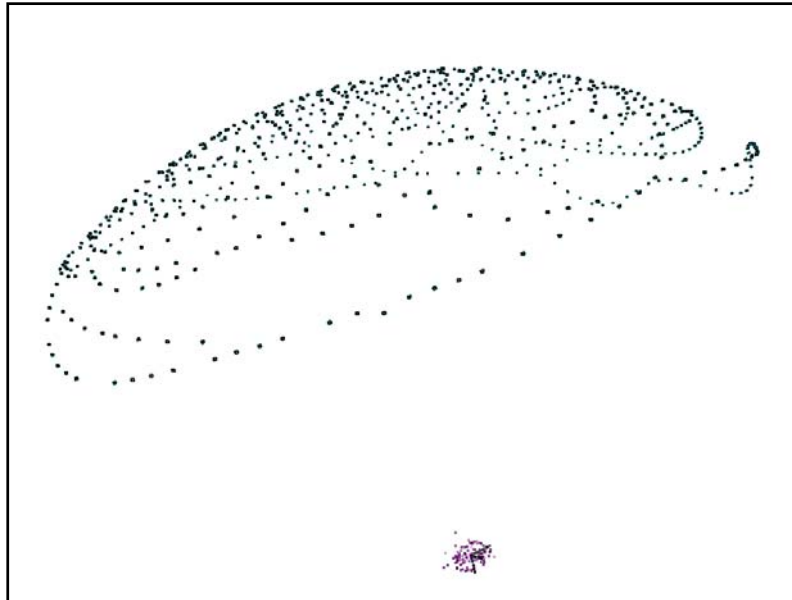
eingereicht
an der Technischen Universität Wien,
Fakultät für Technische Naturwissenschaften und Informatik

von
Rainer Splechtna,
Matrikelnummer 9026565,
Hauptplatz 15,
3910 Zwettl.

Rainer Splechna

Comprehensive Calibration Procedures for Augmented Reality

Master Thesis



Supervised by

Dipl.-Ing. Dr.techn. Anton L. Fuhrmann,
VRVis Research Center in Vienna, Austria

and

Univ. Prof. Dipl.-Ing. Dr.techn. Werner Purgathofer,
Institute of Computer Graphics and Algorithms
Computer Graphics Group
Vienna University of Technology, Vienna, Austria

Abstract

English:

Augmented reality (AR) systems combine three-dimensional computer-generated images with the view of the real environment in order to make unseen objects visible or to present additional information. Since the user of an AR system sees both the real and the virtual environment, such a system needs more adjustments to work properly than immersive virtual environments, where the user is only presented with the virtual environment. Hence the virtual environment has to be properly aligned to the real, physical world, to be perceived as an augmentation of the real environment. The process to achieve this alignment is called *calibration*.

This work presents a comprehensive set of calibration procedures that consists of all tasks necessary for the calibration of devices commonly used in an AR system, so that correct augmentation of the real environment, i.e. correct alignment of virtual and real world can be achieved. This includes procedures for calibrating projective and head-mounted displays, tracking systems, tracked input devices and props. Since the *calibration* process has to be done at least once for every hardware set-up, but may have to be repeated in part or completely for each user, prop or device to be included both in the real and the virtual world, we strived for a method that not only delivers good registration results but also can be applied fast and easily.

The proposed method unifies the necessary tasks of world-to-augmentation alignment — display calibration and registration of tracked and static props — in one, interactive set-up process, which can be conducted in short time and by an untrained user.

Deutsch:

Augmented reality (AR) Systeme verschmelzen dreidimensionale, computer-generierte Bilder mit dem Blick des Benutzers auf die reale Umgebung, um nicht sichtbare Objekte zu visualisieren oder um zusätzliche Informationen darstellen zu können. Da der Benutzer eines AR-Systems sowohl die reale als auch die virtuelle Umgebung gleichzeitig wahrnimmt, benötigt solch ein System, um überzeugend zu wirken, eine genauere Abstimmung als klassische virtuelle Systeme, bei denen der Benutzer ausschließlich mit der virtuellen Umgebung konfrontiert wird. Daher muss die virtuelle Umgebung genau auf die reale,

physische Welt ausgerichtet werden, um als ‚Erweiterung‘ der realen Umgebung wahrgenommen zu werden. Der Prozess, diese Übereinstimmung herzustellen, wird *Kalibrierung* genannt.

Diese Diplomarbeit beschreibt eine umfassende Sammlung von Kalibrierungsprozeduren, welche alle Aufgaben abdeckt, die nötig sind, um die Geräte, die gebräuchlicherweise in AR-Systemen Verwendung finden, zu kalibrieren, sodass der Eindruck der ‚Erweiterung‘ der realen Umgebung, d.h. die Übereinstimmung von virtueller und realer Welt, erzielt werden kann. Dies beinhaltet Prozeduren zur Kalibrierung von Projektionsschirmen, head-mounted displays, Tracking-Systemen, getrackten Eingabegeräten und Objekten. Da der Kalibrierungsprozess zumindest einmal für jedes Hardware-Setup durchgeführt werden muss, aber für jeden Benutzer, jedes Objekt, jedes Gerät, das sowohl der realen als auch der virtuellen Umgebung hinzugefügt werden soll, zum Teil oder zur Gänze wiederholt werden muss, waren wir bestrebt, eine Methode zu entwickeln, die nicht nur gute Registrierungsergebnisse erzielt, sondern auch schnell und einfach angewendet werden kann.

Die vorgeschlagene Methode vereinigt die durchzuführenden Aufgaben zur Abstimmung von realer und virtueller Welt — Kalibrierung von Displays und Registrierung von getrackten oder statischen Objekten — in einem einzigen, interaktiven Setup-Prozess, der in kurzer Zeit und auch von nicht mit dem System vertrauten Benutzern durchgeführt werden kann.

Contents

1 Introduction	1
2 Problem Statement	4
2.1. The Registration Problem	4
2.2. Sources of error and focus of the calibration process	6
2.3. Static registration (static error) in Augmented Reality	7
2.4. Prerequisites.....	9
2.5. Calibration Tasks (to achieve good static registration)	10
2.6. Human-Computer Interaction (HCI) aspects	11
3 Related Work	12
3.1. Overview	12
3.2. Display Device Registration	13
3.2.1. The “Boresight Method”	14
3.2.2. The “Shooting Gallery”	17
3.2.3. The Dynamic Calibration Process	20
3.2.4. Single Point Active Alignment Method (SPAAM)	23
3.2.5. Projection Device Calibration.....	26
3.3. Object Calibration	30
3.3.1. Calibration with Reference Frame.....	30
3.3.2. Calibration with Pointing Device	31
3.4. Summary and Conclusion	34
4 Calibration procedures	36
4.1. Stylus Calibration	36
4.2. Display Device Calibration.....	40
4.2.1. The Studierstube offaxis camera model	40
4.2.2. Calibrating See-Through Head-Mounted Displays	42
4.2.3. Calibrating projection systems.....	52
4.3. Registration of Tracker to World Coordinate System	57
4.4. Calibration of Props	61

5 Implementation	66
5.1. The Studierstube System - Implementation of the user interface.....	66
5.1.1. Software architecture	66
5.1.2. Hardware support.....	67
5.1.3. Application programmer's interface	69
5.2. Human-Computer Interaction (HCI) aspects	70
5.2.1. Paths through the Calibration Process.....	71
5.2.2. User guidance	73
5.3. OpenTracker - An XML based Open Architecture for Reconfigurable Tracking	76
5.4. Minimizing Functions - Direction Set (Powell's) Methods in Multidimensions	82
6 Results and Conclusion	86
6.0. Tracking System	86
6.1. Test setup and evaluation of stylus calibration.....	87
6.2. Test setup and evaluation of HMD calibration	90
6.3. Test setup and evaluation of Projection calibration.....	94
6.4. Conclusion.....	96
7 Future Work.....	97
7.1. Additional constraints and thresholds.....	97
7.2. Faster HMD calibration	98
7.3. Dynamic registration	98
Bibliography and References.....	100

Chapter 1

Introduction

What is Augmented Reality?

Augmented Reality (AR) is a variation of *Virtual Environments* (VE), or Virtual Reality as it is more commonly called. VE technologies completely immerse a user inside a synthetic environment. While immersed, the user cannot see the real world around him. In contrast, AR allows the user to see the real world, with virtual objects superimposed upon or composited with the real world. Therefore, AR supplements reality, rather than completely replacing it. Ideally, it would appear to the user that the virtual and real objects coexisted in the same space, similar to the effects achieved in the film "The Phantom Menace".

AR can be thought of as the "middle ground" between VE (completely synthetic) and telepresence (completely real) [Milgram94a][Milgram94b].

Some researchers define AR in a way that requires the use of Head-Mounted Displays (HMDs). To avoid limiting AR to specific technologies, [Azuma97a] defines AR as systems that have the following three characteristics:

- 1) Combines real and virtual
- 2) Interactive in real time
- 3) Registered in 3-D

This definition allows other technologies besides HMDs while retaining the essential components of AR. For example, it does not include film or 2-D overlays. Films like "Jurassic Park" feature photo realistic virtual objects seamlessly blended with a real environment in 3-D, but they are not interactive media. 2-D virtual overlays on top of live video can be done at interactive rates, but the overlays are not combined with the real world in 3-D. However, this definition does allow monitor-based interfaces, projection display devices, monocular systems, see-through HMDs, and various other combining technologies.

AR has the potential to enhance a user's perception of and interaction with the real world. The virtual objects may display information that the user cannot directly detect with her senses. Two examples where this technology could help are medical applications and the assembly and repair of complicated mechanical devices.

Why do we need Calibration/Registration? What's the difference?

First lets look a little closer at the terms calibration and registration:

The term registration is frequently used as synonym for calibration and vice versa. Where registration rather describes a state — the precise alignment and synchronization of two or more sensory elements [Azuma97b] — and calibration mostly refers to a process or action. So we could say a registration is the result of a calibration. In the following text this definition is used, but both terms mainly stay synonymous for each other.

In order for augmented reality to be effective and accepted by the user the real and computer-generated objects must be accurately positioned relative to each other and properties of certain devices must be accurately specified. When a user perceives two different loci of interaction, one given by the real image of his hand and one by the

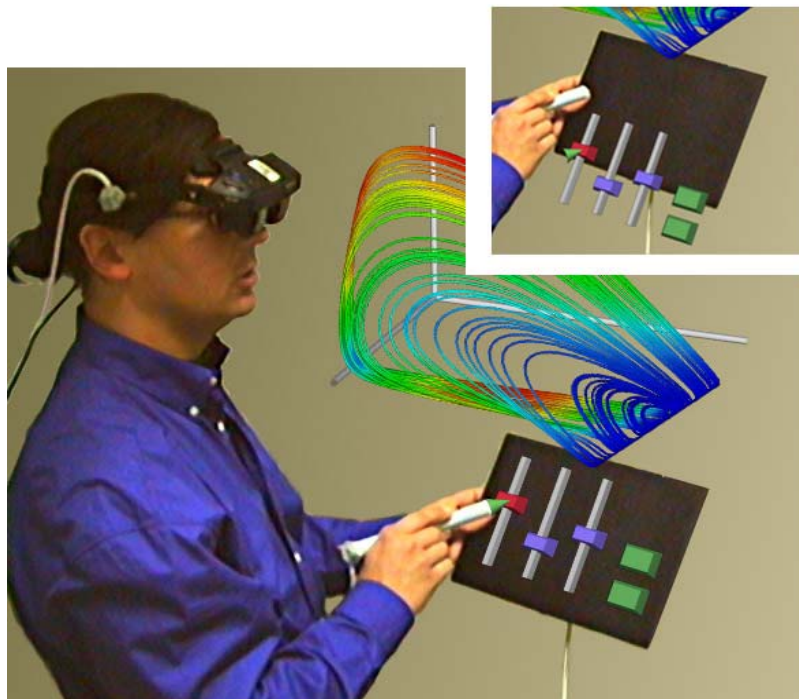


Figure 1 Personal Interaction Panel (inset miscalibrated)

virtual image on a different position, the perceived clues conflict and hand-to-eye coordination is severely impaired. Figure 1 shows the *Personal Interaction Panel* [Szalavári97], a simple tracked board, which is augmented with interaction elements to act as a kind of instrument panel for controlling the parameters of a scientific visualization. The big image shows correct overlaid computer graphics, the inset a misalignment between the virtual sliders and the physical board. Clearly controlling a virtual slider while seeing the real and virtual pen in different places is irritating and leads to problems when interacting with the virtual input elements.

Many Augmented Reality applications will not be accepted unless virtual objects are accurately registered with their real counterparts, but good registration is difficult, because of the high resolution of the human visual system and its sensitivity to small misalignments at edges.

Static vs. Dynamic Registration

Registration errors fall into two categories: *static* errors, which occur even when the user remains still, and *dynamic* errors caused by system delays when the user moves. Dynamic errors are usually the largest errors. Nevertheless correct static registration is highly important. It is the fundamental step in achieving correct overall registration of an AR system and serves as basis for dynamic registration, which only makes sense when good static registration is provided.

Goal: Easy, user-guiding “calibration wizard”

The aim of the described method is to provide a fast, comprehensive calibration procedure. The method should present itself to the user as an interactive sequence of simple, independent tasks, which can easily be performed even without knowledge of the actual parameters they modify. Setting up an augmented reality installation becomes therefore a reliable and reproducible routine. Furthermore no additional hardware, but that already required for a typical AR-System setup, should be needed for any of the calibration tasks.

Chapter 2

Problem Statement

2.1. The Registration Problem

One of the most basic problems currently limiting Augmented Reality applications is the registration problem. The objects in the real and virtual worlds must be properly aligned with respect to each other, or the illusion that the two worlds coexist will be compromised. More seriously, many applications *demand* accurate registration (e.g. medical applications). Without accurate registration, Augmented Reality will not be accepted in many applications.

Registration problems also exist in Virtual Environments, but they are not nearly as serious because they are harder to detect than in Augmented Reality. Since the user only sees virtual objects in VE applications, registration errors result in visual-kinesthetic and visual-proprioceptive conflicts. Such conflicts between different human senses may be a source of motion sickness [Pausch92]. Because the kinesthetic and proprioceptive systems are much less sensitive than the visual system, visual-kinesthetic and visual-proprioceptive conflicts are less noticeable than visual-visual conflicts. For example, a user wearing a closed-view HMD might hold up her real hand and see a virtual hand. This virtual hand should be displayed exactly where she would see her real hand, if she were not wearing an HMD. But if the virtual hand is wrong by five millimeters, she may not detect that unless actively looking for such errors. The same error is much more obvious in a see-through HMD, where the conflict is visual-visual [Azuma97a].

Furthermore, a phenomenon known as *visual capture* [Welch78] makes it even more difficult to detect such registration errors. Visual capture is the tendency of the brain to believe what it sees rather than what it feels, hears, etc. That is, visual information tends to override all other senses. When watching a television program, a viewer believes the sounds come from the mouths of the actors on the screen, even though they actually come from a speaker in the TV. Ventriloquism works because of visual capture. Similarly, a user might believe

that her hand is where the virtual hand is drawn, rather than where her real hand actually is, because of visual capture. This effect increases the amount of registration error users can tolerate in Virtual Environment systems. If the errors are systematic, users might even be able to adapt to the new environment, given a long exposure time of several hours or days [Welch78].

Augmented Reality demands much more accurate registration than Virtual Environments [Azuma93]. Imagine the same scenario of a user holding up her hand, but this time wearing a see-through HMD. Registration errors now result in visual-visual conflicts between the images of the virtual and real hands. Such conflicts are easy to detect because of the resolution of the human eye and the sensitivity of the human visual system to differences. Even tiny offsets in the images of the real and virtual hands are easy to detect.

What angular accuracy is needed for good registration in Augmented Reality?

[Azuma97a] gives the following example:

“Take out a dime [or a 10 euro cent coin] and hold it at arm's length, so that it looks like a circle. The diameter of the dime covers about 1.2 to 2.0 degrees of arc, depending on your arm length. In comparison, the width of a full moon is about 0.5 degrees of arc! Now imagine a virtual object superimposed on a real object, but offset by the diameter of the full moon. Such a difference would be easy to detect.”

Thus, the angular accuracy required is a small fraction of a degree. The lower limit is bounded by the resolving power of the human eye itself. The central part of the retina is called the *fovea*, which has the highest density of color-detecting cones, about 120 per degree of arc, corresponding to a spacing of half a minute of arc [Jain89]. Observers can differentiate between a dark and light bar grating when each bar subtends about one minute of arc, and under special circumstances they can detect even smaller differences [Doenges85]. However, existing HMD trackers and displays are not capable of providing one minute of arc in accuracy, so the present achievable accuracy is much worse than that ultimate lower bound. In practice, errors of a few pixels are detectable in modern HMDs [Azuma97a].

Registration of real and virtual objects is not limited to AR. Special-effects artists seamlessly integrate computer-generated 3-D objects with live actors in film and video. The difference lies in the amount of control available. With film, a director can carefully plan each shot, and artists can spend hours per frame, adjusting each by hand if necessary, to achieve perfect registration [Gibson02][BouJou02]. As an interactive medium, AR is far more difficult to

work with. The AR system cannot control the motions of the HMD wearer. The user looks where she wants, and the system must respond within tens of milliseconds.

2.2. Sources of error and focus of the calibration process

Registration errors are difficult to adequately control because of the high accuracy requirements and the numerous sources of error. These sources of error can be divided into two types: static and dynamic. *Static* errors are the ones that cause registration errors even when the user's viewpoint and the objects in the environment remain completely still. *Dynamic* errors are the ones that have no effect until either the viewpoint or the objects begin moving.

For current HMD-based systems, dynamic errors are by far the largest contributors to registration errors, but static errors cannot be ignored either. See [Holloway95] for a thorough analysis of the sources and magnitudes of registration errors.

When using projection systems, dynamic errors regarding the viewpoint are not as much a contributing factor to registration errors, because the rotation of the users head, normally the fastest movement, does not directly translate into a corresponding change of the viewing direction as in the case of HMD-based systems.

We decided to focus on static registration errors, because correct static registration is the basis and starting point for solving the whole registration problem, i.e. without static registration dynamic registration is not possible, whereas working with a system that is only registered statically is absolutely feasible. Furthermore no calibration procedure at all had been implemented for our augmented reality system before (see section 2.4), so we had to deal with static registration errors first. Dynamic errors, which are caused by system lags, will be addressed in future work.

2.3. Static registration (static error) in Augmented Reality

The four main sources of static errors as stated by [Azuma94] are:

- Optical distortion
- Errors in the tracking system
- Mechanical misalignments
- Incorrect viewing parameters (e.g., field of view, tracker-to-eye position and orientation, interpupillary distance)

1) Distortion in the optics:

Optical distortions exist in most camera and lens systems, both in the cameras that record the real environment and in the optics used for the display. Because distortions are usually a function of the radial distance away from the optical axis, wide field-of-view displays can be especially vulnerable to this error. Near the center of the field-of-view, images are relatively undistorted, but far away from the center, image distortion can be large. For example, straight lines may appear curved. *In a see-through HMD with narrow field-of-view displays, the optical combiners add virtually no distortion, so the user's view of the real world is not warped.* However, the optics used to focus and magnify the graphic images from the display monitors can introduce distortion. This mapping of distorted virtual images on top of an undistorted view of the real world causes static registration errors. The cameras and displays may also have nonlinear distortions that cause errors [Deering92].

Though compensation of optical distortions is often possible, [Holloway95] determined that for typical head motion the additional system delay required by the distortion compensation adds more registration error than the distortion compensation removes. Furthermore we use a *see-through HMD*, which doesn't exhibit much distortion. Hence our calibration approach doesn't compensate for nonlinear optical distortions.

2) Errors in the tracking system:

Errors in the reported outputs from the tracking and sensing systems are often the most serious type of static registration errors. These distortions are not easy to measure and eliminate, because that requires another "3-D ruler" that is more accurate than the tracker being tested. These errors are often non-

systematic and difficult to fully characterize. Almost all commercially available tracking systems are not accurate enough to satisfy the requirements of AR systems. With the advent of accurate optical tracking systems at least nearly linear behavior within the working volume of the tracker is achieved. The downside of optical trackers at the moment is the fact, that the user is not really allowed to freely position herself within the tracked area, because the markers attached to the HMD and/or other tracked props, cannot always be detected by the optical sensors (normally cameras) due to occlusions between the body of the user or markers attached to another prop. This problem gets even worse, when using a multi-user-environment like the Studierstube system, where the chance of occlusions rise due to more obstructing bodies moving within the working area.

Nevertheless we rely on the very good linearity of the optical tracking system, which was used to evaluate our calibration procedures. The linearization problem of magnetical tracking systems is out of scope of this work and dealt with in publications like [Kindratenko99] or [Livingston97].

3) Mechanical misalignments:

Mechanical misalignments are discrepancies between the model or specification of the hardware and the actual physical properties of the real system. For example, the combiners, optics, and monitors in an optical see-through HMD may not be at the expected distances or orientations with respect to each other. If the frame is not sufficiently rigid, the various component parts may change their relative positions as the user moves around, causing errors. Mechanical misalignments can cause subtle changes in the position and orientation of the projected virtual images that are difficult to compensate. While some alignment errors can be calibrated, for many others it may be more effective to *"build it right" initially* [Azuma97a].

4) Incorrect viewing parameters:

Incorrect viewing parameters, the last major source of static registration errors, can be thought of as a special case of alignment errors where calibration techniques can be applied. Viewing parameters specify how to convert the reported head or camera locations into viewing matrices used by the scene generator to draw the graphic images. For an HMD-based system, these parameters include:

- Center of projection and viewport dimensions

- Offset, both in translation and orientation, between the location of the head tracker and the user's eyes
- Field of view

Incorrect viewing parameters cause systematic static errors. Take the example of a head tracker located above a user's eyes. If the vertical translation offsets between the tracker and the eyes are miscalibrated, all the virtual objects will appear lower or higher than they should.

The retrieval of correct viewing parameters is the main topic of this thesis.

2.4. Prerequisites

Development environment: The Studierstube System

The *Studierstube* augmented reality system [Schmalstieg00] project tries to address the question of how to use three-dimensional interactive media in a general work environment, where a variety of tasks are carried out simultaneously. In essence, the main focus of the project is the search for a 3D user interface metaphor as powerful as the desktop metaphor for 2D.

The *Studierstube* augmented reality system uses different techniques to overlay computer graphics onto a user's view of the real world. Primarily we have been using see-through head-mounted displays to accommodate individual viewpoints for a multi-user scenario, but *Studierstube* also supports projection display devices such as the Virtual Table or a stereo projection wall. Our concept includes at least two tracked interaction devices per user: a pen and pad combination called the *Personal Interaction Panel* (PIP) and supports tracking technologies ranging from magnetical to optical and inertial methods.

Since each hardware set-up can consist of almost any combination of the display and tracking technologies mentioned above, most of the parameters of display and tracking system will depend on the set-up. Altering the hardware or setting an environment up in a new location will require a complete calibration process. Studierstube is already in use as development environment in different research institutes in Europe and the United States, but not two of them are employing exactly the same display/tracker/input device combination. In most cases the calibration process proved to be the biggest obstacle when setting up a new site. Even when using the same hardware in a different location – e.g. as

demonstration at a scientific conference or a trade show – altered positions of tracker origin and projector lead to unnecessary delays in the set-up.

Our method implies two prerequisites:

- Firstly, a tracking system capable of delivering accurate results over the working volume, and therefore minimizing the errors in the tracking system, stated above as source of static errors. In the case of a magnetical tracking system, for example, one has to assure linear behavior by appropriate methods like [Kindratenko99] or [Livingston97]. When using an optical tracking system like the DTrack optical tracker [Arto1] accurate (nearly linear) results over the working volume are inherently guaranteed by the tracking system (see section 6.0 for details).
- Secondly, we need a tracked pointing device with one button and precisely defined hotspot (i.e. a point that defines where an action is executed when pressing the button). The Studierstube system provides the so-called pen (stylus) as pointing device, which perfectly fits our requirements.

2.5. Calibration Tasks (to achieve good static registration)

To achieve good and comprehensive static registration for the Studierstube augmented reality system, all devices, which are used within the Augmented Environment, have to be calibrated. The whole calibration process consists of the following calibration (sub)-tasks:

- Stylus (Pointing Device) Calibration
- Display Device Calibration:
 - a) Optical See-Through Head-Mounted-Displays (HMDs)
 - b) Projection Systems
- Calibration of Tracker to Virtual (World) Coordinate System
- Calibration of tracked or stationary props (real objects)

Firstly, we have to determine the hotspot of the pointing device. This step, called stylus calibration, is performed first, because all following tasks depend on inputs, sampled at the pen's hotspot.

The second task performs the calibration of the used display device. Depending on the used Studierstube setup either the HMD- or projection system calibration is executed. This part of the calibration process determines the viewing parameters of the display device of choice.

With the calibration of tracker to virtual (world) coordinate system the user may define the origin and orientation of the world coordinate system. Per default the world coordinate system of Studierstube is congruent with the coordinate system of the tracker (e.g. if using a magnetical tracker, the world origin is typically somewhere within the tracker's emitter), but for convenience reasons it is preferable to place the world origin in the middle of the user's working volume and to align some or all axes with the corresponding axes of the projection screen.

For every physical prop, that should be augmented by the Studierstube system, regardless if tracked or stationary, the prop calibration task has to be performed, to achieve correct registration between the physical prop and its corresponding virtual representation in the Studierstube system.

2.6. Human-Computer Interaction (HCI) aspects

As stated above, our goal is to provide a comprehensive 'calibration suite', which represents an interactive sequence of simple, independent tasks, which can easily be performed. An important point at this is the human-computer-interface. To ensure, that an untrained user can also perform each task, we have to develop an intuitive, more or less self-descriptive user interface. Additionally the user should be interactively guided through each of the tasks. Mechanisms that can be utilized here are input constraints combined with visual feedback, which should lead the user to obtaining only valid data or at least give her conclusive hints how to perform the given task.

Chapter 3

Related Work

3.1. Overview

Some of the here-discussed problems have already been addressed separately:

[Holloway95] analyzed different aspects of registration error and [Hoff00] and Vincent presented analysis of head pose accuracy for AR applications.

[Bajura97] proposed calibration for video see-through systems based on tracking known features in the working environment. Since only optical see-through Head-Mounted Displays (STHMDs) are used up until now with Studierstube, we take a closer look at interactive calibration methods for STHMDs, which have been described by [Azuma94] and Bishop, [Oishi96] and Tachi, [McGarrrity99] and Tuceryan and [Tuceryan00] and Navab among others.

[Summers99] et al. address registration and calibration of an experimental see-through projection-based system that employs shutter glasses. [Czernuszenko98] et al. describe a tracker calibration approach for back-projection systems, trying to compensate for the non-linear behaviour of magnetical trackers.

[Whitaker95] et al. describe a pointer calibration method together with two different object calibration methods for augmented reality. Automatic, image-based object identification and registration methods have been proposed by [Billinghurst99] and [Rekimoto98]. Image-based methods have the potential to work completely without user intervention, but imply a video-based AR set-up, which until now has not been integrated in Studierstube.

3.2. Display Device Registration

Correct registration of the display device is a prerequisite for an Augmented Reality environment setup to be accepted by the user of such a system. A preferred display device for Augmented Reality environments is a STHMD. Hence many different approaches to achieving good registration for STHMDs have been discussed in the past.

In some systems, the viewing parameters are estimated by manual adjustments, in a non-systematic fashion. Such approaches proceed as follows: place a real object in the environment and attempt to register a virtual object with that real object. While wearing the HMD or positioning the cameras, move to one viewpoint or a few selected viewpoints and manually adjust the location of the virtual object and the other viewing parameters until the registration "looks right." This may achieve satisfactory results if the environment and the viewpoint remain static. However, such approaches require a skilled user and generally do not achieve robust results for many viewpoints. Achieving good registration from a single viewpoint is much easier than registration from a wide variety of viewpoints using a single set of parameters. Usually what happens is satisfactory registration at one viewpoint, but when the user walks to a significantly different viewpoint, the registration is inaccurate because of incorrect viewing parameters or tracker distortions. This means many different sets of parameters must be used, which is a less than satisfactory solution.

Another approach is to directly measure the parameters, using various measuring tools and sensors. For example, a commonly used optometrist's tool can measure the interpupillary distance. Rulers might measure the offsets between the tracker and eye positions. Cameras could be placed where the user's eyes would normally be in an optical see-through HMD. By recording what the camera sees, through the see-through HMD, of the real environment, one might be able to determine several viewing parameters. So far, direct measurement techniques have enjoyed limited success [Janin93].

For video-based systems, an extensive body of literature exists in the robotics and photogrammetry communities on camera calibration techniques. Such techniques compute a camera's viewing parameters by taking several pictures of an object of fixed and sometimes unknown geometry. These pictures must be taken from different locations. Matching points in the 2-D images with corresponding 3-D points on the object sets up mathematical constraints. With enough pictures, these constraints determine the viewing parameters and the 3-D location of the calibration object. Alternately, they can serve to drive an

optimization routine that will search for the best set of viewing parameters that fits the collected data.

View-based tasks are another approach to calibration. These ask the user to perform various tasks that set up geometric constraints. By performing several tasks, enough information is gathered to determine the viewing parameters. All view-based tasks rely upon the user accurately performing the specified task and assume the tracker error is negligible. If the tracking and sensing equipment is not accurate, then multiple measurements must be taken and optimizers used to find the "best-fit" solution. The following section will give a summary of some of these view-based methods, illustrating their strengths and shortcomings.

3.2.1. The “Boresight Method”

Overview and setup

[Azuma94] describes a procedure for the static calibration of an optical STHMD, which focuses on determining the correct viewing parameters.

They used an optoelectronic tracking system and an optical STHMD with a field of view (FOV) of about 30 degrees. The displays are LCD monitors containing 340x240 pixels each. The goal was to achieve a registration, which links one real object, a wooden crate (depicted in Figure 2 on the rightmost side), and one set of virtual objects, three orthogonal extruded squares that form a coordinate system (colored red, green and blue); i.e. the intersection of the three virtual bars and the front left corner of the crate should be registered, where the three bars run along the edges that touch the corner.

The calibration procedure consists of simple tasks that rely on geometric constraints and directly measure the desired viewing parameters. The steps, which systematically determine the viewing parameters, in order are:

- Measure the frame's location
- Determine the apparent center of the virtual image
- Measure the transformation between tracker space and eye space
- Measure the field-of-view (FOV)

Note that due to mechanical misalignments only the right eye of the HMD was used for the evaluation of the registration procedure.

The calibration procedure

Frame measurement: A digitization probe attached to a “hat” with four optical sensors returns the 3D position of the probe tip. Eight points on the frame edges where the red and green bars will lie are measured. A pair of orthogonal lines is fit through those points, also determining the axis going down the third edge.

Apparent center of virtual image: Since the center of the frame buffer need not be the center of the virtual image seen with the right eye, off-center projection is required to properly render the images. Assuming that the frame buffer covers the entire area visible through the optics, this center can be measured by drawing a 2D, non-head-tracked crosshair in the frame buffer (Figure 2). Four numbers specify this crosshair: the (X,Y) center coordinate, and the X and Y radii. The user determines the center by adjusting the X center and radius until the left and rightmost lines are equally spaced from the extreme visible edges of the display. This is tested by increasing the radius; both lines should disappear simultaneously or the center is incorrect. A similar procedure determines the Y center.

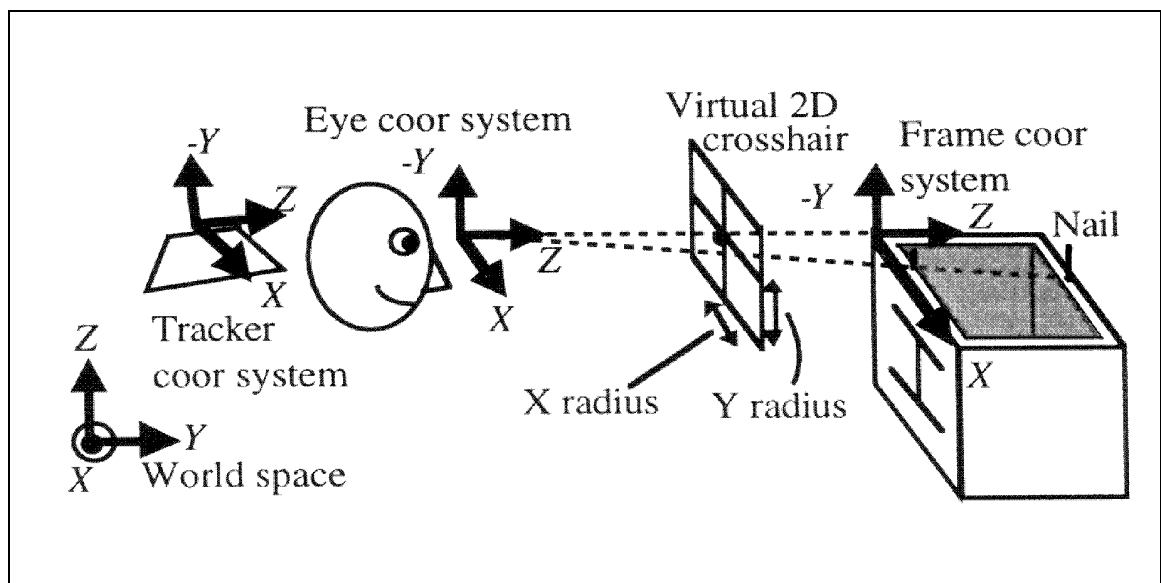


Figure 2 Sketch of the setup used for static registration, showing the virtual crosshair, the reference frame and coordinate systems [Azuma94]

Eye→Tracker transformation: This is measured by *the boresight operation*, where a user wearing the HMD looks straight down the left top edge of the

frame with his right eye. A thin pipe sticking out along the edge helps the user line up accurately. Simultaneously, he centers the virtual crosshair with the edges of the frame and aligns the horizontal and vertical crosshair lines with the edges of the frame. Then the Eye coordinate system has the same orientation as the Frame coordinate system, and the Z-axes coincide. Since the orientation of the frame relative to world coordinates is known, the desired Eye→Tracker orientation can be calculated.

The Eye→Tracker position offset is measured by the boresight and one additional task. The position of the corner of the frame in World space is known, due to step one. The head tracker returns the position of the tracker origin in world space. Therefore, a vector from the corner of the frame to the tracker origin can be drawn in World space, which can be transformed to Eye space by the now known rotations. Since Eye space and Frame space share the same orientation and their Z-axes coincide, the X and Y values of the vector in Eye space are the X and Y Eye→Tracker offsets, in Eye space. To determine the Z offset, one more operation is needed. Two nails are on top of the frame, one in front and one in the rear. While performing the boresight, the user must also position him so that the front nail covers the rear nail. The known locations of these two nails identify a specific distance along the frame's Z-axis where the user's eye must be. Subtracting that from the corner→tracker vector in Eye space yields the Z component of the Eye→Tracker offset.

The user performs two boresights: one from a few feet away for greater orientation sensitivity, and one less than a foot away (matching the two nails) for greater position sensitivity.

FOV measurement: It suffices to measure FOV along the vertical Y direction in screen space, since scaling that by the frame buffer's aspect ratio yields the horizontal FOV. The crosshair's Y radius is set to a quarter of the frame buffer's height to be easily visible. The user stands in front of the frame and lines up the top and the bottom virtual crosshair lines with corresponding real lines drawn on the frame's front surface. This forces the Eye space X-axis to be parallel to the frame's X-axis. From the information in steps one to three, the locations of the real lines in Eye space can be computed. With the now gathered data the FOV can now also be computed using simple trigonometry.

Results

Azuma reports that the accuracy achieved with this procedure using the optoelectronic tracker is about ± 5 mm from different viewing angles and positions. But he also states, that the registration accuracy depends on how

successfully the user can complete the registration procedures. Users reported difficulty in keeping their heads still during the boresight and FOV operations, because of the weight of the HMD. Testing the procedure with different users, which performed the whole procedure five times, the average standard deviation in computed orientation, position and FOV were 0.32 degrees, 4.8 mm, and 0.1 degrees respectively.

3.2.2. The “Shooting Gallery”

Overview and setup

[Oishi96] describes methods to compensate for mechanical misalignments in STHMDs and differences between actual and designed location of a user’s eye. The assumption is made that no distortions in the optical system exist making the calibration problem a linear one. Hence only the projection transformation has to be modified. The prototype STHMD they used has a horizontal FOV of 40 degrees and a designed location of the virtual image plane of 1m. The whole optical system of the HMD is mounted on a helmet and a plate is fixed on the front of the helmet as reference for physical measurements of the HMD parameters.

The Calibration Procedure

Mechanical misalignments

Mechanical misalignments result in a displacement of the virtual image plane from the designed location. Hence the position of the realized virtual plane has to be measured and the projection transformation modified accordingly. This procedure has to be executed only once for each HMD.

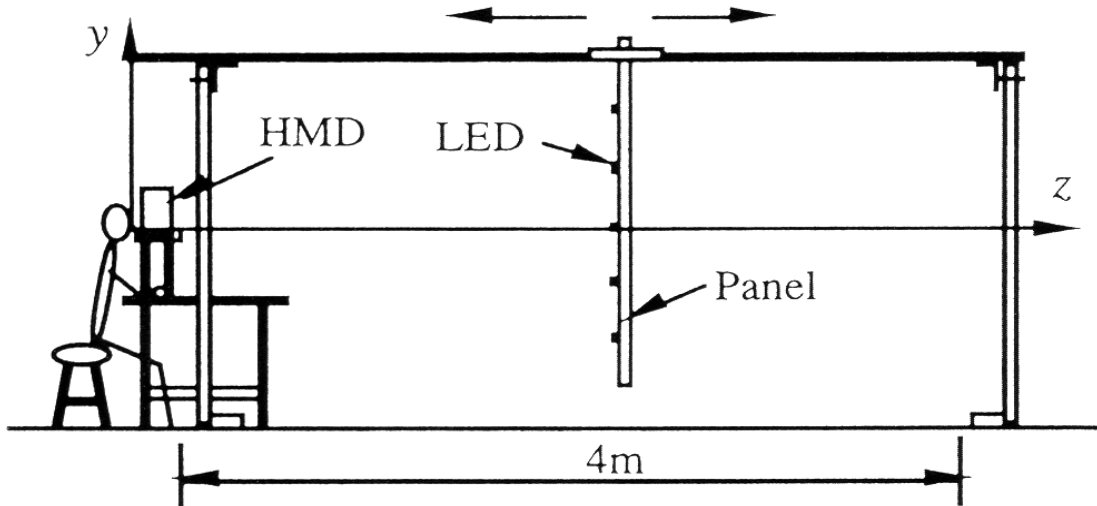


Figure 3 Measurement system for calibration [Oishi96]

The process of calibrating mechanical misalignments looks as follows:

1. Set HMD at the origin of the world coordinate system of the real environment;
2. Read the visual parameters and draw a cursor in the virtual environment;
3. Measure the distance to the realized virtual plane and put real environment marks at this distance;
4. Place a virtual cursor on the marks and record its location expressed in the virtual environment (13 marks per eye);
5. Compare recorded locations with original locations calculated according to the current projection transformation parameters; If the record is accurate enough, the process is finished, else continue with step 6.
6. Derive a measurement equation from the record and get modified visual parameters by solving it;
7. Redraw a cursor in the virtual environment by using modified visual parameters and continue with step 4.

The world coordinate system and marks for fitting are required in the real environment to measure the shift of the virtual plane and to perform the calibration. Figure 3 shows the measurement system used for the calibration. LEDs marking positions in the real environment are installed on the panel. The z-axis of the coordinate system runs through the panel's center and perpendicular to it. The panel held by two rails can move in the range between 0.5 and 4 m.

After the HMD was placed at the origin of the real environment, taking the plate mounted on the HMD as reference, it is firmly fixed at the calibration jig (step 1). In step 2 a cross-shaped cursor is drawn in the virtual environment, which is movable by a joystick.

The operator now swings her face up and down or left and right and observes the virtual cursor drawn on the center of the virtual plane, while moving the panel back and forth until the movement parallax between the virtual cursor and the LED on the center of the panel disappears. The panel distance at which this occurs is a close approximation of the realized virtual plane. Hence only parallax between the points on the virtual plane and the panel is now visible, i.e. the effects of the operators eye not being at the designed eye position is eliminated (step 3).

Now a human operator moves the virtual cursor so that it overlaps each LED and records its location (in virtual environment coordinates). This is done separately for both eyes, since the optical system of each eye is independent of the other (step 4).

If the data gathered in the previous step is not accurate enough (step 5), a measurement equation, which describes the relation between the result of the measurement and errors of the projection transformation parameters, is derived and used to estimate the actual visual parameters (step 6). These parameters are used to redraw the cursor (step 7) and the calibration process continues with step 4.

Differences between actual and designed location of user's eye

Since this calibration procedure should be done every time a user starts using the HMD, it is inconvenient to fix the HMD at a particular position. Hence the marks for the calibration are fixed on the plate mounted on the HMD. The calibration process looks as follows:

1. Place virtual cursor on marks on the plate P and record its location expressed in the virtual environment (5 marks per eye);
2. Derive straight lines from the recorded values and the marks on P.
3. Calculate the user's actual viewpoint as the point that minimizes the sum of distances to these straight lines.

Results

The RMS error of measured location of the virtual marker to actual position of the physical LED position was 1.5 mm for the left and 0.7 for the right eye. There was an error of 1 to 2 mm in the location of the drawn virtual cursor due to the low resolution of the HMD's LCDs. In addition the experiment system also contained 3 mm maximum error due to its hand-made fabrication.

For the eyepoint calibration an RMS error of 2.1 mm, 0.9 mm, 1.3 mm and 1.7 mm at a distance of the LED panel of 0.5 m, 1 m, 2 m and 4 m was measured, which is a reduction of the RMS error of about 40%-50% compared to the measurements taken without viewpoint calibration.

3.2.3. The Dynamic Calibration Process

Overview and setup

[McGarrity99] et al. also describe a method for calibrating STHMDs. They refer to a camera calibration method described in their previous work [Tuceryan95], which was based on using the relationship between the projected image positions of known 3D points and their 3D positions. This method is only applicable for a video-see-through display system where one can always access the image digitized by the video camera and use it to analyze the input images.

With a see-through system, the images of the scene are formed on the retina of the human user's eye and we do not have direct access to the image pixels. Therefore, a different approach is needed for the calibration of STHMDs. The approach described by [McGarrity99] uses a dynamic process in the forward direction (i.e., from 3D objects to 2D projected images) and let the user interactively adjust (estimate) the parameters of the imaging system until the projected image of the calibration model as seen by the human eye matches the image of the real calibration object in the scene. This is a truly *dynamic system* in the sense that while the user is interactively adjusting the camera parameters to align the displayed image, he is free to move his head. *That is, there is no requirement on the user to stay still or to keep his head in a static position.* The system updates the graphics reflecting the changes in the transformation, read by the camera marker. The parameters estimated by this calibration procedure are the intrinsic camera parameters and the camera-to-mark transformation, whereas the used camera model is the standard pinhole camera model.

The system setup consists of a pair of *i-glasses* head-mounted display. A 6-degrees-of-freedom (6-DOF) magnetic tracker provides continually updated values for the position and orientation of the tracked objects, which includes the *i-glasses* and a 3D mouse pointing device. The software is based on the Grasp system that was developed at ECRC for the purposes of writing AR applications.

The Calibration Procedure

The process of calibrating the camera using this approach consists of moving landmark points in the 2D image by grabbing and dragging them until they are aligned with their corresponding physical points in the image. During the dragging process, at every time interval, a set of dynamic equations (see [McGarrity99] for details) is solved for the camera parameters and the resulting projected image of the dragged model is displayed. This process continues until a sufficient number of points have been aligned with their physical counterparts so that the entire calibration object model is aligned with the physical calibration object. [McGarrity99] reports that the user interaction is very difficult when trying to solve for all the parameters at once. Therefore, they have broken the calibration procedure into a series of moded interactions in which the user can separately translate, rotate, and scale (by varying the focal length parameters) the virtual camera. These modes may be selected as the user desires, with the goal being to align the virtual object to the physical object.

The calibration of the stereo display system is a straightforward extension of this approach. The stereo system consists of a pair of cameras, which have a parallax due to their different poses (i.e., positions and orientations). To calibrate the stereo display system, [McGarrity99] uses the above calibration procedure to estimate the parameters for the left and right displays independently. This will account for the different rigid transformations for the poses of the two cameras that represent the two eyes as well as for the differences in the focal lengths for the left and right eyes. The final scene is displayed using the resulting camera parameters estimated using this process.

Results

For the evaluation of the calibration procedure, a video camera was placed inside the head of a mannequin where the eye would be located. The head was then attached to a camera tripod and the *i-glasses*/marker assembly was placed

onto it as if it were a real person. The graphics were sent to the HMD so that the camera received both the virtual and real objects simultaneously.

The resulting calibrations using this method are acceptable within the calibration volume, but the errors increase as the camera moves outside the calibration volume. [McGarrity99] also states that the quality of the calibrations seem to be better when done on a human head as they are intended, instead of the artificial setting they had for the purposes of collecting quantitative data, because the calibration done from a single viewpoint does not yield sufficient information to get all the scaling ambiguities correctly. When the calibration is performed dynamically on a human head, however, where the head is free to move and look at the calibration object from multiple views as the interactive calibration process proceeds, the calibration results improve considerably.

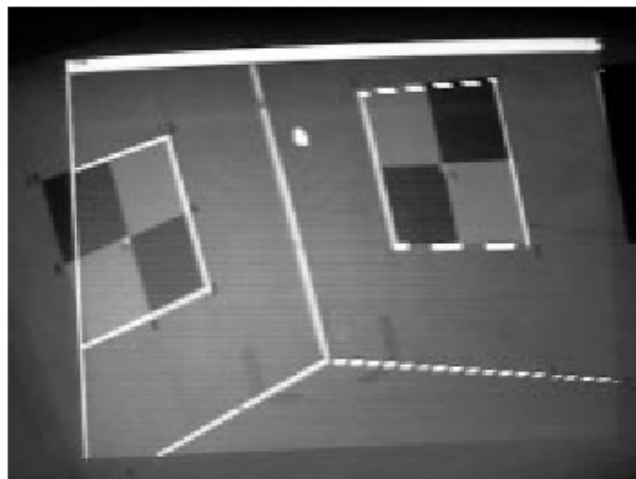


Figure 4 Alignment error image from the viewpoint in which calibration was done (all control points are visible). The alignment errors in this case are minimal. [McGarrity99]

[Tuceryan00] calls the user interface cumbersome, and further states that in addition, the number of parameters being estimated is too large, and therefore, the interaction does not provide a very intuitive feedback to the user. He proposes another method for the calibration of STHMDs, which is described in the following section (3.2.4.).

3.2.4. Single Point Active Alignment Method (SPAAM)

Overview and setup

[Tuceryan00] describes another method for the calibration of STHMDs, which was developed for the Grasp system, also used by [McGarrrity99] for the calibration method described in the previous section (3.2.3). Hence the hardware setup is basically the same. The used display device is an *i-glasses* head-mounted display. A 6-degrees-of-freedom (6-DOF) magnetic tracker provides continually updated values for the position and orientation of tracked objects and the software as mentioned above is based on the Grasp system that was developed at ECRC for the purposes of writing AR applications.

The Calibration Procedure

The goal for the method presented by [Tuceryan00] was to make the user interaction needed to collect the data for the calibration a streamlined process that does not impose a great burden on the user.

The calibration procedure has been implemented as follows:

The world coordinate system is fixed with respect to the tracker coordinate system by defining the world coordinate system on the tracker transmitter box (Figure 5 left). The tracker transmitter calibration is performed as described in [Tuceryan95]. This calibration is then stored and unless the decal put on the transmitter box is replaced or is somehow moved, there is no need to redo this calibration again. Fixing the world coordinate system with respect to the transmitter box has the added advantage that the tracker can be moved at will to any position and the calibration still stays valid. The world coordinate system could also have been assumed to correspond to the tracker coordinate system by definition, however, this would have been harder to use because we do not know exactly where the tracker coordinate system is on the transmitter box. Therefore, it seems to be better to define the world coordinate system whose location is known and estimate its relation to the unknown tracker coordinate system by a calibration procedure.

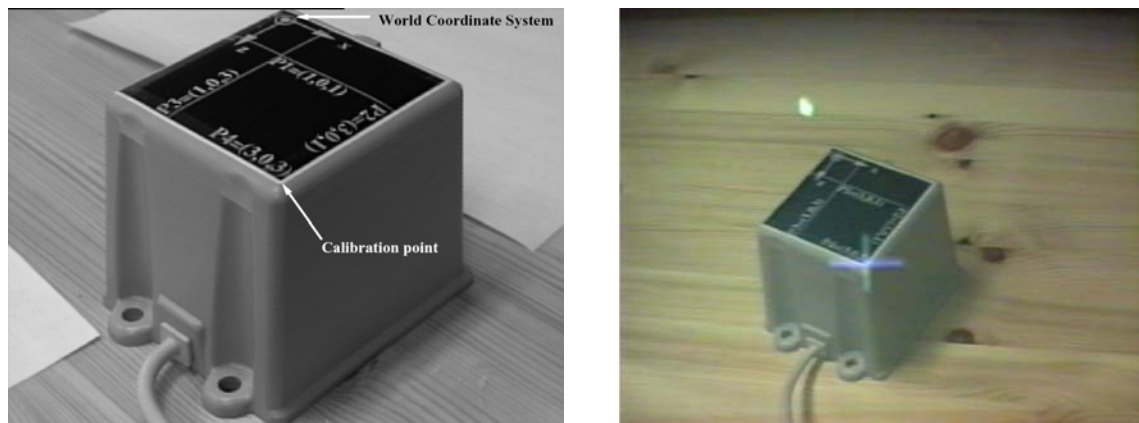


Figure 5

The world coordinate system is fixed on the tracker transmitter box (left)
The user aligns a cursor with a fixed point in the world (right) [Tuceryan00]

A *single* point in the world coordinate system is used to collect the calibration data. This single point in the world coordinate system is mapped to many distinct points in the marker coordinate system as the user's head is moved about, i.e. the points are transformed to the head marker coordinate system.

The user is presented with cross-hairs on the display and is asked to move about his head until the crosshair is aligned with the image of the single calibration point as seen by the user (

Figure 5 right). The user then clicks a button on the 3D mouse and the data is collected for calibration that consists of the image coordinates of the cross-hair and the 3D coordinates of the calibration point in marker coordinates. These collected points are then fed into the camera equation (see [Tuceryan00] for details), which is then used to estimate the camera parameters. There are 12 parameters of the 3×4 projection matrix, which have to be estimated by the calibration algorithm. But the projection matrix is defined up to a scale factor, therefore actually only 11 parameters have to be estimated. Since each calibration point gives us two equations, at least 6 points are needed for the calibration. However, in order to account for the errors and obtain a more robust result, [Tuceryan00] proposes to collect 12 points and use a least squares estimation. He states that the more of the tracker volume the user's head covers, the more of possible systematic errors in the tracker measurements will be taken into account in the optimization process. Hence the user is encouraged to move his head around the tracker transmitter as much as possible while collecting the calibration data.

Results

The user's collection of the necessary data to calibrate the display is a very quick and easy process. During this process, the user is not required to have his head fixed and is allowed to move. [TuceryanOO] reports that this calibration method was evaluated in numerous trials and in all instances the calibration results are very good. The quality of the calibration results does not change greatly as the head moves around in the world. The only problem is due to the lag in the readings from the magnetic tracker, which tends to settle down to the correct position after a certain delay after the head stops moving. Some of the factors that affect the calibration include the distance of the user's head from the tracker transmitter and how quickly the user clicks the mouse to collect the calibration data. The magnetic tracker they use has a range of about 3 feet and the quality of the sensor readings is not very reliable when the receivers operate near the boundaries of this range. The second factor that affects the calibration is the lag in the tracker data at the point of collection (i.e., when the mouse is clicked). If the button is clicked too quickly, the tracker data read might not correspond to where the user's head is. [TuceryanOO] states that if the user is careful during the calibration, both of these factors can be put under control and the calibration results are good.

3.2.5. Projection Device Calibration

Overview and setup

[Summers99] et. al describe calibration procedures for the Virtual Hand Lab (VHL). The VHL is a desktop augmented reality environment for conducting experiments in human perception and motor performance that involve grasping manipulation, and other 3D tasks that people perform with their hands.

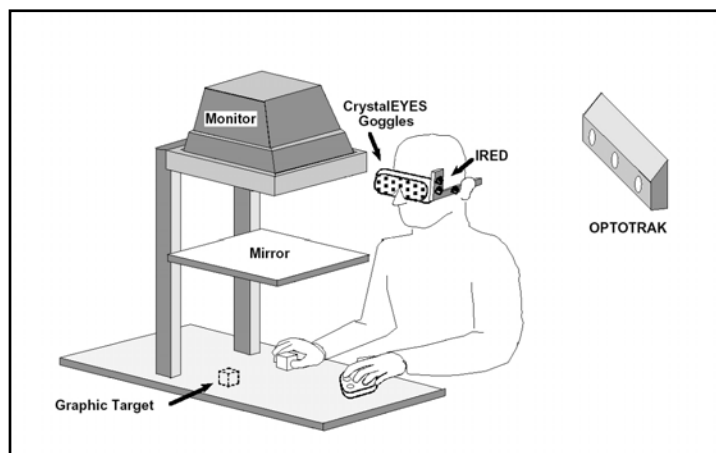


Figure 6 Hardware setup [Summers99]

The hardware setup (Figure 6) consists of a workstation, a slaved secondary *monitor* that is reflected through a mirror, a Northern Digital Optotrak 3D motion analysis system (*tracker*), StereoGraphics CrystalEyes stereographic glasses (*stereo glasses*). The workstation displays stereo images on its regular screen (which is seen by the experimenter) and also on the slaved monitor. The mirror places these images appropriately within the subject's view. The Optotrak senses the 3-D positions of infrared emitting diodes (*markers*) that are strobed under computer control. The markers are placed on all objects (including the subject) whose position or orientation is required during an experiment. The stereo glasses, used in conjunction with position and orientation information for a subject's head obtained (from the Optotrak), enable the monitor to display head-coupled stereo images for the subject. Subjects look through the half-silvered mirror and can reach underneath it, resulting in the virtual image reflected through the mirror appearing to be on top of the actual scene beneath the mirror. The workspace is defined as the volume between the mirror and the desktop into which a subject can physically reach.

The Calibration Procedure

The calibration process falls into two parts. The workspace calibration (volume of physical space which can be augmented), which only needs be performed once for a new location of the workspace, i.e. if the tracker or the monitor/mirror setup moved. The point of view calibration (for a head-tracked stereo display) is done on a per user basis.

Step 1: The workspace calibration

The calibration procedure consists of aligning a set of virtual crosses with a set of markers according to a pre-defined one-to-one correspondence. The markers have identity because they are strobed, and the crosses are described relative to the workspace orientation chosen by the experimenter. For example, the first marker is placed in the “left front corner”, where “left” and “front” are arbitrarily chosen by the experimenter. For a given virtual cross, there is exactly one location in 3-space in which a marker can be placed so that the alignment of the virtual cross and the marker are not a function of head position. In all other positions, moving one’s head will cause the virtual cross to “swim” with respect to the physical marker. For this task of proper alignment, the procedure relies on human perception, but only to detect 2D misalignment, not differences in depth. Alignment in the third dimension is determined by the absence of swim, which is something that humans are very good at detecting. Because 2D alignment of the markers with the virtual crosses and the check for lack of swim do not vary with the head position, this calibration can be performed without stereo, i.e. independent of the point of view calibration.

Using standard techniques, a coordinate system is obtained from a set of four points, which according to [Summers99] is a good tradeoff between gathering additional redundant data for further reduction of the variance and a short execution time of the calibration procedure.

Step 2: Point of View calibration

Instead of tracking the eyes directly, the eye position relative to the glasses is estimated, and then the glasses are tracked while the subject interacts with the environment.

Outline of the calibration process:

1. The experimenter attaches rigid plastic plates to the sides of the stereo glasses and then places three markers on the plate facing the tracker in any non-collinear positions. These markers are used to track the head position and form a head coordinate system.
2. The subject dons the stereo glasses, which are secured against subsequent slippage.
3. The subject closes her left eye. A bar with a small hole is provided. The bar is aligned over the subject's right eye so that a point in the center of the workspace is clearly visible. The bar is then attached to the glasses (see Figure 7).
4. The subject repeats this for the other eye.
5. The subject then looks through both holes simultaneously. The holes should align so that she sees the same fused image through both holes. It should appear as though there is only a single hole, not two separate or overlapping holes. If necessary, the subject rearranges the bars until this is the case.
6. The experimenter places markers directly over the holes.
7. Two seconds of data (120 frames) are captured, processed and analyzed. A summary of the analysis is produced. The experimenter will rerun the data collection if too many of the data frames are erroneous. This is typically solved by changing the subject's head position.
8. The bars and markers are removed from the subject's eyes. The markers, which track head position, remain.

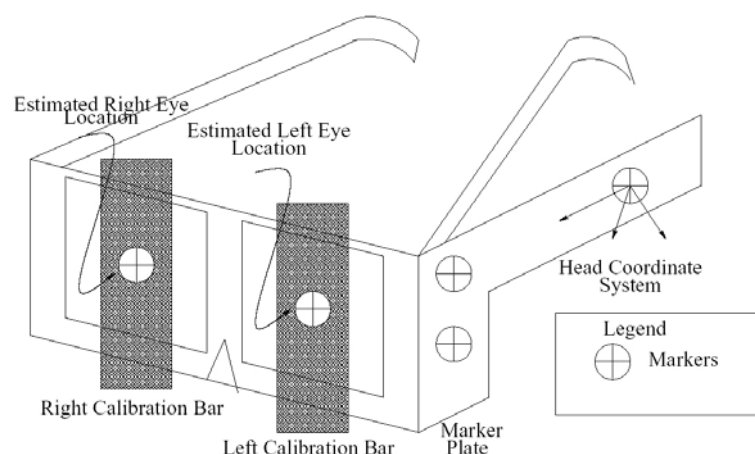


Figure 7 Shutter glasses for point of view calibration [Summers99]

Using the collected data, rigid body transformations between the position of the eye markers and the head coordinate system are calculated and used to

estimate the eye positions after the calibration markers are removed from the eyes. These estimates are actually 1- 2 cm in front of the real eye points, which results in a slight magnification of the scene.

Results

To evaluate the consistency of the workspace calibration, the workspace was calibrated three times. For each calibration, data was collected four times without moving any markers, for a total of 12 readings. Each marker produces an (x,y,z) triplet, in tracker coordinates. Comparing readings within the same calibration allows evaluating the tracker error. The maximum range (maximum value minus minimum value for a given marker/axis combination) was 0.24 mm. The maximum range over all readings and calibrations was 1.49 mm.

The viewpoint calibration procedure does not give the exact point of view. The markers used for estimating the eyes are placed on the glasses, not in the eyeballs. To evaluate the error, an optician's pupillometer was used to measure interpupillary distance (IPD) for three subjects for each of seven focal distances: 35, 40, 50, 65, 100, 200 and ∞ (cm). The calibration IPDs all fell within range of the pupillometer readings.

To quantify the effect of the point of view (POV) calibration, wireframe blocks of various sizes were displayed at various locations using a randomized trial script. The width (Y-axis) of the virtual block was measured with a physical ruler. The head moved freely to obtain the best perspective. Over 15 trials, the mean error was 0.47 mm with a maximum error of 1 mm. This experiment was repeated for the other dimensions. For height (Z) the mean error was 1.73 and the maximum error was 4 mm, and for depth (X) the mean error was 1.33 mm with a maximum error of 3 mm.

3.3. Object Calibration

3.3.1. Calibration with Reference Frame

Calibration Procedure

[Summers99] also described a method for the calibration of tracked props for the VHL (see section 3.2.1 for details of the hardware setup). Calibrating an object involves matching a computer model to the physical object. Both the model and the object are assumed to have distinguishable dimensions. The calibration technique proposed by [Summers99] is only dependent on the experimenter's ability to manipulate physical objects, not on an ability to manipulate physical and virtual objects together. This technique works for any non-deformable object.

The procedure for this calibration is as follows:

1. Attach three markers to the object in any non-collinear positions.
2. Place three markers in predefined locations of the calibration frame. The frame may be placed anywhere within view of the tracker. Its placement is completely unrelated to the location of the workspace except that it must be within range of the tracker.
3. Place the object in the corner of the calibration frame so that the XYZ orientations of the object and frame match.
4. Collect marker locations for both frame and object in tracker coordinates.

The position of the object markers relative to the frame markers plus knowledge of the size and shape of the physical object allow the computation of the internal coordinate system of the physical object relative to the object markers.

Results

Sources of error for this procedure are tracker error, incorrect measurement of the physical object and incorrect position of the object in the calibration frame. Tracker error is within 0.3 mm, physical measurements with a ruler are accurate within 1 mm, and position errors are negligible because both the

objects and the frame are rigid and they fit tightly together. Recalibrating for new marker locations takes 30-60 seconds.

3.3.2. Calibration with Pointing Device

Overview and setup

[Whitaker95] describes methods for the calibration of a pointing device, which is then used for the calibration of objects. The Grasp system, which was developed at ECRC as a platform for research in augmented reality, is used to test the proposed procedures with a monitor-based setup (Figure 8). The graphical image is generated by the workstation hardware and displayed on the workstation's high-resolution monitor. A scan converter takes the relevant portion of the graphical image and converts it to a standard video resolution and format. The scan converter also mixes this generated video signal with the video input from the camera. A tracker, which is capable of sensing the three translational and the three rotational degrees of freedom, provides the workstation with continually updated values for the position and orientation of the tracked objects including the video camera, a pointing device, and other objects of interest. A frame grabber is used to grab digital video images for processing within the system.

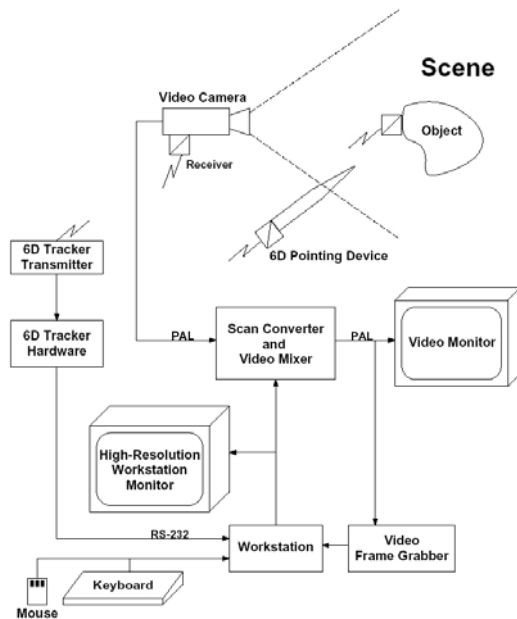


Figure 8 Hardware configuration of the Grasp monitor-based setup [Whitaker95]

The Calibration Procedure

1) Pointer Calibration

The pointer calibration calculates the geometry of the pointing device used within an application. In the system described by [Whitaker95], this pointer object is a wooden wand with a tracking mark (receiver) attached at its base, which is used to locate 3D points on real objects during user interaction. In particular, this calibration step calculates the position of the tip of the pointer relative to the tracker mark. This calibration is a prerequisite step for the object calibration.

The geometry of the pointer object is not pre-defined but calculated during the calibration procedure. The mechanism to calibrate the pointer requires the user to pick the same point in 3D space several times, using a different orientation for the pointer each time. For each pick, the position and the orientation of the tracker mark within the tracker coordinate system are recorded. The result of this procedure is a set of points and directions with the common property that the points are all at the same distance from the single, picked point in 3D space and all of the directions associated with the points are oriented toward the picked point. This geometrical constraint is used to formulate an equation, which gives an estimate of the desired offset from the wand's marker position to the wand's tip when solved, in this case using a least squares method.

2) Object Calibration

Object calibration is the process whereby the location and orientation of a real-world object is calculated such that a virtual counterpart can be placed at the corresponding location, with the appropriate orientation, within the virtual world. Some real-world objects will subsequently have their movements tracked by their virtual "shadows", in which case the corresponding tracker marks must also be calibrated. We first need a computer model of the object and then a calibration procedure to locate the real object so that the virtual model can be registered to it.

The calibration procedures described here require a number of landmark points whose positions are known in the coordinate system of the object model. Geometric models of objects might be created piece-by-piece from a set of geometric primitives or they might come from a CAD system. Regardless of their origin, models are generally stored as files on disk for subsequent use. Therefore files describing real objects must contain, in addition to the

geometric and attribute data, the 3D positions and labels of the landmark points. These points should correspond to features on the object, such as corners or creases that can be easily identified by a user. The registration procedure consists of locating the corresponding points on the real object and the model, and then calculating the object-to-world transformation from these point pairs.

For this method a pointing device is needed, which is capable of generating the world coordinates of positions in the real world. In this case, the pointing device (which itself must already be calibrated) is a magnetic tracker attached to a wooden cone which, when calibrated, has an accuracy of about $\pm 1\text{cm}$. Over half of this error is bias and transfers directly into object calibration error.

The problem here is to compute the rigid transformation between a set of 3D point pairs. Using the 3D pointer and several keystrokes the user indicates the world coordinates (or some other 3D coordinate system) of landmark points on the object, which are also given in the object's local coordinate system. The relationship between these sets of points gives rise to a linear system of 12 unknowns (see [Whitaker95] for mathematical detail). For a unique solution 4 points are needed, but in most cases more than 4 points are used and the equation is solved for the least-squares error.

Results

The pointer-based procedure provides the object-to-world transformation that is needed for object registration within a small number of seconds. Figure 9 shows a model engine, which has been calibrated in such a manner. Rotations of the engine (Figure 9 right) show that this calibration does not suffer from the depth problem of the *image-based* approach.

This *image-based* approach described by [Whitaker95], is based on a calibrated camera, which is used to compute the object-to-camera transformation of a single object for which there is a known geometric model. The position of an object is determined “through the lens” of the camera. The calibration begins by capturing an image of the real-world object and locating a set of landmark points in the image. The locations of landmark points in the image are found manually by a user with a mouse. With the assumption that the points are mapped from known locations in 3-space to the image via a rigid 3D transformation and a projection, and that the camera is calibrated, the pose of the object in camera coordinates can be calculated.

Despite good point wise alignment in the image plane, the image-based calibration can produce significant error in z -direction (distance from the camera), which is not seen in the re-projected solutions. For instance, in the case of the engine model, the image-based approach can produce a rigid transformation, which matches landmark points in the image to within about 2 pixels. Yet the error in the z -direction can be as much as 2-3 centimeters. This error becomes evident as the object is turned.



Figure 9 A wireframe engine model registered to a real model engine using pointer-based calibration. [Whitaker95]

3.4. Summary and Conclusion

In many cases the methods discussed above require additional custom made hardware, which is an impractical approach for many AR setups, e.g. when it has to be moved to an exhibition, or difficult to reproduce. This additional hardware also introduces new calibration problems and new sources of error. Nearly all calibration procedures are somewhat tailored to the actual hardware setup that was used for the evaluation of the particular procedure and sometimes are only applicable to that particular setup.

With the exception of the procedures developed for the Grasp system, only single calibration tasks are described without showing the intention or presenting a concept for the integration of this task in a set or context of calibration tasks, which are needed for supporting a variety of different, often changing hardware setups, typical for AR installations and especially for the Studierstube AR system.

The calibration tasks, which had to be performed by the user, i.e. the user interactions needed for the particular procedure, are sometimes cumbersome and time consuming and often lack any conclusive feedback or support for the non-expert user.

So we try to avoid these pitfalls and take useful and practical solutions for the calibration problems described in this section — e.g. the use of a pointing device, which delivers accurate registration and is especially practicable in our case, since the pointing device is an integral part of a typical Studierstube system setup — as inspiration for developing and implementing our calibration procedures, which should be comprehensive, accurate and user friendly.

The innovative aspect of our method lies in its unification of separate but connected problems: calibrating the HMD or shutter glasses or projection screen for example, works only when the stylus' hotspot has been registered, but is independent of the world-to-tracker transformation. The sequential execution of the different steps of our method takes these interdependencies into account.

A further advantage of our method lies in its simplicity as perceived by the user: every sub-task consists of an easily performed gesture, or the touching of a displayed marker. Additionally all of the used algorithms have fail-safe mechanisms: if the user wrongly positions his head during display calibration, it is detected via the head-tracker and the step is repeated; if the fitting algorithm when registering a prop or the stylus shows discrepancies, a notification message is displayed.

Chapter 4

Calibration procedures

This section gives a detailed description of the principles used for each step of the calibration process. Implementation details regarding the utilities used to realize this work, like the Studierstube system and the OpenTracker framework will be presented in chapter 5. The results of the calibration tasks will be discussed in chapter 6.

4.1. Stylus Calibration

The first step in the whole calibration procedure is the stylus calibration. It determines the offset of the tracker sensor as measured to its tip or hotspot (h_{Ts}). This has to be done at the start of the whole process, if the tracking system does not provide an implicitly registered stylus (e.g. the origin of a tracking target of the DTrack system lies in the center of one of the markers, hence this marker may be used as hotspot), because most of the following tasks in the process need to sample the position of the stylus' hotspot in world coordinates, hence the offset has to be obtained beforehand.

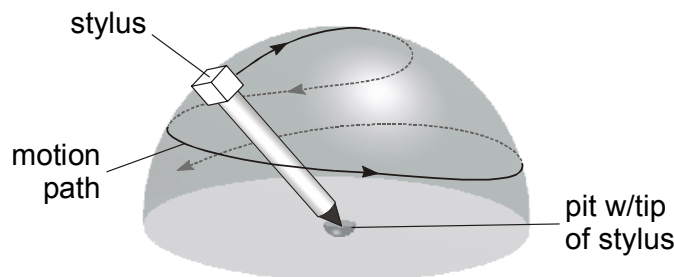


Figure 10 Calibrating the stylus

We can accomplish this quite easily by fixing the stylus' tip in a small pit drilled into a table (Figure 10), and moving the tracker sensor on a hemisphere

measuring its position and orientation. Having fitted a sphere to the measured sensor positions, the center of the sphere gives us the position of the hotspot with respect to the sensor.

For fitting an optimal sphere to the measured data points, we optimize the offset vector from tracker sensor to the stylus' hotspot (h_{Ts}). The principle of the optimization algorithm used for this purpose is called 'Direction Set (Powell's) Methods in Multidimensions [Press88] and will be discussed in section 5.4. The function we want to minimize utilizing Powell's algorithm, uses the variance V of the hotspot positions for the measured data points as a metric. The hotspot position in world coordinates is calculated for each sample as follows:

$$h_w = h_{Ts} T_{TsW}$$

h_w is the position of the stylus' hotspot in world-coordinates
 h_{Ts} is the position of the stylus' hotspot in tracker-sensor-coordinates
 (the offset from tracker sensor to hotspot)
 T_{TsW} is the transformation matrix, which transforms a point given in tracker-sensor-coordinates to world-coordinates (given by sampled position and orientation of the tracker sensor attached to the stylus)

The variance V of the calculated hotspot positions is then given by:

$$V = \frac{1}{n-1} \sum_{i=1}^n |h_i - h_{mean}|^2$$

n is the number of samples
 h_i is the hotspot in world coordinates
 h_{mean} is the mean of all calculated hotspots in world coordinates

The optimization of the variance gives us a least squares fit solution for the hotspot offset h_{Ts} .

To enforce stability of the solution, the user should cover a large part of the hemisphere, ideally by sweeping the sensor along two orthogonal great-arcs. To enforce this, each valid sample, i.e. a sample that will finally be used in the optimization step, has to meet a minimal distance criterion. So each sample fed into the program during the sampling stage is tested against all other valid samples obtained so far. If the distance between the sampled position and one of the valid samples is smaller than a specified minimal distance, it is discarded otherwise it is stored as another valid sample. Hence holding the stylus

stationary will only store one sample, forcing the user to move the stylus around. The progress of the stylus calibration process is displayed on the screen, where a counter tells the user how many samples have to be gathered and how many already have been retrieved.

The actual user interaction needed for this calibration task is quite simple:

The user has to put the stylus' hotspot into the prepared drilled in pit or fix the hotspot in another appropriate way and press the button on the stylus to start the sampling. Then she has to move the stylus around its tip until the specified number of samples or more are acquired. Finally she has to press the pen-button again to stop the sampling and start the calculation of the desired hotspot offset h_{Ts} . Each action the user has to perform is described in text form on the display device of choice (e.g. HMD, monitor etc.) to guide the user through the process.

To incorporate the resulting offset in the Studierstube system, the program has to change the used tracker parameters. Since Studierstube uses the OpenTracker framework [Reitmayro0] as interface to all tracker data, the offset has to be included in the OpenTracker tree, which describes the actual tracker configuration at runtime (see section 5.3 for details). This is done, by creating a new virtual offset node, which represents the calculated hotspot offset, and inserting it as first child of the sub-tree associated with the tracker-station of the stylus, i.e. the StbSink node representing the input data from the stylus' sensor (Figure 11). Furthermore the new configuration of the whole tracker framework is stored in a configuration file, so that it can be easily loaded the next time Studierstube is started. Hence once the stylus was registered the stylus calibration only has to be performed, when a new stylus shall be used or the tracker setup of the old one changes.

```
<StbSink event="off" station="1">
  <EventVirtualPositionTransform translation="-0.063 -0.017 -0.009">
    <NetworkSource    number="1"
                      multicast-address="224.100.200.101"
                      port="12346"/>
  </EventVirtualPositionTransform>
</StbSink>
```

Figure 11: The snippet from the XML configuration file, produced during this calibration step, shows the inserted virtual offset (bold).

So after the stylus calibration has been completed all further position data sampled from the stylus' tracker station will implicitly specify the position of the stylus hotspot in the tracker coordinate system.

4.2. Display Device Calibration

To achieve image registration, meaning alignment of real and virtual world when projected on the retina of the user, we need a precise description of the projection from the real world onto the retina. This step, the display device calibration, has to determine (the intrinsic) and extrinsic parameters of the virtual camera which has to mimic the projection of the real environment. The first chapter of this section will deal with the camera model used by the Studierstube system, to provide an insight into the parameters that need to be obtained by this calibration step. As already mentioned, the *Studierstube* system permits the use of see-through head-mounted displays (HMDs) and different kinds of front or back projection displays. Although both setups are to some extent similar, we shall address the calibration processes for projection and HMD setup separately in the second and third chapter of this section.

4.2.1. The Studierstube offaxis camera model

Since the Studierstube system is based on the Open Inventor toolkit [Strauss92], the viewing parameters for a scene are specified within a camera node. Open Inventor supports only orthographic and (on-axis) perspective pinhole cameras, so in order to be more flexible in regard to defining viewing parameters, especially when concerning projection setups, a new camera model was introduced to the Studierstube System implemented as extension (SoOffAxisCamera node) to the Open Inventor toolkit.

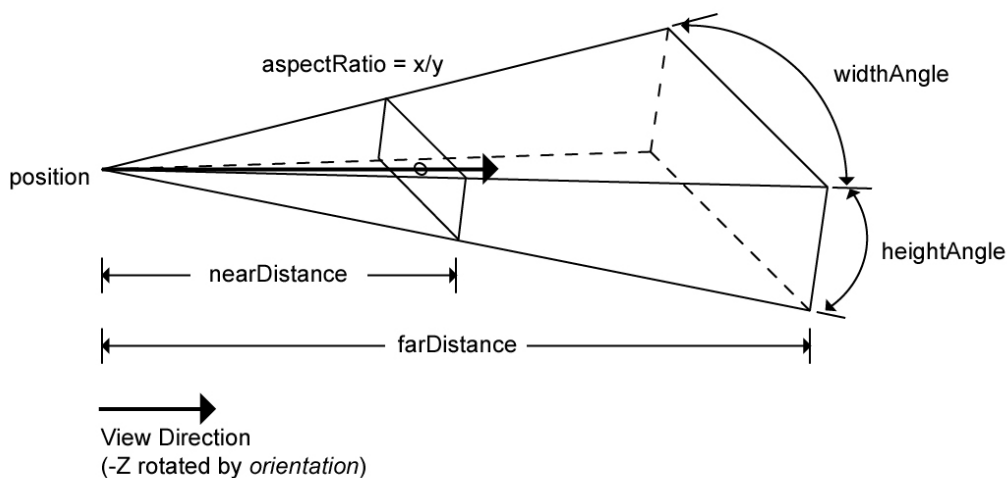


Figure 12 View Volume and Viewing Projection for a SoPerspectiveCamera node [Wernecke94]

Whereas the common perspective camera model describes the viewing volume relative to the camera's eyepoint (Figure 12), the basic principle of the SoOffaxisCamera (Figure 13) is the decomposition of the camera model into two mutually independent logical parts:

- the eyepoint (or viewpoint) and
- the projection plane (or projection area).

The eyepoint describes the position of the viewing pyramid's apex and can be placed arbitrarily in 3D-space. The parameters for the projection plane specify a rectangular area, whose corners represent the intersection points of the projection plane with the viewing pyramid's edges. Like the viewpoint, the projection area can be placed arbitrarily in space, hence it is placed independently from the viewpoint. The projection area is specified by three parameters:

- position
- orientation and
- size

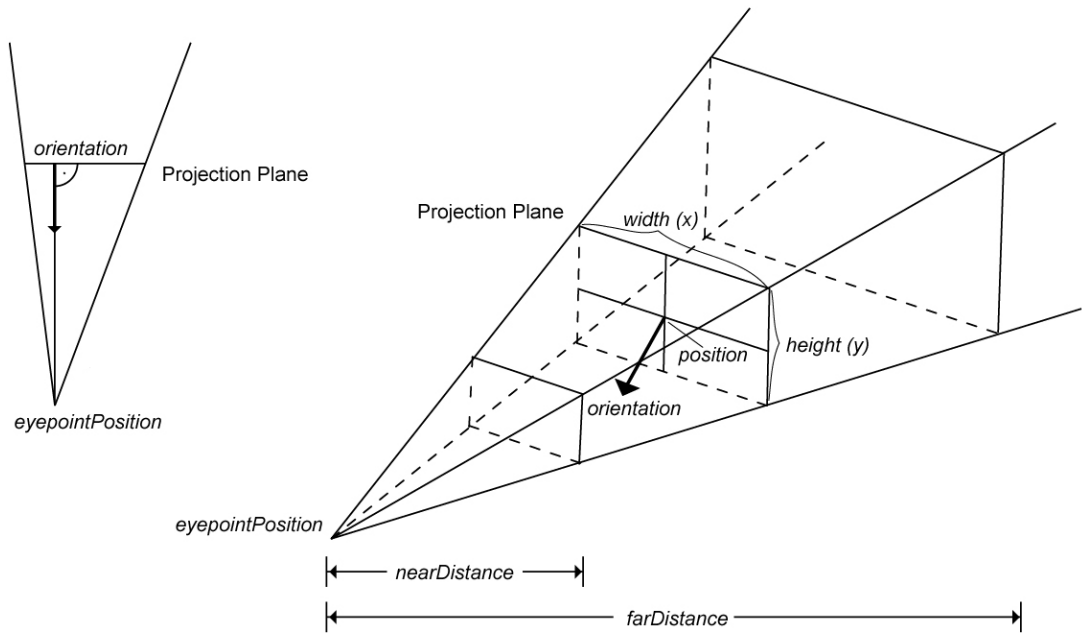


Figure 13 View Volume and Viewing Projection for a SoOffAxisCamera node

The 3D-vector *position* specifies the location of the center of the area. The rotation given by *orientation* (usually defined as quaternion due to Open Inventor conventions) specifies the orientation of the projection plane, defined as rotation from its default orientation. At the default orientation, the positive

z-vector is equivalent to the plane's normal and the positive y-vector is equivalent to the plane's up-vector. The *size* given by a 2D-vector specifies the size (width, height) of the area. Hence it also specifies the aspect ratio of the camera. The (horizontal) field of view (FOV), which is often used in a camera model, is implicitly specified by eyepoint and projection plane properties of the Studierstube offaxis camera model.

As mentioned above the eyepoint and the projection plane can be positioned independently from each other and define (in conjunction with the near and far plane) a viewing frustum. If the eyepoint lies within the positive half-space of the projection plane, the camera will render the scene, otherwise the camera is not valid, because the viewpoint lies "behind" the projection plane, hence the scene will not be rendered. For the special case, where the eyepoint lies on the normal through the center of the area, the camera will render like a usual perspective pinhole camera.

To achieve stereo perception with the Studierstube system two virtual cameras must be specified separately. Hence the distance between the eyepoint-position of the left and right eye camera implicitly gives the inter-pupillary distance.

4.2.2. Calibrating See-Through Head-Mounted Displays

As stated before [Azuma94][Bajura95] HMD-calibration requires in most cases direct interaction of the user. In the stated cases this interaction required a principal understanding of what different calibration steps were supposed to be achieved (e.g. calibration of field-of-view) and complex interactions with the system (see section 3). We want to reduce user interaction to a guided approach, which in a few simple steps allows the user to calibrate the HMD without needing special training or understanding. This allows for a setting where a high throughput of different users is to be expected, e.g. a scientific exhibition or a museum. To achieve correct registration for a specific user we have to calibrate the HMD while it is being worn by this user, because even slight differences in eye-distance and distance between eye and optical system of the HMD lead to invalid registration. Even when a user has calibrated the HMD before and puts on the HMD again, the previously achieved 'personal' registration is very likely to be compromised, due to the fact that every time the user puts on the HMD, the relative position between the user's eyes and the optical system of the HMD will be different. As an example for a similar calibration procedure the joystick calibration procedure used in many computer games may serve: "Move the joystick to upper left corner, then press

button. ...". To put it another way: we want to maintain interactivity while reducing user effort.

Previous approaches [Azuma94][Oishi96] make use of additional hardware for their calibration procedures. They deliver high-quality registration results at the expense of a complicated setup and considerable user effort. We want to avoid the use of additional hardware as far as possible, since it adds further registration problems (e.g. calibration of the registration hardware itself) and can considerably reduce the mobility of the whole system.

Since the precision of the calibration depends on the users interaction, we have to find a method, which presents us with a stable solution. This means that errors in some of the input data points should still produce a viable solution and not render the resulting registration completely unusable. This instability may happen when a projection matrix is optimized without regard to its inherent redundancies. The requirement of reduced user effort implies an upper limit of the amount of input data, which could further increase instability.

Overview of the calibration process

The focus of this calibration step [Fuhrmann00] is the retrieval of the correct viewing parameters, i.e. the correct virtual camera, which ideally models the user's view through the HMD. Since the user's eye generally does not lie centered over the projection plane not only the determination of the viewing direction but also of the orientation of the image plane is necessary, hence we use the Studierstube off-axis camera model (see section 4.3.1), which takes into account the physically decoupled nature of eye-point and image plane inherent in a see-through HMD setup.

The following parameters have to be calibrated:

- Position of eyepoint
- Position of (middle of) image plane
- Orientation of image plane
- Aspect ratio (size of image plane)

We have implemented a two-step optimization procedure, which optimizes these camera parameters for a given set of data quintuple. Each quintuple contains the 3D coordinates of one sample point and its 2D projection. The full calibration process consists of the following steps:

1) Acquisition of calibration data

The user samples the positions of virtual markers with a 6DOF input device in an interactive process.

2) Geometric determination of camera parameters

Using inherent geometric properties of the acquired data, a first approximation is determined.

3) Numerical optimization

A further optimization step calculates a solution for off-axis projection.

User interaction is normally only necessary in step 1, but exceeded error tolerances in one of the further steps may prompt the user for reentry of some data samples.

Acquisition of Calibration Data

The properties of see-through HMDs make their calibration significantly different from the calibration of video-based HMDs. Video-based HMDs are essentially immersive HMDs with attached cameras. The cameras supply the video streams which – after being overlaid with the computer generated images – are fed into the HMDs. Calibration of video-based augmentation [Bajura95] only determines the parameters of the video camera. Differences between the cameras' parameters (FOV, inter-pupillary distance, etc.) and the user's eyes are not taken into account, since the alignment of real and virtual images is inherently guaranteed by taking the 'real' images from the camera's video stream. The discrepancies in the complete system only result in the same effects as in an immersive VE, as discussed in section 2 ('The calibration problem').

The advantage of a video-based Augmented Environment is that the video images of the real environment can be used to directly gather calibration data. We can present calibration patterns to the HMD, i.e. the video cameras, and extract the coordinates of the projected data points from the captured image, using image-processing techniques. The only place where the complete augmented image is visible when using a see-through HMD is at the retina of the user. While one may use a video camera in the position of the user's eye, the resulting registration will only be valid for the position of this camera. The data gathering stage of our calibration scheme, in which we have to acquire a quintuple of 3D coordinates of a point and its 2D projection therefore has to rely on the user to identify, whether a real point in space and its virtual projection match. To achieve this, we reverse the image processing approach –

presenting a real calibration pattern and identifying points of its projection – and present the user with a virtual calibration pattern, with which real points have to be aligned.

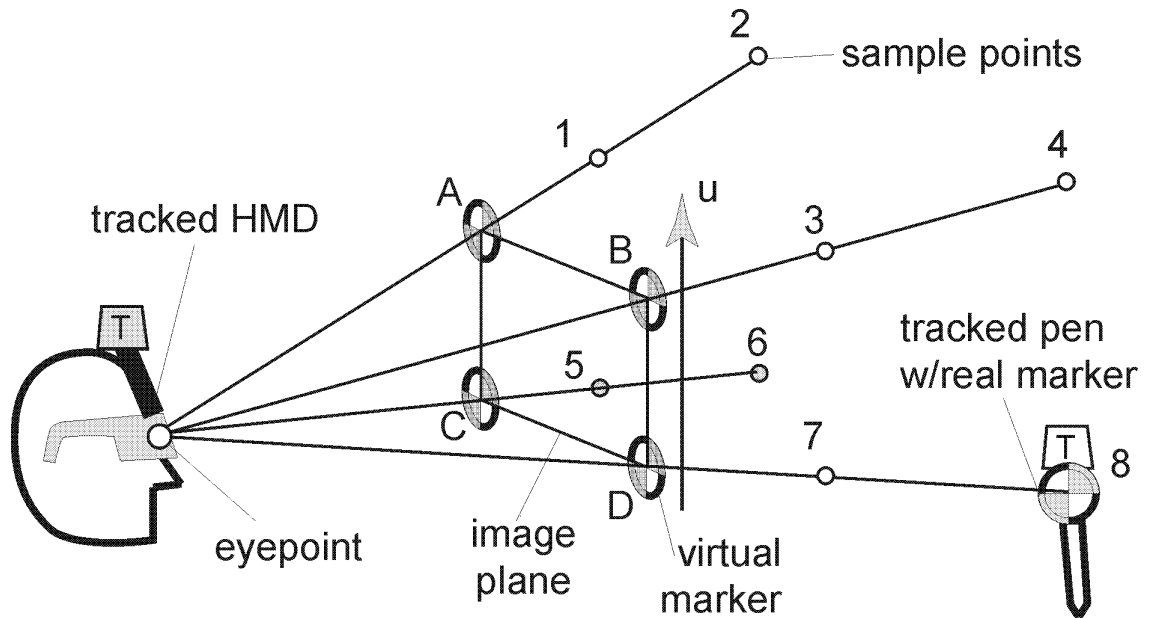


Figure 14 Calibration setup for see-through HMD calibration

The user sees a real marker on the (already calibrated) tip of the tracked stylus (Figure 14), which she has to align with a virtual marker displayed via the HMD on the virtual image plane (virtual markers A-D). When the alignment is reached (Figure 16) the user presses a button on the pen and the next virtual marker is displayed. At the press of the button, tracking data of the sensors attached to the pen *and* the HMD is sampled. The position of the pen's tip is transformed into the coordinate system of the HMD tracker sensor, which eliminates influences of different head positions during the calibration process. Hence the fact that the position of the stylus' tip is sampled relative to the head-tracker allows the user to freely move her head while performing the calibration task. The resulting 3D point gives us - together with the known 2D location of the virtual marker - one quintuple of calibration data. Figure 15 shows the calibration process for one eye.

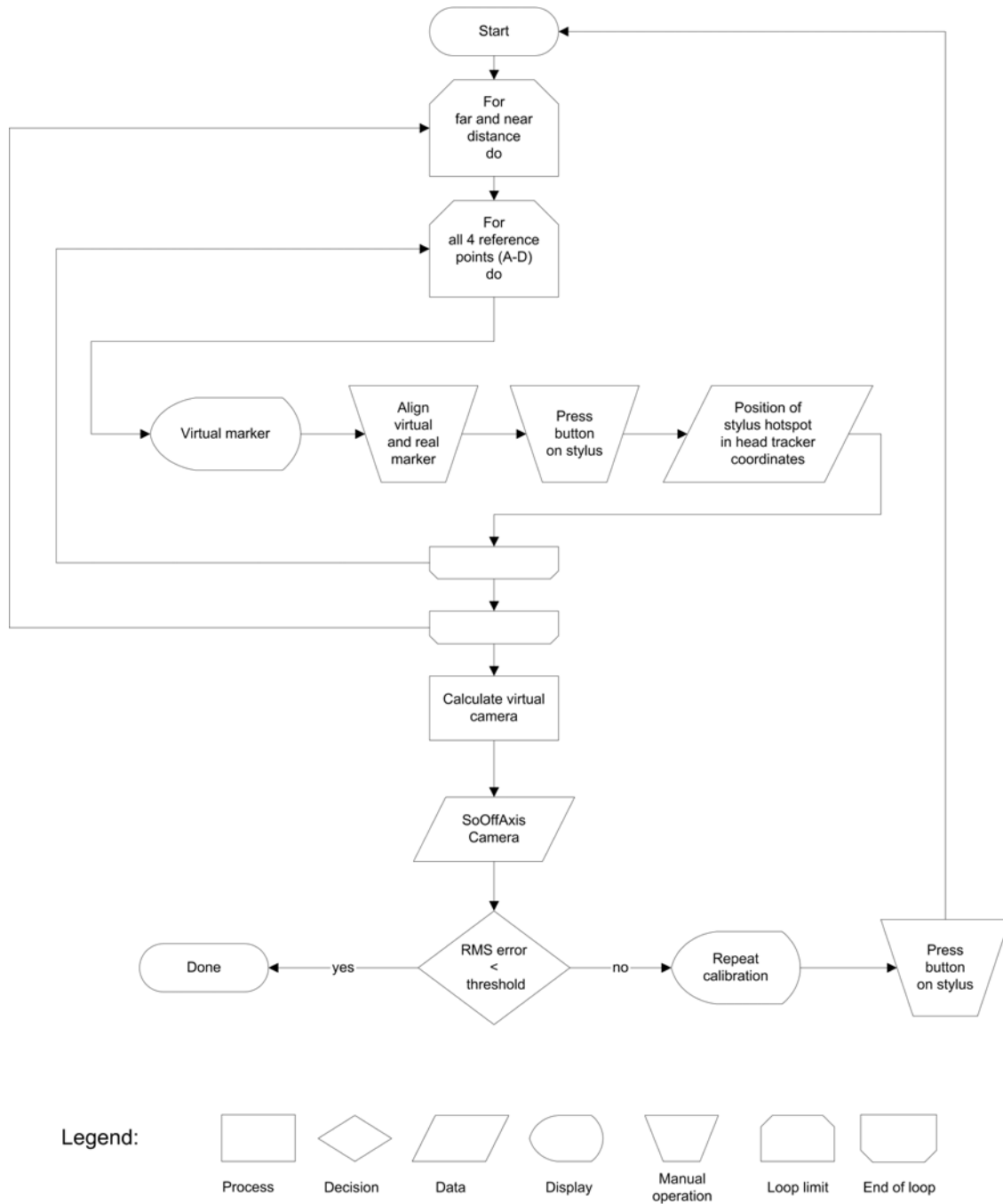


Figure 15 Calibration process for one eye

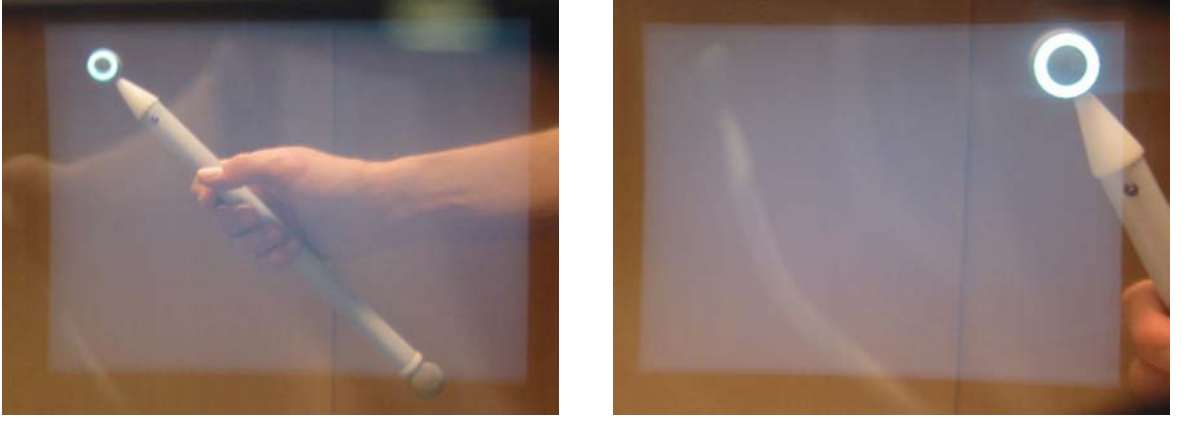


Figure 16 Alignment of virtual and real marker for far sample point (left) and near sample point (right)

Geometric Determination of Camera Parameters

Since we want to keep the number of sampled data points low, we need to maximize their information content with respect to our problem. We do this by imposing geometric constraints on the sampled points to allow direct determination of a viable start solution for our numerical optimization step. In Figure 14 the distribution of sample points for correct calibration is depicted as black circles (sample points 1-8). Every pair of samples lies on a line connecting one corner of the image plane with the eye point, essentially defining the viewing pyramid in this way.

In a first optimization step this gives us the location of the eye-point as a least-squares solution for the point lying nearest to all of these lines. The optimization is again (like in section 4.1) utilizing ‘powell’s algorithm’ [Press88].

The function to be minimized is given by:

$$f(e) = \sum_{i=1}^4 |l_i - e|^2$$

e is the eyepoint

l_i is the point on the i -th line with minimal distance to e

The starting value for powell’s algorithm is calculated by searching the shortest distance for all six possible pairs of lines and averaging the position of the six center points of the lines running along these shortest distances.

We now use the calculated eye-point to estimate the parameters of the projection plane. By averaging the directions of all rays from eye-point to the four far sample points we get a good approximation of the viewing direction. For this first estimation step we assume, that the viewing direction is normal to the projection plane, hence the location of the center of the projection plane lies on a ray starting at the eye-point and pointing in the estimated viewing direction. The distance between eye-point and center of the image plane as well as the size of the projection area is an assumption based on the knowledge, that the horizontal field of view of the used HMDs is approximately 30 degrees wide and the aspect ratio of the displays is about 4:3. Since position as well as size of the projection plane is optimized in the next step, an estimation of these parameters from the sampled points does not gain us an advantage over the assumption. Both approaches are nevertheless feasible, but the assumption gives us better control of the starting values used for the following optimization of the camera parameters. To estimate the remaining degree of freedom of the projection plane orientation, we intersect the approximated image plane with the lines $(1/2)$ and $(5/6)$, which gives us an approximation for the vertical direction, i.e. the up-vector of the projection plane. This intermediate solution already gives a good approximation of the calibration problem. Since this preliminary result only holds for eye positions on the axis of the optical system, we have to append an optimization procedure to account for off-axis positions of the eye. Any errors made in the assumption of FOV and aspect ratio due to variances in the production of the HMDs will also be corrected by the following optimization.

Numerical Optimization of Parameters

Since - as already mentioned above - the solution at this stage is already reasonably good, we do not have to apply sophisticated optimization techniques to it. A rather simple multi-dimensional least-squares optimization (powell's method) is being applied to the geometric solution reached in the previous step. The parameters optimized in this step are:

- Position of eyepoint,
- position,
- orientation,
- and height of the projection plane.

The result of the evaluation function used for powell's algorithm is calculated as follows:

The parameters given to the function (position of eyepoint, position, orientation, height of the projection plane) describe together with the fixed

parameter (width of projection plane), a virtual camera (cam) based on the Studierstube offaxis camera model. Now the 3D-sample (s) of a quintuple of calibration data is projected to screen space according to the current parameters of the virtual camera. The resulting 2D-position should ideally be congruent with the 2D-position (r) of the quintuple (the location of the center of the virtual marker, when the 3D-position was sampled). The distance between the two 2D-positions gives the error within a quintuple:

$$\mathbf{error}(\text{cam}, s, r) = |\mathbf{projectToScreen}(\text{cam}, s) - r|$$

cam	is the virtual camera to test; the camera parameters that are being optimized are: position, orientation, height of the projection plane and eyepoint position
s	is the sampled 3D-position
r	is the reference point, i.e. position of the virtual marker in screen space
<i>projectToScreen</i>	is the function projecting a given 3D-position to screen space using the given camera (parameters)

The return value of the evaluation function is then given by the sum of the quadratic errors of all eight quintuples:

$$\mathbf{eval}(\text{cam}) = \sum_{i=1}^8 \mathbf{error}(\text{cam}, s_i, r_i)^2$$

So figuratively speaking, during the optimization process the projection plane is moved around, rotated and squeezed until the best solution is found. Due to the fact that the preliminary geometric solution is close to the final result and the width of the projection area is fixed - i.e. the variability of the parameters to be optimized is rather restricted and degeneration of the virtual camera is prevented - the optimization runs stable and converges quickly.

In a second optimization step the two quintuples of calibration data, which produce the largest error with the current solution, are removed from the whole set of quintuples and the optimization method described above is repeated with the reduced data set. This step is added to compensate for some erroneous data that may have been sampled.

HCI aspects: “Guiding and constraining user input”

But as mentioned above the whole procedure is rather sensible to erroneous input data, which is sampled interactively by the user. Therefore we added some input and evaluation constraints to enhance and control the quality of the input data itself.

After starting the HMD-calibration application from the ‘calibration-suite’ menu the user can see a crosshair representing the first virtual marker. She now has to align this virtual marker with the marker mounted on the pen’s tip. To guide the user to move the pen’s tip to the correct position, she is obviously led by the displayed crosshair itself, which indicates the intended direction from the user’s eyepoint, i.e. a ray from the user’s eyepoint on which the sample position should lie. To indicate the favored distance (far or near distance) from the user’s eye, where the sampling should be triggered the size of the virtual marker is adjusted to the size of the real marker as seen from the preferred distance. By changing the color of the virtual marker from ‘inactive’ to ‘active’ color (which can be scripted by the ‘calibration suite administrator’), the user gets feedback, whether she has moved the pen’s tip into the range of the desired distance and may trigger the sampling, i.e. press the pen button, or not. Since the correct sampling distance is enforced by the constraint described above, the correct directional alignment of the markers remains as single source of incorrect user input.

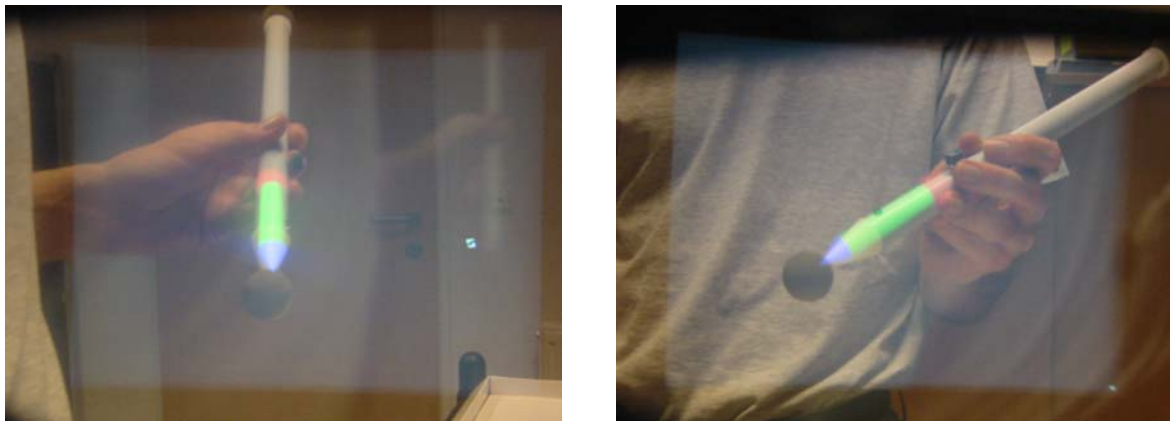


Figure 17 Visual control of the achieved HMD-registration (alignment of real and virtual stylus)

After the user has sampled all eight positions for one eye, the calculation of the camera parameters is started. The first step as stated above is the calculation of the eyepoint. At this stage the quality of the input data is probed, by imposing

an upper limit on the shortest distance between each possible pair of lines. If the criterion is not met the user is prompted to repeat the acquisition step, otherwise the calculation continues. At the end of the optimization step the RMS-error over all calibration data quintuples is calculated, which has to be smaller than a specified upper limit. As before not meeting the criterion results in a rollback to the calibration data acquisition step.

When the calculated virtual camera is valid, the whole procedure is repeated for the second eye. After successfully calibrating the second virtual camera the user may choose to test the achieved quality of the calibration. She can switch between the currently and the previously used camera registration and compare how good the displayed virtual stylus overlays the real stylus from different view angles and distances (Figure 17).

If a good registration is achieved the user just exits the HMD-calibration application with the current registration staying active and may begin working with the Studierstube system. She may also choose to repeat the whole procedure, if the result was not satisfactory, which can occur, when the quality criteria described above are too forgiving. The HMD registration is saved to a file and may be used as default registration of the HMD the next time the Studierstube system is started, though it will not be optimal for other users or even the same user, who calibrated the HMD, as stated at the begin of section 4.2.2. Nevertheless even a sub optimal registration may sometimes be adequate, e.g. for testing mechanisms of an application during development, where the tester can compensate for a small error in the calibration of the display system.

4.2.3. Calibrating projection systems

Here we use a variation of the calibration method we applied to HMDs in the previous chapter (section 4.2.2), which differs from [Summers99] method in the use of the stylus instead of additional markers and a bar attached to the shutter glasses (see section 3.2.5).

Overview of the calibration process

Like for the HMD-setup we want to retrieve the correct viewing parameters for the Studierstube off-axis camera model, i.e. the correct virtual camera, which ideally recreates the user's view to the projection display device. Contrary to the HMD-setup, where the relative position of the eyepoint to the projection plane is static for one user, but the whole virtual camera follows the movement of the user's head, the position and orientation of the projection plane typically remains static for a given projection setup. Therefore the calibration of the viewing parameters for a projection setup is divided into two steps:

1. Calibration of projection plane parameters
2. Calibration of eyepoint-position relative to head-tracker

Calibration of projection plane

The position and orientation of the display screen and the area actually utilized by the device are determined by this task. This is done by projecting reference markers near the four corners of the screen and measuring their position (Figure 20 left) by touching the displayed markers with the tip of the stylus and pressing the pen's button. Hence an obvious prerequisite for this step is, that the whole surface of the projection display has to lie within the working volume of the utilized tracker.

After the position of all four markers (upper left (S_{ul}), lower left (S_{ll}), upper right (S_{ur}), lower right (S_{lr})) has been retrieved, the absolute position and orientation of the projection plane in tracker coordinates can be calculated (Figure 18). Firstly the plane, which is described by any three sampled points, is optimized, so that the sum of the shortest distances between a point and the plane is minimized. Then all four sample points are projected to this plane, hence all points are now certainly laying on one plane. The normal of the plane gives us the projection plane's z-vector. The vectors $S_{ul}-S_{ll}$ and $S_{ur}-S_{lr}$ are averaged to

calculate the up-vector of the projection plane. Hence the orientation of the projection plane is specified. The origin (position) of the projection plane is calculated by intersecting the lines d_1 and d_2 given by:

$$d_1 = S_{ul} + t(S_{ur} - S_{ul}) \text{ and} \\ d_2 = S_{ul} + t(S_{lr} - S_{ul}).$$

The size of the projection plane is calculated by averaging the length of the vectors given by:

$$S_{ul} - S_{lr}, S_{ur} - S_{lr} \text{ (height) and} \\ S_{ur} - S_{ul}, S_{lr} - S_{ul} \text{ (width).}$$

Since the projection plane typically remains static for a given setup, the calibration of the projection plane usually has to be performed only once per setup.

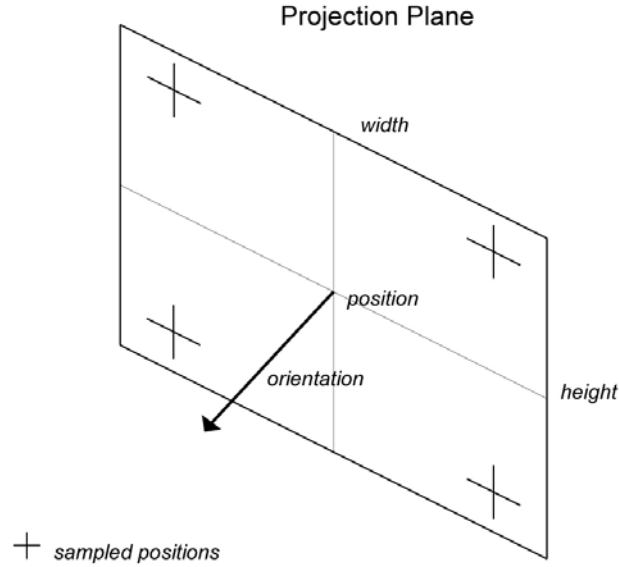


Figure 18 Parameters calculated during projection plane calibration.

Calibration of eyepoint-position

To determine the eye to tracker offset for both eyes, which implicitly also specifies the inter-pupillary distance of the user, we project the four markers — of which we sampled the absolute location in tracker coordinates in the previous step — on the screen again. The user brings the tip of the stylus in

alignment with a displayed virtual marker and presses the button accordingly (Figure 19 and Figure 20 right). At the press of the button the position of the stylus is sampled as well as the position and orientation of the head-tracker. The procedure is repeated for each eye and eventually gives us the desired offset from the head-tracker for the left and right eye.

For each measurement we now have:

- head position
- head rotation
- stylus' hotspot position
- marker position

The sampled hotspot position and the previously retrieved marker position define a line in space, which we transform into head-relative coordinates utilizing the simultaneously sampled head position and rotation. Due to this transformation we remove the influence of the user's head motion between the four sampling steps, i.e. the user does not have to hold her head fixed at one position for this procedure. The four gathered lines are now used to determine the user's eye positions relative to the head-tracker. Since in most cases the lines will not precisely intersect, we use an optimization procedure as we used in section 4.3.2. for the calculation of the eyepoint-position, resulting in an eye position with minimal distance to all measured lines.

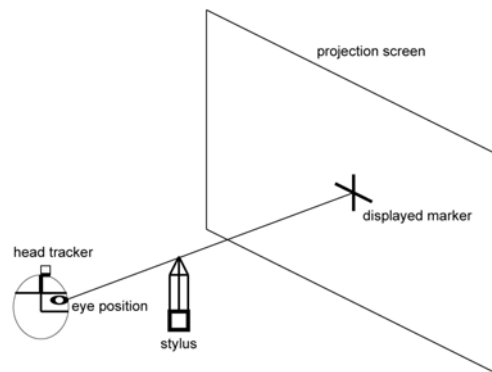


Figure 19 Eyepoint calibration for a projection set-up

After starting the projection calibration application from the 'calibration-suite' menu the user can choose between 'full calibration' (step 1 and 2 will be performed) and 'eye calibration' (only step 2 will be executed). When choosing 'full calibration' the first virtual marker will be displayed. Now the user has to touch the virtual marker with the pen's hotspot and press the button of the pen, then the next marker will be displayed. After all four marker positions where measured the projection plane parameters are calculated. If the plane cannot be

optimized good enough, i.e. the residual is greater than a threshold value, the plane calibration has to be repeated.

Having successfully completed step 1, step 2 is started automatically and the first marker is displayed again. The user has to align the tip of the pen with the center of the virtual marker. Two criteria for positioning the stylus during data acquisition have to be observed by the user to guarantee a valid sample of measurements:

- the stylus should be much nearer to the user's eye than to the marker on the screen
- the user's head should always point in the general direction of the screen center (not in the direction of the active marker), so that the angle between any two lines is maximized and hence a higher precision of the intersection is achieved.

To guide the user to position the pen's tip at the correct distance from the virtual marker and his eyes, the marker is colored differently indicating a 'valid' or 'invalid' distance, i.e. if the user may trigger the sampling now or not. Additionally text is shown on the display to tell the user what to do (e.g.: "The stylus' tip is too near to the projection plane, move it closer to your eye"). The same is done to enforce great enough angles between the lines, i.e. the user is told to rotate his head either left or right and the marker's color indicates whether the user struck the right pose.



Figure 20 Calibrating the Virtual Table: projection plane calibration (left), eye position calibration (sighting along the stylus' tip) (right). (For illustration purposes, the displayed marker has been enhanced.)

After the user has acquired the data for all four marker positions the eyepoint is calculated as described above. If the upper limit for the shortest distance between each possible pair of lines is exceeded the user is prompted to repeat

the acquisition step, otherwise the whole procedure is repeated for the second eye.

If a good registration is achieved the user just exits the projection device calibration application with the current registration staying active and may begin working with the Studierstube system. The registration (i.e. the camera parameters for left and right eye) is saved to a file and may be used as default registration of the display device setup the next time the Studierstube system is started. Since the parameters of the projection plane do not change, when the same setup is used again, usually only the eyepoint-calibration step (called step 2 above) has to be repeated on a per user basis to achieve good registration.

4.3. Registration of Tracker to World Coordinate System

The next step in the proposed calibration method is the registration of the tracker system so that it corresponds to the selected world coordinate system, i.e. the determination of the transformation from tracker coordinate system to world coordinate system (T_{tw}). Per default the tracker and world coordinate system are identical. Thus, when e.g. using a typical magnetical tracker setup, the origin of the world coordinate system lies somewhere near the middle of the tracker emitter. The Studierstube system places its 3D-windows and other virtual interaction elements around the world origin. Hence for the sake of consistency between and usability of the various different possible setups of the Studierstube system the world origin should lie at or near the center of the users' working volume (center of the tracker's working volume), and not at some point forced by the utilized tracking system.

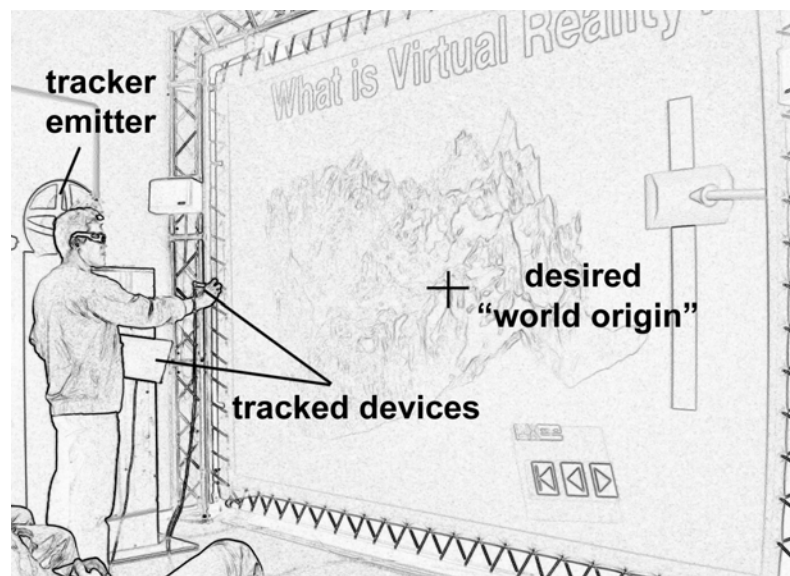


Figure 21 A typical AR system setup, where the tracker emitter is placed on the side of the user's working volume. In this case the desired origin of the world coordinate system is the center of the projection wall.

The calibration process

Previously the transformation T_{tw} was estimated and the appropriate tracker configuration files were altered manually, then the Studierstube system was

started and the result of the transformation could be tested. For every change of the transformation T_{tw} , one had to exit the Studierstube system, alter the configuration files again and then the Studierstube system was started again. In contrary to this rather tedious approach the proposed calibration procedure represents an interactive process, where the user can change the transformation T_{tw} at runtime and get immediate visual feedback how her changes affect the system.

The principle of the procedure is quite simple: The user just uses the stylus to ‘move around’ the world coordinate system, represented by a virtual object (coordinate axes), until it is placed at the desired position and orientation. The tracker to world transformation matrix T_{tw} is then given by:

$T_{tw} = T_{wt}^{-1}$, if the world and tracker coordinate systems were identical before or
 $T_{tw} = T_{tw_old} T_{wt}^{-1}$, if a transformation (T_{tw_old}) was applied previously.

Since we know T_{wt} , which is given by the transformation that was applied to the virtual coordinate axes by the user’s interaction, T_{tw} can be directly calculated from T_{wt} by calculating its inverse.

Integration into Studierstube using OpenTracker

To incorporate the resulting transformation in the Studierstube system, the program has to change the currently used tracker parameters. Since Studierstube uses the OpenTracker framework as interface to all tracker data (see section 5.3), the transformation has to be included in the OpenTracker tree, which describes the actual tracker configuration at runtime. The transformation has to be applied to all tracker stations of the utilized tracking system. Hence for every StbSink node present in the tracker tree a transform node, which represents the calculated tracker to world transformation, has to be inserted as first child. The new configuration of the whole tracker framework is then stored in a file, so that it can be easily loaded the next time Studierstube is started. So after the registration of tracker to world coordinate system has been achieved, i.e. the desired transformation T_{tw} had been inserted in the tracker tree, all further tracker data fed into the Studierstube system by the OpenTracker framework will be relative to the interactively defined world coordinate system.

Since all offsets retrieved in previous steps (section 4.1 and 4.2) are relative to a tracker sensor, they are not affected by the transformation T_{tw} . The only parameters calculated previously, which are specified in absolute tracker coordinates, are those describing the projection plane for a projection system

(section 4.2.3). Hence the projection plane parameters are transformed into the world coordinate system, if the current setup utilizes a projection system. The recomputed values are also written back to the configuration file, which describes the virtual camera parameters, to save them for subsequent runs of Studierstube utilizing the current setup.

HCI aspects: “Modes of interaction”

To fully utilize the interaction widgets provided by the Studierstube system, the display device (section 4.2) and the PIP (section 4.4) have to be calibrated. The user interface for this procedure provides support for application control utilizing the PIP. Alternatively the application can also be controlled via input from the keyboard.

After starting the application from the ‘calibration-suite’ menu the user can see the virtual coordinate axes, which visualize the current origin and orientation of the world coordinate system. The user has the following options to position the virtual coordinate axes and thus specifying the world coordinate system (WCS):

1. Snapping of WCS-origin to hotspot of stylus
2. Automatic alignment of WCS to projection plane
(only for projection display setups)
3. Direct (6DOF) manipulation of WCS’s position and orientation
4. Direct (3DOF) manipulation of WCS’s position with fixed orientation
5. Direct (3DOF) manipulation of WCS’s orientation with fixed position

The user can choose a specific manipulation mode by either pressing a key on the keyboard or by pressing a virtual button displayed on the PIP.

At the start of the process the virtual WCS may be out of reach of the user, e.g. when the WCS is identical to the tracker coordinate system, resulting in the WCS-origin being close to the tracker emitter, which may be mounted on the ceiling. Therefore mode 1 should be chosen, which lets the origin of the WCS snap immediately to the position of the stylus’ hotspot, when the button on the stylus is pressed. For projection-based displays, where an alignment of the WCS to the projection screen coordinate system is preferred mostly, the application provides the option to achieve this alignment automatically (mode 2).

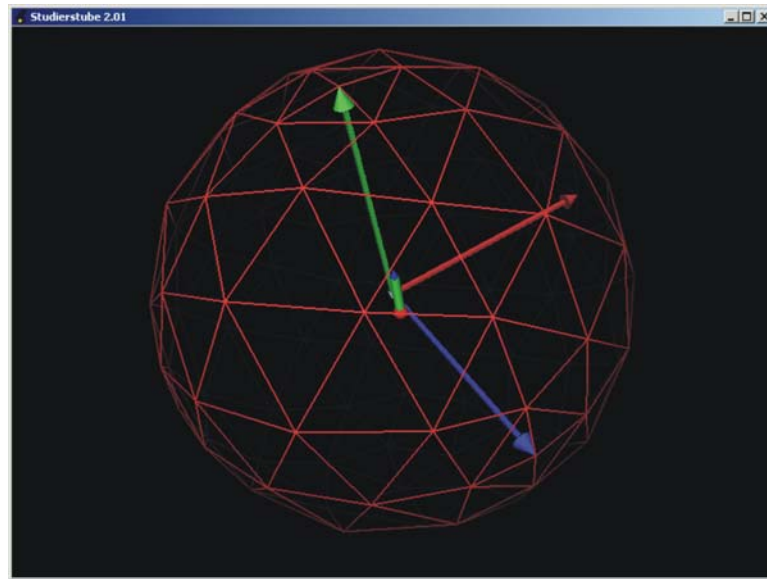


Figure 22 The virtual coordinate system; the wire frame sphere indicates, that it currently follows the movement of the stylus.

The user can also move the virtual WCS around arbitrarily using the stylus. The virtual WCS can easily be grabbed by moving the stylus within the bounding box of the virtual WCS, which is indicated by the appearance of a spherical wire frame around the object, then pressing and holding the stylus' button (Figure 22). The object follows every movement of the stylus until the pen's button is released again. If either the position or orientation is already satisfactory (e.g. alignment of WCS with projection screen), it can be locked again by pressing the associated virtual button on the PIP or with a key press. Consequently the drag and drop mechanism described above changes only orientation or position of the virtual WCS. If the user is pleased with the alignment of the WCS, she may choose to apply it to the system. The user may change the transformation T_{tw} multiple times, until she is satisfied with the achieved result. Every change made to the WCS may also be undone step by step. After the user exits the application she can continue working with the Studierstube system.

4.4. Calibration of Props

The last step of the proposed calibration method deals with the calibration of props. Props for the Studierstube system are defined as physical objects of which virtual representations need to exist in the system, either to be able to augment them or to use their virtual representation internally to e.g. calculate occlusions between real and virtual objects. A good static registration of all props is of paramount importance, since any misalignment between the physical object and its virtual representation results in a compromised augmentation as stated in section 2.

There are two sorts of props:

- Stationary props and
- Tracked props

Stationary props — e.g. desks, walls — are calibrated within the world coordinate system and cannot be moved without invalidating their registration. Tracked props — e.g. a pad, or a mock-up to be augmented and moved — are calibrated within the object’s tracker coordinate system instead of the absolute world coordinate system. Hence a tracked prop can be moved around and still stays registered.

The calibration process

For any prop to be calibrated, features on the object to be used as calibration points must be specified. These features may be corners or attached markers, which can easily and unambiguously be touched with the stylus’ tip. Hence a prerequisite for this calibration process, especially for the calibration of stationary props, is that all specified feature points must lie within the working volume of the utilized tracker system. The positions of these features have to be described in the virtual representation of the prop, allowing for inferring the position of the

- stationary prop in the real world, i.e. the specified world coordinate system, (transformation T_{ow})
- tracked prop relative to the prop’s tracker sensor coordinate system, (transformation T_{ots})

by an appropriate fitting algorithm.

Again the first step in the process is the acquisition of calibration data. The user just touches a feature point with the stylus' tip and presses a button on the pen. When pressing the button the position of the stylus' hotspot is sampled. When calibrating a tracked prop the position and orientation of the tracked prop's sensor is measured additionally. This procedure is repeated for every defined feature point.

The next step of this process is the determination of a viable geometric solution of the fitting problem, which is then used as starting value for an optimizing procedure. In the case of a tracked prop the sampled positions of the stylus' hotspot h_w are transformed from world coordinate system (WCS) to tracker sensor coordinate system (TSCS). Since we also sampled the translation t_w and orientation R_{wTs} of the tracker sensor attached to the prop the pen's tip in TSCS is given by:

$$h_{Ts} = (h_w - t_w)R_{wTs}^{-1}$$

For every specified feature point we now have a corresponding 3D-point:

- h_{Ts} for tracked props (in TSCS) or
- h_w for stationary props (in WCS)

which will be simply referred to as h , because the following calculations are similar for both cases.

We need at least three feature points to be able to unambiguously calculate all degrees of freedom of the searched transformation T_{FH} (from 'feature-point' (object) coordinate system to 'hotspot' (destination) coordinate system). To determine a preliminary transformation we take the first three feature points and calculate the normal n_F of the plane these three points span. The rotation R_{FHI} is specified as the rotation needed to transform the normal of the feature point plane n_F into the normal of the plane spanned by the corresponding sampling points n_H ($n_H = n_F R_{FHI}$). By transforming the vector from the first to the second feature point f_2-f_1 with the previously calculated rotation R_{FHI} we get $(f_2-f_1)R_{FHI}$. The third degree of freedom of the rotational component of the transformation T_{FH} is then determined by the rotation R_{FH2} needed to rotate $(f_2-f_1)R_{FHI}$ to h_2-h_1 , the vector from the first to the second sampled position ($h_2-h_1 = (f_2-f_1)R_{FHI} R_{FH2}$). Hence the rotational part of the transformation T_{FH} is given by:

$$R_{FH} = R_{FHI} R_{FH2} .$$

The translational part of T_{FH} is then given by:

$$t_{FH} = h_i - (f_i R_{FH})$$

This preliminary solution is now fed to powell's optimization algorithm, which optimizes the transformation in such a way that the RMS-error given by:

$$\text{RMSError} = \sqrt{\frac{1}{n} \sum_{i=1}^n |h_i - (f_i T_{FH})|^2}$$

is minimized.

- n is the number of samples
- f_i is the i -th feature point
- h_i is the i -th position of the stylus' hotspot
- T_{FH} is the transformation that is being optimized (from the prop's own coordinate system, wherein the feature points are specified, to the destination coordinate system (TSCS or WCS), wherein the hotspot positions are specified)

The resulting optimized transformation is then inserted as SoTransform node into the scene graph representing the prop's virtual shape and saved to a file.

HCI aspects

Similar to many applications developed for the Studierstube system, the user can control the application by either pressing a specific key on the keyboard or by pressing a virtual button displayed on the PIP, which is of course the preferred interaction method. When speaking of 'pressing a button' in the following text, it is used as abstraction from the specific interaction method, i.e. both methods can be utilized.

After starting the application from the 'calibration-suite' menu the user can see the virtual representation of the prop specified first in the application's configuration file. It is displayed centered on the origin of the world coordinate system. By specifying all props that should be calibrated in the configuration file, the whole process can be streamlined, because the user is able to choose a prop quickly from the set of specified props by pressing the 'next' or 'previous' button. Alternatively the user might choose from a virtual file browser, which prop she wants to calibrate. But the props to be calibrated are known before the system is started and it is more convenient for the user to choose from a rather

small selection of props than browsing through many directories just to select a single object of interest.

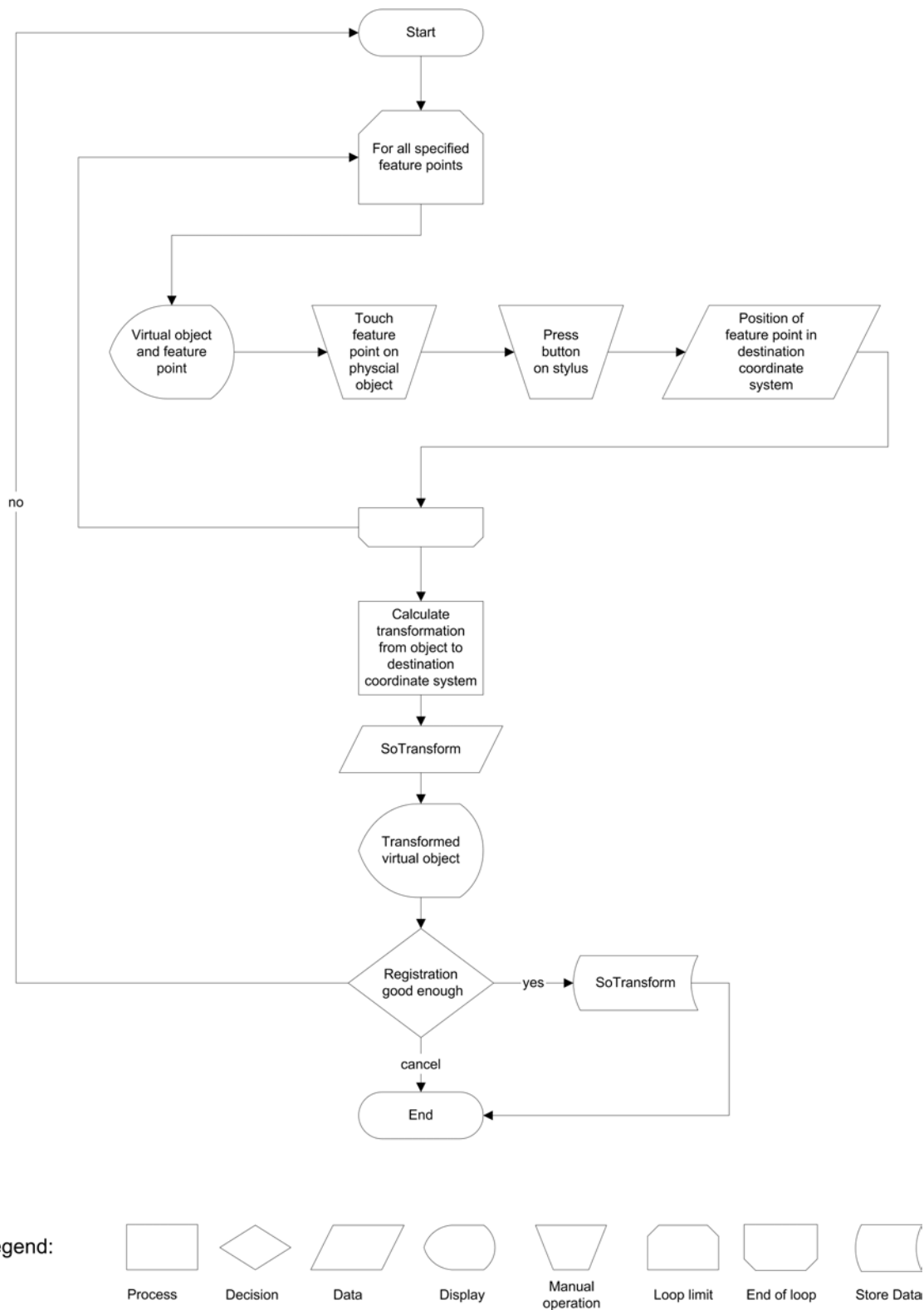


Figure 23 Calibration process for a prop

So after the user has chosen the prop she wants to calibrate, she presses the 'register' button to start the actual calibration process (see Figure 23). The first feature point is marked on the virtual representation with a pulsing crosshair, to give the user a visual clue where it is located on the physical object (see Figure 27). The user can arbitrarily move around the virtual prop using the stylus, to get a better impression of the model or to uncover the currently marked feature point, which might be occluded by the virtual object itself. The virtual prop can easily be grabbed by moving the stylus within its bounding box, which is indicated by highlighting the prop, then pressing and holding the stylus' button. The object then follows every movement of the stylus until the pen's button is released again. When the user has identified the feature point on the physical object, he has to touch it with the stylus' tip and press the pen's button. Then the next feature point is marked on the virtual prop. If the user knows she missed the feature point or pressed the pen button accidentally she can repeat the sampling.

After the user has acquired sampling data for all feature points the transformation is calculated and applied to the virtual representation of the prop. The user can now judge the quality of the achieved registration and choose to return to the prop selection stage with or without saving the result of the calibration process.

Chapter 5

Implementation

5.1. The Studierstube System - Implementation of the user interface

5.1.1. Software architecture

Studierstube's [Schmalstieg00] software development environment is realized as a collection of C++ classes built on top of the Open Inventor (OIV) toolkit [Strauss92]. The rich graphical environment of OIV allows rapid prototyping of new interaction styles. The file format of OIV enables convenient scripting, overcoming many of the shortcomings of compiled languages without compromising performance. At the core OIV is an object-oriented scene graph storing both geometric information and active interaction objects. The implementation approach has been to extend OIV as needed, while staying within OIV's strong design philosophy [Wernecke94]. This has led to the development of two intertwined components: A toolkit of extensions of the OIV class hierarchy—mostly interaction widgets capable of responding to 3D events—and a runtime framework which provides the necessary environment for *Studierstube* applications to execute.

Together these components form a well-defined application programmer's interface (API), which extends the OIV API, and also offers a convenient programming model to the application programmer (see section 5.1.3).

Applications are written and compiled as separate shared objects, and dynamically loaded into the runtime framework. A safeguard mechanism makes sure that only one instance of each application's code is loaded into the system at any time. By using this dynamic loading mechanism, *Studierstube* supports multi-tasking of *different* applications (e.g. a medical visualization and a 3D modeler) and also a multiple document interface (MDI). Depending on the semantics of the associated application, ownership of a context may or

may not privilege a user to perform certain operations on the information (such as object deletion).

PIP sheets

Studierstube applications are controlled either via direct manipulation of the data presented in 3D-windows, or via a mixture of 2D and 3D widgets on the PIP (Figure 24). A set of controls on the PIP— a *PIP sheet*—is implemented as an OIV scene graph composed primarily of *Studierstube* interaction widgets (such as buttons, etc.). However, the scene graph may also contain geometries (e.g., 2D and 3D icons) that convey the user interface state or can be used merely as decoration. Every type of context defines a PIP sheet template, a kind of application resource. For every context and user, a separate PIP sheet is instantiated. Each interaction widget on the PIP sheet can therefore have a separate state. For example, every user for every context can set the current paint color in an artistic spraying application individually. However, all users and/or all contexts can also share widgets. Consequently, *Studierstube*'s 3D event routing involves a kind of multiplexer between windows and users' PIP sheets.

5.1.2. Hardware support

Displays

Studierstube is intended as an application framework that allows the use of a variety of displays, including projection based devices and HMDs. There are several ways of determining camera position, creating stereo images, setting a video mode etc. An OIV compatible viewer with a plug-in architecture for camera control and display mode was implemented to meet these requirements. The following display modes are supported:

- Field sequential stereo: Images for left/right eye output in consecutive frames
- Line interleaved stereo: Images for left/right eye occupy odd/even lines in a single frame
- Dual screen: Images for left/right eye are output on two different channels
- Mono: The same image is presented to both eyes

The following camera control modes are supported:

- Tracked display: Viewpoint and display surface are moving together and are tracked (usually HMD)
- Tracked head: A user's viewpoint (head) is tracked, but the display surface is fixed (such as a workbench or projection wall)
- Desktop: The viewpoint is either assumed stationary, or can be manipulated with a mouse

This approach, together with a general off-axis camera implementation (see section 4.2.1), allows runtime configuration of almost any available display hardware.

Tracking A software system like *Studierstube* that works in a heterogeneous distributed infrastructure and is used in several research labs with a variety of tracking devices requires an abstract tracking interface. The approach taken by most commercial software toolkits is to implement a device driver model, thereby providing an abstract interface to the tracking devices, while hiding hardware dependent code inside the supplied device drivers. While such a model is certainly superior to hard-coded device support, it falls short for various requirements:

Configurability: Typical setups for tracking in virtual environments are very similar in the basic components, but differ in essential details such as the placement of tracker sources or the number and arrangement of sensors. The architecture allows the configuration of all of those parameters through simple scripting mechanisms.

Filtering: There are many necessary configuration options that can be characterized as filters, i.e., modifications of the original data. Examples include geometric transformations of filter data, prediction, distortion compensation, and sensor fusion from different sources.

Distributed execution and decoupled simulation: Processing of tracker data can become computationally intensive, and it should therefore be possible to distribute this work over multiple CPUs. Moreover, tracker data should be simultaneously available to multiple users in a network. This can be achieved by implementing the tracking system as a loose ensemble of communicating processes, some running as service processes on dedicated hosts that share the computational load and distribute the available data via unicast and multicast mechanisms, thereby implementing a decoupled simulation scheme.

Extensibility: As a research system, *Studierstube* is frequently extended with new experimental features. A modular, object-oriented architecture allows the rapid development of new features and uses them together with existing ones. The latest version of tracking support in *Studierstube* is implemented as an object-oriented framework called OpenTracker [Reitmayr00] (see section 5.3), which is available as open source. It is based on a graph structure composed of linked nodes: source nodes deliver tracker data, sink nodes consume data for further processing (e. g. to set a viewpoint), while intermediate nodes act as filters. By adding new types of nodes, the system can easily be extended. Nodes can reside on different hosts and propagate data over a network for decoupled simulation. By using an XML [Bray00] description of the graph, standard XML tools can be applied to author, compile, document, and script the OpenTracker architecture.

5.1.3. Application programmer's interface

The *Studierstube* API imposes a certain programming model on applications, which is embedded in a foundation class, from which all *Studierstube* applications are derived. By overloading certain polymorphic methods of the foundation class, a programmer can customize the behavior of the application. The structure imposed by the foundation class supports multiple contexts. Context nodes are implemented as OIV *kit* classes. Kits are special nodes that can store both fields, i.e., simple attributes, and child nodes, both of which will be considered part of the scene graph. Default parts of every context are at least one 3D-window node, which itself is an OIV kit and contains the context's "client area" scene graph, and a set of PIP sheets (one for each participating user). In other words, data, representation, and application are all embedded in a single scene graph, which can be conveniently managed by the *Studierstube* framework.

To create a useful application with all the properties mentioned above, a programmer needs only create a subclass of the foundation class and overload the 3D-window and PIP sheet creation methods to return custom scene graphs. Typically, most of the remaining application code will consist of callback methods responding to certain 3D events such as a button press or a 3D direct manipulation event, although the programmer has the freedom to use anything that the OIV and *Studierstube* toolkits offer.

5.2. Human-Computer Interaction (HCI) aspects

When having the goal of providing easy to handle tasks for calibrating the AR system, which [Azuma97b] calls a “*desirable result*”, the aspect of providing adequate human-computer interaction is of paramount importance to reaching it. [Szalavári99] proposes basic design guidelines for AR interfaces, which led to the design of the PIP interface (see section 5.1.1), which now is the standard user interface provided by *Studierstube*.

Where applicable the PIP interface is used for controlling the different calibration applications. Since the PIP interface is only useful, when all components (pen, panel) and the display device are calibrated, i.e. the *Studierstube* 3D workspace is already setup and calibrated, it obviously cannot be used for the display device and stylus calibration procedures. For these calibration steps a simple 2D menu interface was created (Figure 25). The navigation through the menu items can be controlled via keyboard or the stylus’s buttons. By pressing the secondary button of the stylus, the user can cycle through the menu items. The primary button triggers the action associated with the currently selected item. The control via keyboard works similarly, with the exception that the menus can be browsed up *and* down. For expert users additional keyboard commands – so-called hotkeys – are implemented, which trigger their associated action by a single key press.

Since we implemented a whole suite of calibration procedures, there are two levels of interaction that have to be considered.

The *first* is the navigation in between calibration processes. The interface that is provided for the user to choose the appropriate calibration task should be adaptable to the system’s registration status and the amount of control the user wants over the system level task, which is calibration. The following requirements have to be considered for the design of the user interface.

- To calibrate a new hardware setup the different calibration steps have to be performed in a predefined order.
- When the initial calibration was performed, only the display calibration has to be performed for each user.
- It must also be possible to randomly select the task, which the user wants to perform, e.g. if additional props have to be integrated in the system.

The *second* is the navigation through a particular calibration procedure. When looking at this type of interaction, it is important not only to tell the user what tasks she has to perform, but also guide the user, so that the result of the interaction, mostly sampling of tracker data, meets certain criteria. Therefore we implemented methods to instruct and guide the user while performing a calibration task.

5.2.1. Paths through the Calibration Process

The starting point for each calibration process is an application called calibration suite. This application manages all calibration tasks. It can be configured to run in different modes:

- **New setup guide**

The user is automatically guided through the whole setup process, and performs the necessary tasks in order. The calibration suite starts the particular calibration applications automatically and in order. Hence the user is only navigating through each necessary calibration task and is not concerned with choosing the application himself. For a typical HMD setup, the order of applications is as follows:

- Stylus calibration:
(only once for a particular system setup)
- HMD calibration:
(ideally for every user “entering” the AR environment)
- Tracker to World coordinate registration
(typically only once for a particular system setup)
- Registration of the PIP
(only once for a particular system setup)

- **Automatic display calibration**

After the initial calibration of a specific hardware setup is completed, the calibration suite switches to this mode. Every time the calibration suite is started, preferably when the system is started, the current user is asked to perform the display calibration task, to ensure proper display registration for every user.

- **Expert mode**

When this mode is activated, the user can select any task in any order. Hence she has full control over the calibration suite. To produce useful results the user has to know the current system setups' calibration status, to determine if the particular task chosen is dependent on another calibration task.

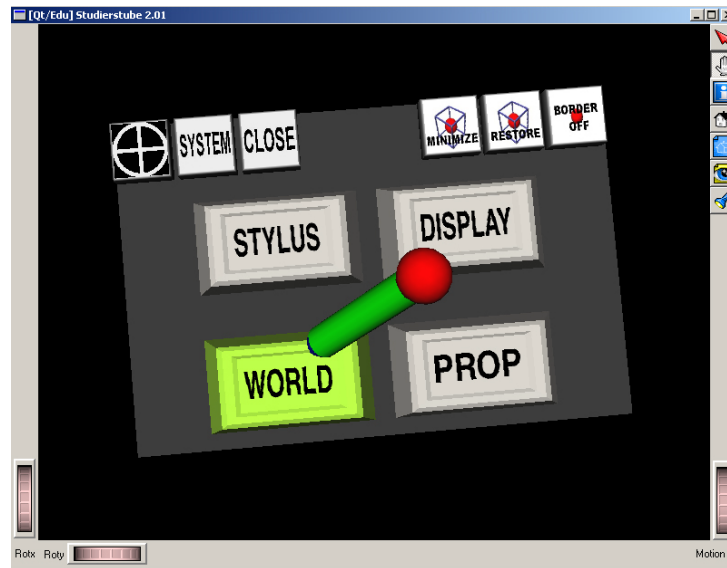


Figure 24 Screenshot of the calibration suite's PIP sheet in "expert mode".

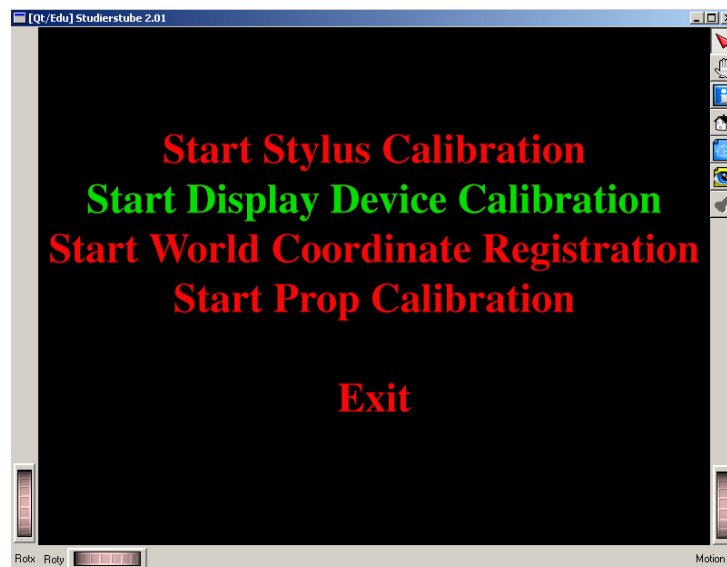


Figure 25 Screenshot of the 2D menu of the calibration suite when in "expert mode".

5.2.2. User guidance

To lead the user through the calibration process she is provided with the following types of information:

- Instructions, what step has to be performed next
- Status information for the currently running calibration task
- Visual feedback of meeting/not meeting certain input constraints and
- Instructions, what has to be done, to meet a certain input constraint

Instructions and status information are mostly displayed as simple 2D text on the utilized display (Figure 26). A scene object node for 2D text is part of the implementation of OIV. But for the sake of a more convenient way to setup and parameterize the scene sub-graph needed to display a message, we implemented a new node kit, called SoMessageKit. It can be easily parameterized within an IV-file or the application itself, e.g. to specify color, position, font style of the message, without having to construct the message scene graph explicitly.

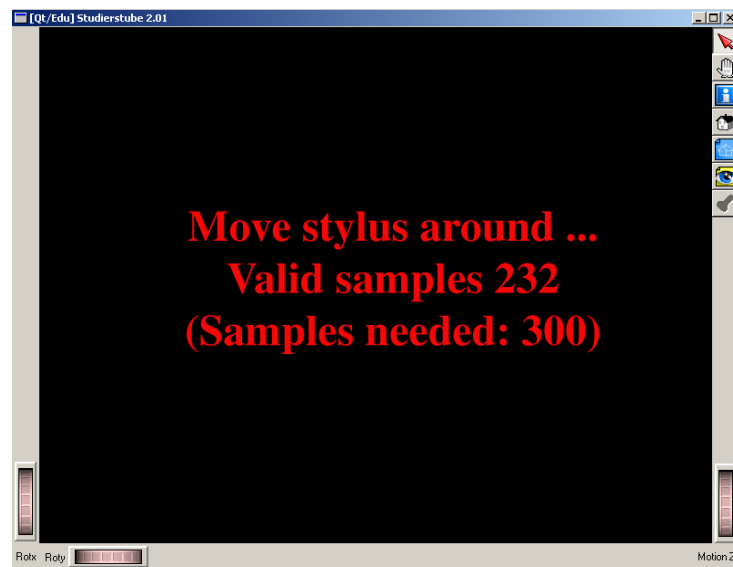


Figure 26 This screenshot shows an example of the provided status information during the stylus calibration procedure.

For the prop calibration the visualization of the feature point, which should be sampled next is an example of a non-text based instruction. The next feature point that the user has to sample – by placing the tip of the stylus on a specified point on the real object – is indicated by a pulsing 3D crosshair on the

visualization of the corresponding virtual object displayed on the utilized display (Figure 27). Hence the user gets direct visual context specific information.

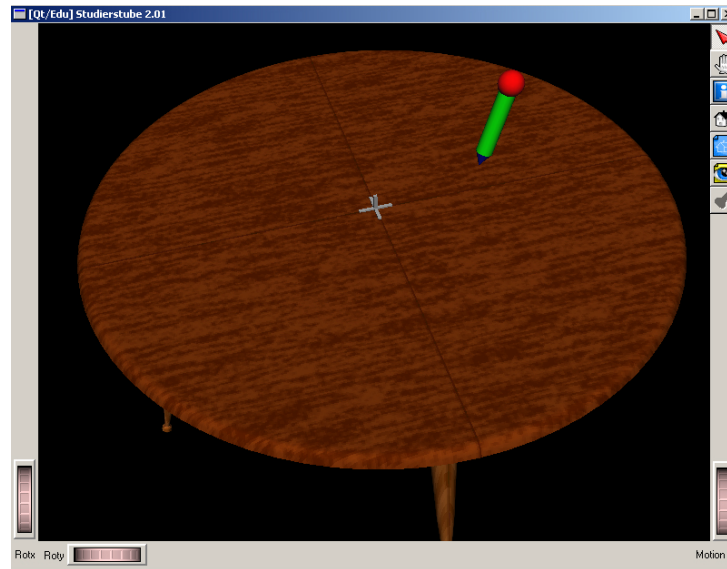


Figure 27 Screenshot of the visualization of the feature point (in the middle of the table), which should be sampled next during the prop calibration process.

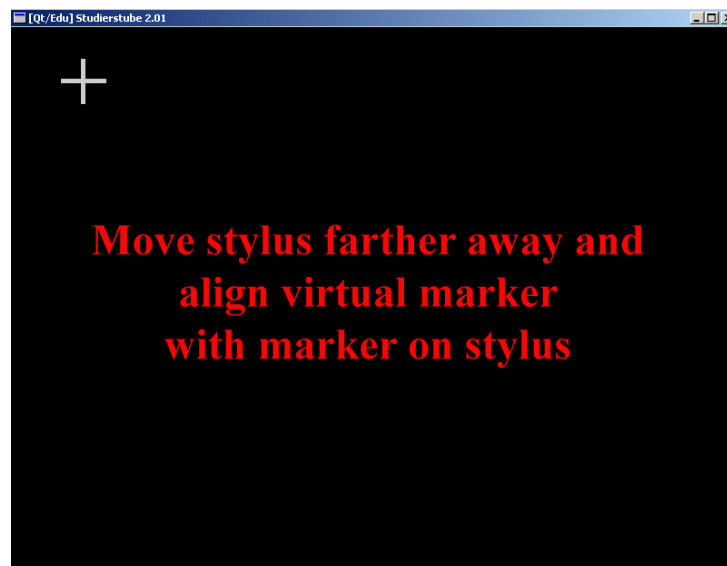


Figure 28 This screenshot shows the use of 2D text and visual feedback.

Especially during the display device registration procedures, the visual feedback, which indicates the validity of the current stylus position, i.e. if the

current position meets the predefined constraint (e.g. angular or positional), is very helpful. Figure 28 shows a screenshot taken during an HMD calibration. The white cross hair indicates, that the current relative position of the stylus in respect to the HMD does not allow the acquisition of a valid sample. Hence the 2D text is used to instruct the user, what to do to meet the criteria. When the cross hair turns red, the user gets the visual feedback that he now may trigger the sampling.

5.3. OpenTracker - An XML based Open Architecture for Reconfigurable Tracking

Tracking is an indispensable requirement for all kinds of virtual reality (VR) and augmented reality (AR) systems. OpenTracker is a tracking software system that allows mixing and matching of different features, as well as simple creation and maintenance of complex tracker configurations.

OpenTracker has the following characteristics:

- An object-oriented approach to an extensive set of sensor access, filtering, fusion, and state transformation operations
- Behavior specification by constructing graphs of tracking objects (similar in spirit to scene graphs or event cascades) from user defined tracker configuration files
- Distributed simulation by network transfer of tracker state at any point in the graph structure
- Decoupled simulation by transparent multi-threading and networking
- A software engineering approach based on XML [Bray00], which allows to use many generic tools for development, documentation, and configuration.

The current scope of OpenTracker is traditional VR applications. It thus deals primarily with position and orientation information (six degrees of freedom, 6DOF), although some other event types such as button events and 2D position information (such as from a desktop mouse) are supported.

The software is designed as a class hierarchy of tracker objects, implemented in C++. Every tracker object defines an interface that can answer a query for the current position and orientation as well as the state of the associated buttons. At runtime, these tracker objects are assembled into a directed acyclic graph (DAG)- or frequently, a set of DAGs - according to the instructions in a user-supplied configuration file written in XML. We distinguish *source objects*, which are leaves in the graph and receive their data values from external sources, *filter objects*, which are intermediate nodes and modify the values received from their child nodes, and *sink objects*, which propagate their data values received from their child nodes to external outputs.

Source objects

Most source objects encapsulate a device driver that directly accesses a particular tracking device, such as a Polhemus or Ascension tracker connected to a serial interface. Other source objects form bridges to complex self-contained systems, such as the video tracking library from ARToolKit [Kato99]. Yet other source objects emulate tracker via the keyboard or simply respond with constant values (useful for development and debugging) or access network data. Some source objects have a multi-threaded execution model to implement a decoupled simulation model (e. g., when blocking I/O must be used).

Filter objects

Filter objects have one or more children. When queried, filter objects pass on the query to determine the state of their children, and then compute their own state based on the returned data. A non-exhaustive list of filters includes:

- Transformation filters perform geometric transformations of their children's values. These include pre- and post-transformations and may be static or depend on data values received from other children. The latter allows modifying the filtered state relative to another tracker state.
- Prediction filters allow to partially compensate for lag in the measuring and processing tracker data.
- Noise and smoothing filters are handy to deal with inherent inaccuracies of trackers.
- ...

Sink objects

Sink objects are similar to source objects but distribute data rather than receive it. They include output to network multicast groups, debugging output to a user interface or shared memory to integrate *OpenTracker* as a library into other applications.

The presence of sink objects drives the evaluation model of *OpenTracker*. All sink objects in a tracker object graph are registered upon creation, and their respective state evaluation method is triggered periodically. [Reitmayroo] found this to be more effective than a pure client-driven lazy evaluation scheme, as it avoids potentially costly recomputation of intermediate values for every invocation.

Software engineering with XML

XML, the extensible markup language, is the emerging standard for web-based applications and software systems [Bray00]. XML is a markup definition language that allows defining hierarchical markup languages with so-called document type definitions (DTD). With the appropriate DTD, standard XML tools can be used to conveniently edit, type check, parse, and transform any XML file.

Thus, providing a simple DTD for describing hierarchies of tracker objects opens access to software libraries and tools that simplify several steps of the development cycle:

- A DTD editor can be used to design and maintain the DTD.
- An XML parser enforces content format on the tracker configuration file while building the corresponding structure in memory, thus automatically performing many of the consistency checks that have otherwise to be hand-coded.
- A convenient XML editor with a graphical user interface allows the end user to design the tracker configuration without having to master the syntax.
- Using the extendible style language (XSL) [Adler01], automatic textual and even graphical documentation can be created from a tracker configuration file.

Markup languages are generally used to annotate textual documents with structural information. Thus a general XML document consists of text grouped and structured with tags. Markup languages defined in XML consist of elements, essentially expressed as tags, and a structural model (the content model) of the possible ways these elements may be nested. Moreover, elements are annotated by name - value pairs called attributes.

OpenTracker maps elements to objects and attributes to members of these objects and does not use any textual content but purely relies on the content model provided by the DTD. An open source XML parser [Apache99] builds a tree of elements representing the given configuration file. *OpenTracker* walks the tree and creates a new object for each element based on the elements name. The string values of the attributes are parsed according to the objects class and the corresponding members are set. Attributes typically describe such data as the parameters of a transformation.

Restrictions on the number of children and the possible types are described in the DTD. *Source objects* typically do not have any children objects as they rely on data from external sources to compute their own data. A number of *filter objects* get the value of a single child object, transform it and pass it on. Confidence filters use any number of children to compute their data value. The data of the different children enters in the same way into the computation.

In another case different children objects influence the computation in different ways. Dynamical transformations, for example, are parameterized by the value of another object and thus use the data value of the object to be transformed differently from the data of the parameterizing object. This is handled by using wrapper elements. An object requiring marked children is mapped to an element that may only have certain marker elements as children. These marker elements in turn may have any other element as child again. The marker elements are mapped to marker objects that perform no special function and return the value of their child object. They can be queried by the filter object to derive how to use this value.

The following XML code describes a simple Studierstube tracker setup. *NetworkSource* nodes receive data, in this case tracker data from an ART optical tracking system, via multicast and *StbSink* nodes, representing the interface between *OpenTracker* and Studierstube, update the internal Studierstube tracking state. Station 1 represents the stylus tracking station. Stations 2 and 3 are used to track the PIP respectively the HMD.

```
<!DOCTYPE OpenTracker SYSTEM "opentracker.dtd">
<OpenTracker>
  <configuration>
    <NetworkSourceConfig />
  </configuration>
  <StbSink event="off" station="1">
    <NetworkSource number="1"
      multicast-address="224.100.200.102"
      port="12345" />
  </StbSink>
  <StbSink event="off" station="2">
    <NetworkSource number="2"
      multicast-address="224.100.200.102"
      port="12345" />
  </StbSink>
```

```

    <StbSink event="off" station="3">
      <NetworkSource  number="3"
        multicast-address="224.100.200.102"
        port="12345" />
    </StbSink>
</OpenTracker>

```

After the stylus calibration is finished, an *EventVirtualPositionTransform* node is inserted into the XML document, to store the calculated stylus offset and apply it to the tracker data coming from the station attached to the stylus. The used *EventVirtualPositionTransform* implements an offset in the child's affine space. That is the configured translation is post transformed with the child's position and orientation values. This effectively offsets the tracked point with respect to the tracked affine base, i.e. in this case, the origin of the tracker sensor is moved into the tip of the stylus.

```

...
<StbSink event="off" station="1">
  <EventVirtualPositionTransform translation="-0.063 -0.017 -0.009">
    <NetworkSource  number="1"
      multicast-address="224.100.200.102"
      port="12345" />
  </EventVirtualPositionTransform>
</StbSink>
...

```

After the registration of tracker to world coordinate is completed, the resulting transformation has to be applied to the whole tracker reference frame. Hence all data fed into the Studierstube framework – in this case all data coming from the tracker stations (*NetworkSource*) specified within the configuration file – has to be transformed. The *EventTransform* node transforms the data by applying a rotation, scale and translation to the child's data as post transformations. The transformation itself is fixed and set with the elements attributes. Only the rotational part acts on the child's orientation data.

```

<!DOCTYPE OpenTracker SYSTEM "opentracker.dtd">
<OpenTracker>
  <configuration>
    <NetworkSourceConfig />
  </configuration>
  <StbSink event="off" station="1">
    <EventTransform scale="1 1 1"
      rotation="0.0348 0.0523 -0.0316 0.998"
      translation="-0.773995 0.538000 0.000000"
      rotationtype="quaternion">
      <EventVirtualPositionTransform
        translation="-0.063 -0.017 -0.009">
        <NetworkSource number="1"
          multicast-address="224.100.200.102"
          port="12345" />
        </EventVirtualPositionTransform>
      </EventTransform>
    </StbSink>
    <StbSink event="off" station="2">
      <EventTransform scale="1 1 1"
        rotation="0.0348 0.0523 -0.0316 0.998"
        translation="-0.773995 0.538000 0.000000"
        rotationtype="quaternion">
        <NetworkSource number="2"
          multicast-address="224.100.200.102"
          port="12345" />
        </EventTransform>
      </StbSink>
      <StbSink event="off" station="3">
        <EventTransform scale="1 1 1"
          rotation="0.0348 0.0523 -0.0316 0.998"
          translation="-0.773995 0.538000 0.000000"
          rotationtype="quaternion">
          <NetworkSource number="3"
            multicast-address="224.100.200.102"
            port="12345" />
          </EventTransform>
        </StbSink>
      </OpenTracker>

```

5.4. Minimizing Functions - Direction Set (Powell's) Methods in Multidimensions

For many calibration tasks, the strategy to obtain good results is to find a rather good initial solution for the parameters by taking into account the known geometrical constraints (e.g. see section 4.2.2. - Calibrating See-Through Head-Mounted Displays), and then optimizing the desired parameters, so that the function calculating the registration error with a given set of parameters is minimized. Hence we have to use multidimensional minimization, which means finding the minimum of a function of more than one independent variable.

Minimization along a line in N -dimensional space

We know how to minimize a function of one variable (see [Press88] for further reference). If we start at a point P in N -dimensional space, and proceed from there in some vector direction n , then any function of N variables $f(P)$ can be minimized along the line n by a one-dimensional method. One can dream up various multidimensional minimization methods that consist of sequences of such line minimizations. Different methods will differ only by how, at each stage, they choose the next direction n to try. All such methods presume the existence of a “black-box” sub-algorithm, which we might call *linmin*, whose definition can be taken as:

linmin: Given as input the vectors P and n , and the function f , find the scalar λ that minimizes $f(P + \lambda n)$.
Replace P by $P + \lambda n$. Replace n by λn . Done.

For most of our calibration procedures, calculation of the gradient is out of the question, hence *Powell's* minimization method, which falls under the above stated general schema of successive line minimizations, and whose choice of successive directions does not involve explicit computation of the function's gradient is the method perfectly fitting our problem.

Multidimensional minimization

You might first think of this simple method: Take the unit vectors e_1, e_2, \dots, e_N as a *set of directions*. Using *linmin*, move along the first direction to its minimum, then from there along the second direction to *its* minimum, and so on, cycling through the whole set of directions as many times as necessary, until the function stops decreasing.

This simple method is actually not too bad for many functions. Even more interesting is why it *is* bad, i.e. very inefficient, for some other functions. Consider a function of two dimensions whose contour map (level lines) happens to define a long, narrow valley at some angle to the coordinate basis vectors. Then the only way “down the length of the valley” going along the basis vectors at each stage is by a series of many tiny steps. More generally, in N dimensions, if the function’s second derivatives are much larger in magnitude in some directions than in others, then many cycles through all N basis vectors will be required in order to get anywhere. This condition is not all that unusual; according to Murphy’s Law, you should count on it.

Obviously what we need is a better set of directions than the e_i ’s. All *direction set methods* consist of prescriptions for updating the set of directions as the method proceeds, attempting to come up with a set which either (i) includes some very good directions that will take us far along narrow valleys, or else (more subtly) (ii) includes some number of “non-interfering” directions with the special property that minimization along one is not “spoiled” by subsequent minimization along another, so that interminable cycling through the set of directions can be avoided.

Conjugate Directions

Suppose that we have moved along some direction u to a minimum and now propose to move along some new direction v . The condition that motion along v not *spoils* our minimization along u is just that the gradient stays perpendicular to u , i.e., that the change in the gradient be perpendicular to u . If not then there would still be a nonzero directional derivative along u .

When this condition holds for two vectors u and v , they are said to be *conjugate*. When the relation holds pair wise for all members of a set of vectors, they are said to be a conjugate set. If you do successive line minimization of a function along a conjugate set of directions, then you don’t need to redo any of

those directions (unless, of course, you spoil things by minimizing along a direction that they are *not* conjugate to).

A triumph for a direction set method is to come up with a set of N linearly independent, mutually conjugate directions. Then, one pass of N line minimizations will put it exactly at the minimum of a quadratic form. For functions f that are not exactly quadratic forms, it won't be exactly at the minimum; but repeated cycles of N line minimizations will in due course converge *quadratically* to the minimum.

Powell's Quadratically Convergent Method

Powell first discovered a direction set method that does produce N mutually conjugate directions. Here is how it goes: Initialize the set of directions u_i to the basis vectors,

$$u_i = e_i \quad i = 1, \dots, N$$

Now repeat the following sequence of steps ("basic procedure") until your function stops decreasing:

- Save your starting position as P_0 .
- For $i = 1, \dots, N$, move P_{i-1} to the minimum along direction u_i and call this point P_i .
- For $i = 1, \dots, N - 1$, set $u_i \leftarrow u_{i+1}$.
- Set $u_N \leftarrow P_N - P_0$.
- Move P_N to the minimum along direction u_N and call this point P_0 .

Powell, in 1964, showed that, for a quadratic form, k iterations of the above basic procedure produce a set of directions u_i whose last k members are mutually conjugate. Therefore, N iterations of the basic procedure, amounting to $N(N + 1)$ line minimizations in all, will exactly minimize a quadratic form [Brent73]. Unfortunately, there is a problem with Powell's quadratically convergent algorithm. The procedure of throwing away, at each stage, u_1 in favor of $P_N - P_0$ tends to produce sets of directions that "fold up on each other" and become linearly dependent. Once this happens, then the procedure finds the minimum of the function f only over a subspace of the full N -dimensional case; in other words, it gives the wrong answer.

To fix up the problem of linear dependence in Powell's algorithm you can give up the property of quadratic convergence in favor of a more heuristic scheme

(due to Powell), which tries to find a few good directions along narrow valleys instead of N necessarily conjugate directions. This is the method that was implemented by [Press88], which we use for our optimization problem.

Chapter 6

Results and Conclusion

6.0. Tracking System

Prerequisite: Precise tracking system, with linear behavior over the whole working volume.

The calibration procedures described in the previous chapters were all tested and evaluated with an optical tracker system called DTrack, which is produced by the company Advanced Realtime Tracking [Art01]. This system was chosen for the evaluation of the calibration procedures because it features linear behavior over the whole working volume, high update rates of up to 60 Hz, small system lag of about 20-40 ms, and high overall precision. The manufacturer gives the following example for the accuracy of a 4 Camera tracking system, which resembles our test setup:

In this setup a central measurement volume is defined as that part of the total measurement volume that is seen by all four cameras simultaneously and that covers most of the tracked persons movements. A typical result for the tracking of a person's head position and orientation (the person is wearing a head mounted display) is given in the table below:

Possible accuracy (<i>conditions like described below, ideal measurement</i>):	Target position	Target orientation
Accuracy absolute (<i>RMS over central measurement volume</i>)	250 μm	0.12 deg
Repeatability (<i>standard deviation</i>)	60 μm	0.03 deg
Maximum error (<i>calculated, in central measurement volume</i>)	900 μm	0.4 deg
Noise	30 μm	0.015 deg

Table 1 Accuracy possible with the DTrack optical tracking system.

The following conditions apply:

- camera position: in the 4 upper corners of a cube, ca. 6m x 6m x 4m
- rigid body: 5 spherical markers of 30 mm diameter
- ca. 200 mm size of the rigid body
- central measurement volume: cube of ca. 3.5m x 3.5m x 1.5m, located in the center of the 6x6x4m-cube and in 1m height above the floor
- ideal measurement: target in central measurement volume, seen by all tracking cameras, no occlusions

6.1. Test setup and evaluation of stylus calibration

The stylus calibration procedure was tested with the ART optical tracker. A standard target was mounted on a stylus (Figure 29) with a rather sharp tip, so that the tip would not easily move from its fixed position on a table.



Figure 29
The stylus with mounted target for optical tracking.

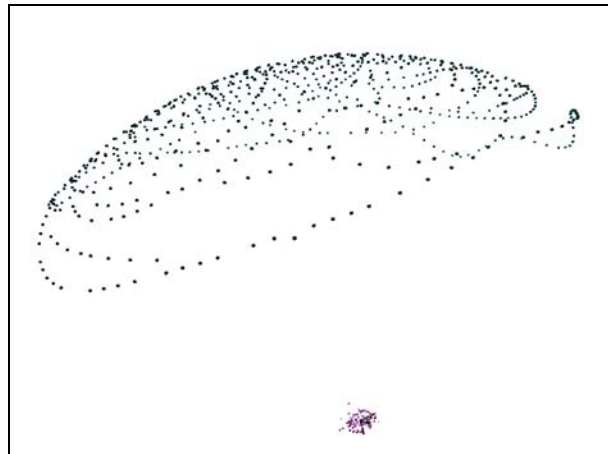


Figure 30
Point cloud sampled during the stylus calibration procedure.

The procedure was repeated five times. For each run of the procedure 500 or more samples were taken. The sample filter, which imposes a minimum distance between any two samples, was set to 5 mm. This particular

combination of settings forces the user to cover a great portion of the part of the sphere she can cover within the given physical limits (Figure 30). The average acquisition time was less than 30 seconds.

The standard deviation within one calibration is about 6 mm on average. The metric for the standard deviation is the distance between the calculated position of the hotspot for one 6DOF sample we took, i.e. the optimized offset is transformed from stylus space to world space by the sampled translation and rotation, and the average of all calculated positions of the hotspot, i.e. the “mean” hotspot. Figure 31 shows the error distribution from two different views. The sphere in the center of the two pictures depicts the “mean” hotspot of the stylus. The point cloud around the “mean” hotspot visualizes the “real” hotspots for each taken sample. The central sphere has a diameter of 1 mm, which is also the scale of the ruler.

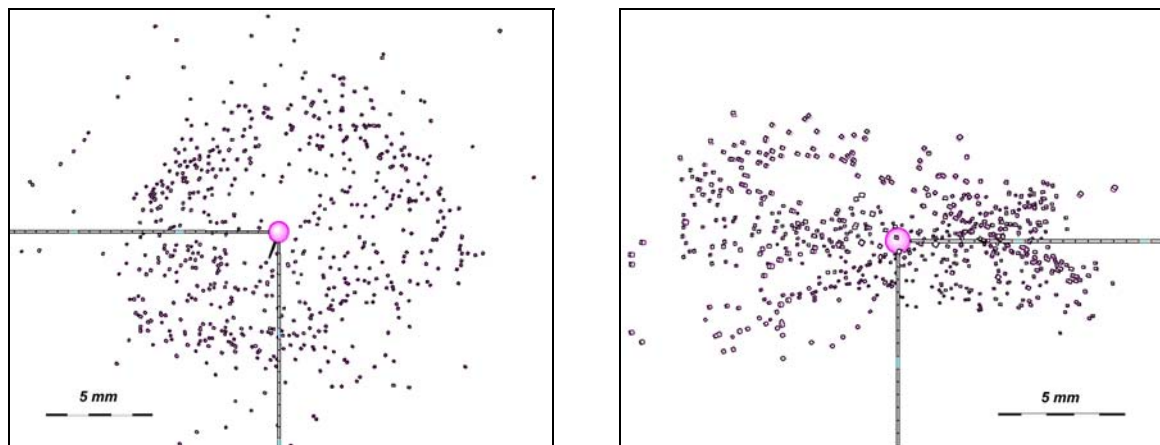


Figure 31 Distribution of hotspot positions calculated from optimized offset and sampled positions/orientations depicted from two different viewing angles.

The standard deviation of the offsets vectors we retrieved from the five test runs was 1.354 mm and the standard deviation of the length of these offset vectors of 1.202 mm was in the same league. This means that we get a very good result for repeatability of the stylus calibration. A test for the accuracy of the resulting offset vector was performed, by activating the calculated offset vector, i.e. the position that is associated with the stylus and delivered to the Studierstube system by the OpenTracker framework now indicates the actual position of the stylus' tip. We now performed the same task as for the calibration itself, i.e. moving the stylus around the steady stylus' tip. The sampled data is shown in Figure 32. Ideally all samples should converge in one point, since the stylus' tip is supposed to be steady. What we see is a small point cloud, which gets rather dense around the calculated mean of all sampled points. The standard

deviation of 1.288 mm confirms this observation and shows the good accuracy of the retrieved styluses offset.

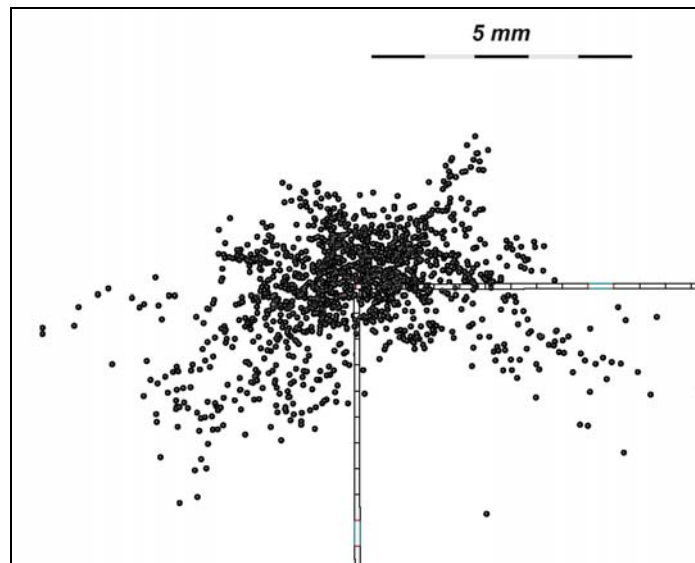


Figure 32 Distribution of sampled hotspot positions after applied registration.

6.2. Test setup and evaluation of HMD calibration

For the test setup of the HMD calibration procedure we used the aforementioned ART optical tracking system with four cameras. One rigid body was mounted on the HMD and a second was fixed to the stylus.

The user was guided by the distance constraint described in section 4.2.2, which indicated the right distance between the stylus hotspot and the head tracker (about 80 cm for 'far' markers and 35 cm for 'near' markers). The error threshold for the procedure was set to 0.75% of the display resolution for the maximum RMS error in image space. For the HMD we used, which has a resolution of 800 by 600 pixels and a horizontal FOV of about 32 degrees, this means that we tolerate a RMS horizontal angular error of 0.24 degrees. This translates to a positional error of 2.9 mm at about arms length away from the user, which also marks the border of the ideal working volume for this calibration method. Since only samples from within the volume, limited by the user's arm length, are taken into account when calculating the viewing parameters, the registration is best within this volume.

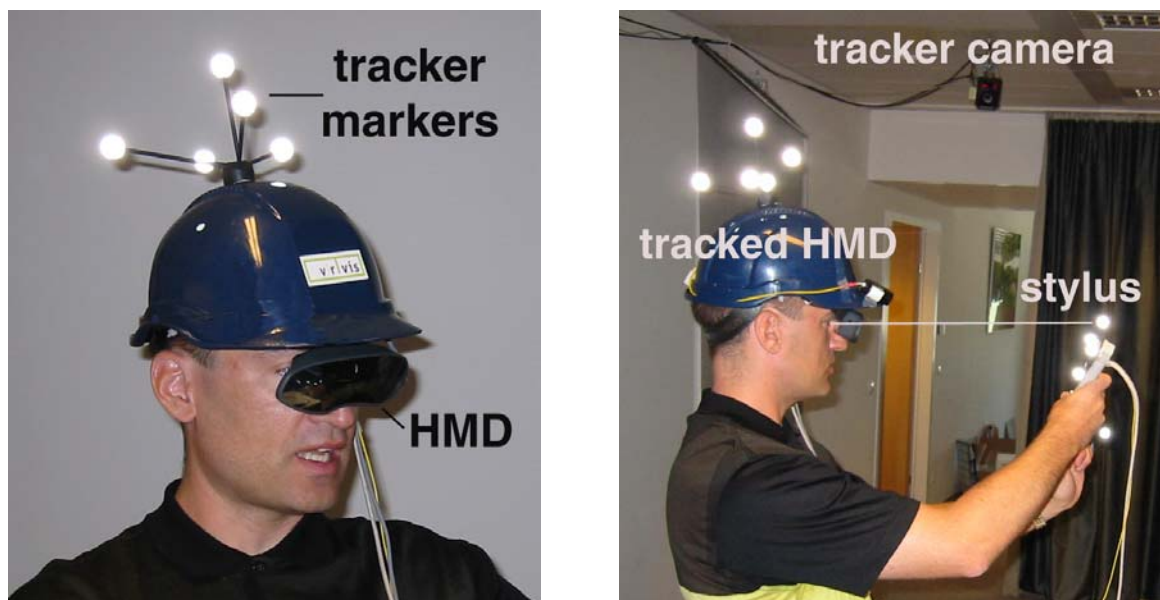


Figure 33 For better stability, the HMD is mounted in a helmet; the rigid body tracked by the optical tracker is mounted on the helmet (left). The user aligns the hotspot of the stylus with the virtual marker displayed via the HMD (right).

Since the acquisition of the samples and the calculation of the viewing parameters for each eye are independent, the user had to repeat only the procedure for the eye, for which no registration within the given threshold was possible. The time it took on average to calibrate one eye (eight markers have to be aligned with the stylus hotspot) was about one minute, translating to a time of about two minutes for the whole calibration procedure, including the calibration steps that had to be repeated. The computational time for the calculation and optimization of the viewing parameters of fewer than one second seems not relevant, when looking at the whole procedure.

For the evaluation of the quality of the registration we calculated the RMS error in image space, which also served as threshold parameter, as mentioned above. So we calculated the distances between the known positions of the virtual markers in image space (reference points) and the sampled 3D positions associated with these markers projected to image space using the calculated viewing parameters (the resulting registration). An example is given in Figure 34. Figure 35 shows the relative distances (errors) between sampled points and reference points in image space. As stated in chapter 4.2.2 we perform two optimization steps. The first takes all sampled positions into account, whereas the second discards the two samples which show the largest error. Since right and left eye calibrations are independent, we took the data from all 10 calibrations (left and right combined) to compile the following table.

	Worst maximum single error	Best maximum single error	Average maximum single error	Worst RMS error	Best RMS error	Average RMS error
After 1st optimization	2.645	0.640	1.579	1.468	0.512	0.992
After 2nd optimization	1.136	0.290	0.686	0.746	0.208	0.483

Table 2 Comparison of single and RMS error for 1st and 2nd optimization step. The errors are given in percentage of image space resolution.

What we see in Table 2 is that after the second optimization step the errors are reduced by about 50%. The results achieved by each single run of the calibration procedure show quite large divergences. This is of course due to the

fact that we rely heavily on the user input data. But by imposing the aforementioned threshold, we nevertheless achieve an average RMS error of 0.48%, which translates to an average angular error of 0.155 degrees or a positional error of 1.9 mm at about arms length away from the user.

We also compared the position of the virtual and the physical tip of the stylus while wearing the registered HMD, which is also described in section 4.2.2 as “visual control of the achieved registration”. Thereby we found that this practical experiment confirms the above-discussed numerical results, which basically only measure the ability of the algorithm to find a good solution for the provided set of data samples, though it is hardly possible to exactly quantify the real registration error.

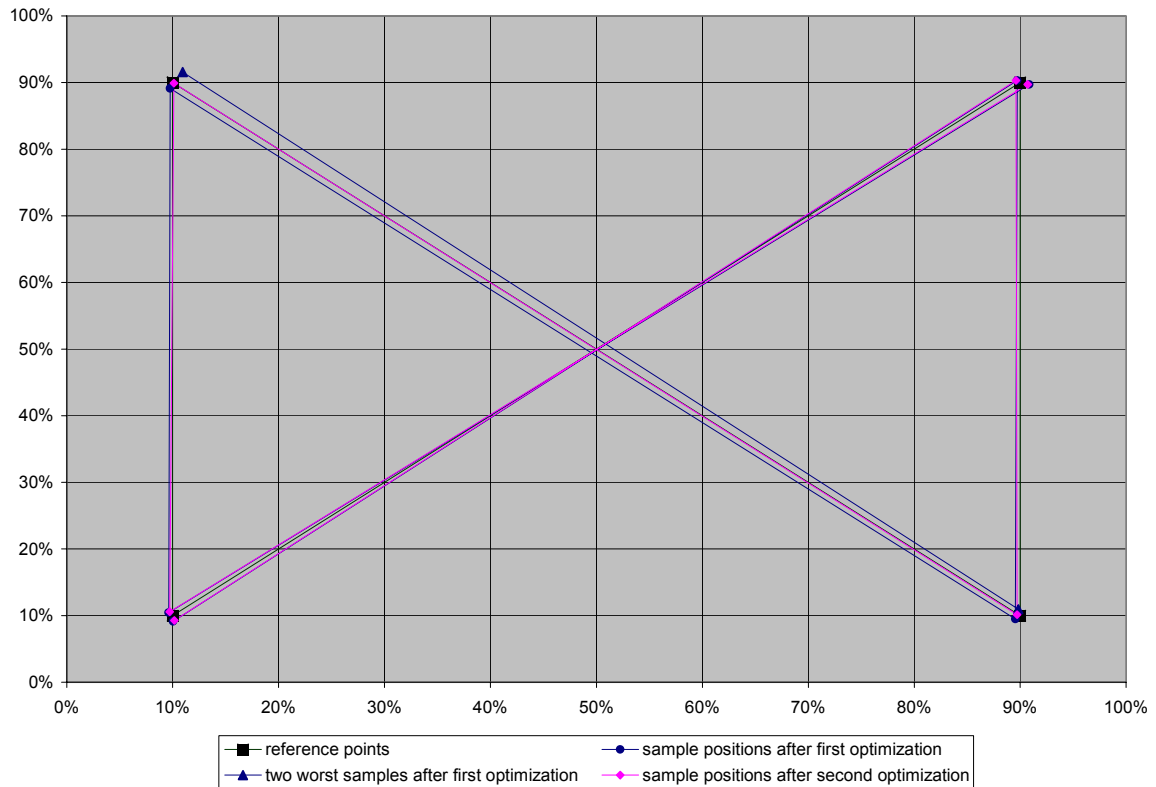


Figure 34 Typical distribution of the sampled 3D points after they have been projected onto the image plane using the viewing parameters calculated during the 1st and 2nd optimization step. (Note: The lines are used for better visualization of the positions of the projected sample points.)

The first real test for the accuracy this calibration procedure can achieve will be the Augmented Reality Aided Surgery (ARAS) project. This project’s goal is to

display 3D pre-operative data for the surgeon via HMD in the operating theater. Therefore a quick and accurate calibration procedure is mandatory.

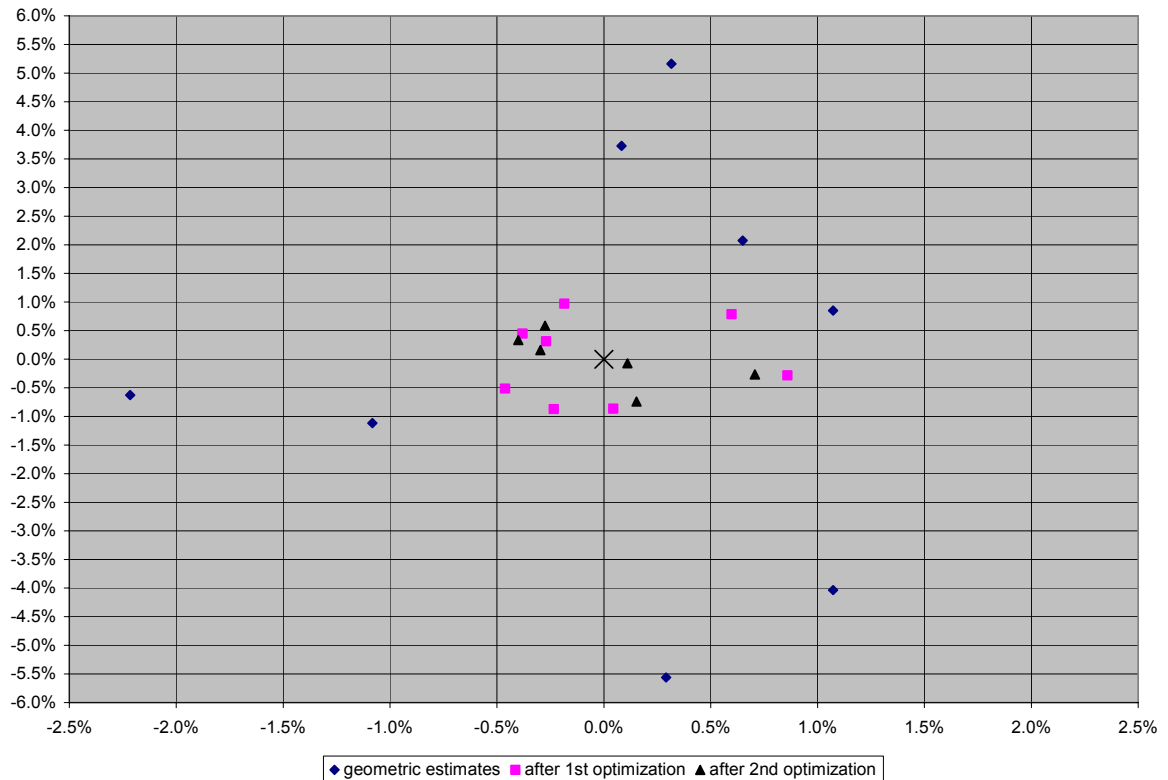


Figure 35 Relative error of sampled points in image space.

To have a measure for the repeatability of the procedure we compared the eye distances calculated during separate runs of the calibration procedure. The results are given in the table below.

Eye distance	66.58 mm	68.31 mm	66.39 mm	71.57 mm	69.20 mm
---------------------	----------	----------	----------	----------	----------

Table 3 Comparison of eye distance of one user, calculated in multiple runs of the calibration process for both eyes

This translates to a standard deviation of the calculated eye distances for one user of 2.122 mm.

The precision of our method still depends heavily on the input data, but when executed carefully it delivers good accuracy with small time exposure and without the need for additional hardware for an existing Studierstube setup.

6.3. Test setup and evaluation of Projection calibration

As test setup we used a virtual table (BARCO BARON), a back-projection desk specifically designed for VR applications. It has a resolution of 1024 by 768 pixels and features active stereo. Therefore the user has to wear shutter glasses to perceive the rendered scene in stereo. As tracking device we once again used the ART optical tracker. One rigid body was mounted on the shutter glasses to realize head tracking and a second was fixed to the stylus, which again served as sampling device.



Figure 36 Virtual Table (left); User wearing tracked shutter glasses for head tracking and controlling an application with the PIP (right)

The calibration of the projection plane was performed five times, to get a measure for the repeatability of the procedure. The acquisition time to retrieve the sample points was in the range of 15 seconds. The results are presented in the table below.

For the calibration of the eye offsets, we set the angular constraint to be 25 degrees, the distance constraint to be 25 cm and the constraint for the intersection of the rays to be 1 cm. This particular set of parameters was chosen to impose a tight limit on the input data and therefore elevate the chances of a good registration result. The eyepoint calibration was also repeated five times. The time it took on average to calibrate one eye (four markers have to be aligned with the stylus hotspot) was about one minute, translating to a time of

about two minutes for the whole calibration procedure. Note that since we impose only constraints and no thresholds on this task, it is not necessary to repeat a calibration step, but it took the users longer to align the hotspot of the stylus and the displayed marker, since the input constraints were harder to fulfill than in the case of the HMD calibration.

	Plane position (center)	Plane orientation	Width	Height
Standard deviation	1.588 mm	0.0276 degrees	3.105 mm	3.35 mm

Table 4 Results of the plane registration portion of the projection device calibration, giving a measure for the repeatability of the plane parameter registration.

We took the standard deviation of the resulting eye distances as measure for the repeatability of the procedure. The results are given in the table below.

Eye distance	66.48 mm	69.11 mm	67.42 mm	70.34 mm	66.97 mm
-------------------------	----------	----------	----------	----------	----------

Table 5 Comparison of eye distance of one user, calculated in multiple runs of the calibration process for both eyes

This translates to a standard deviation of 1.612 mm for the calculated eye distance of one user.

The precision of our method again depends on the input data, but by imposing strict constraints to the input data we achieve quite good accuracy and repeatability.

6.4. Conclusion

As we showed in this chapter, good results for the repeatability and accuracy of the stylus calibration were achieved. For the HMD calibration we also got good results for solving of the optimization problem and satisfying results for the repeatability of the whole procedure. Since the accuracy of the achieved registration for the optical see-through HMD was only verified subjectively, it would be interesting to develop a method for measuring the ‘objective’ registration error. The calibration of the projection table was very accurate and repeatable regarding the calibration of the projection plane. The calibration of the eye offsets again showed a good repeatability. Its accuracy was also satisfying but was again verified only subjectively. The accuracy of all calibration procedures is dependent on the quality of the input data gathered by the user. By imposing constraints on the input data and evaluating quality criteria for the achieved registrations this major factor of error was minimized effectively.

The users testing our calibration procedures were able to execute any calibration task in short time (at most a few minutes), after they were given a short explanation of the task they had to perform. The response of the users was quite positive in respect to the ease of use and accuracy of the results of the calibration procedures. The real field test will take place, when all users of Studierstube utilize the whole calibration suite.

Chapter 7

Future Work

With the work presented in this thesis, static registration, the basis for correct overall registration, has been implemented for the Studierstube Augmented Reality environment. Possible further improvements of the existing implementation and the Studierstube system in particular regarding registration are discussed in this chapter.

7.1. Additional constraints and thresholds

A major contributing factor for the precision of the procedures is obviously the precision of the user's input samples, as stated in the previous chapters. Therefore input constraints with visual feedback (see chapter 5.2.2) were implemented to both guide the user through the process and enforce the acquisition of valid samples. To further improve the guidance of the user and to ensure the best achievable accuracy during the sampling stage of the calibration procedures additional, mostly geometric constraints could be imposed on the input data. When doing so it has to be observed not to ask too much of the user, i.e. it should always be clear for the user, what constraint she has to comply with.

Since precise or even valid input data cannot be guaranteed, despite the before mentioned input constraints, different accuracy thresholds are currently implemented. If such an accuracy threshold is exceeded the user is asked to repeat the last step of the calibration process. Firstly the existing thresholds could be further tuned to balance the numerical accuracy of the registration and the possibility a user has to repeat a calibration step better. Secondly additional thresholds could be implemented, to add further control to the accuracy that should be achieved by the particular registration procedure.

Another possibility for improving the quality of the input data would be to fix the stylus in a certain place, so that the user only has to move her head to align

virtual and real markers. This would eliminate any jitter introduced by unintentional hand movement.

7.2. Faster HMD calibration

At the moment a user has to sample eight points per eye to achieve good results with the HMD calibration procedure described in this thesis (see section 4.2.2). To further reduce the time and effort needed to perform an HMD calibration, we could use the information gathered during previous HMD calibrations a user has performed, e.g. the eye distance of the user, to reduce the number of viewing parameters we have to calculate. Hence it should be possible to develop a method, which needs less user interaction – less input samples – and therefore should be quicker to perform.

7.3. Dynamic registration

Static errors, which we dealt with in this thesis, are the ones that cause registration errors even when the user's viewpoint and the objects in the environment remain completely still, whereas dynamic errors have no effect until either the viewpoint or the objects begin moving. Dynamic errors occur because of system delays, or lags. The end-to-end system delay is defined as the time difference between the moment that the tracking system measures the position and orientation of the viewpoint to the moment when the generated images corresponding to that position and orientation appear in the displays. To the user, the virtual objects appear to "swim around" and "lag behind" the real objects. For current HMD-based systems, dynamic errors are the largest contributors to registration errors.

Hence the next step to making the Studierstube Augmented Reality environment a believable experience is to implement dynamic registration.

Methods used to reduce dynamic registration errors fall under these main categories [Azuma97a]:

- Reduce system lag
- Reduce apparent lag
- Predict future locations

The last method, which is the most promising one, is to predict the future viewpoint and object locations. If the future locations are known, the scene can be rendered with these future locations, rather than the measured locations.

Then when the scene finally appears, the viewpoints and objects have moved to the predicted locations, and the graphic images are correct at the time they are viewed. For short system delays (under ~ 80 ms), prediction has been shown to reduce dynamic errors by up to an order of magnitude [Azuma94]. It will be our future work, to test and implement methods for prediction.

Bibliography and References

- Adler01 Adler, S. et al., Extensible stylesheet language (XSL) 1.0.
<http://www.w3.org/TR/xsl/>.
- Apache99 Apache. Xerces XML parser.
<http://xml.apache.org/xerces-c/index.html>.
- Art01 Advanced Realtime Tracking GmbH, A.R.T. Infrared
Tracking System. Information brochure, April 2001.
- Azuma93 Azuma, Ronald. Tracking Requirements for Augmented
Reality. *Communications of the ACM* 36, 7 (July 1993), 50-
51.
- Azuma94 Azuma, Ronald, and Gary Bishop. Improving Static and
Dynamic Registration in a See-Through HMD. *Proceedings
of SIGGRAPH '94* (Orlando, FL, 24-29 July 1994). In
Computer Graphics, Annual Conference Series, 1994, 197-
204.
- Azuma97a Azuma, Ronald T.. A Survey of Augmented Reality.
Presence: Teleoperators and Virtual Environments 6, 4
(August 1997), 355-385.
- Azuma97b Azuma, Ronald T. Course notes on "Registration" and
"Correcting for Dynamic Error" from Course Notes #30:
Making Direct Manipulation Work in Virtual Reality. ACM
SIGGRAPH '97 (Los Angeles, CA, 3-8 August 1997).
- Bajura95 Bajura, M., and Neumann, U. Dynamic Registration
Correction in Augmented-Reality Systems. *VRAIS'95*,
1995.

- Bajura97 Bajura, Michael A.: Merging Real and Virtual Environments with Video See-Through Head-Mounted Displays. Dissertation, UNC, 1997.
- Billinghurst99 Billinghurst, Mark, Kato, Hirokazu: Collaborative Mixed Reality. Proceedings of the First International Symposium on Mixed Reality, Yokohama, Japan, March 1999, pp. 261–284.
- BouJou02 <http://www.2d3.com/2d3/products/features.shtml>
- Bray00 Bray, T., Paoli, J., Sperberg-McQueenC. et al. Extensible Markup Language (XML) 1.0. <http://www.w3.org/TR/REC-xml/>, 2000.
- Brent73 Brent, R.P. 1973, *Algorithms for Minimization without Derivatives* (Englewood Cliffs, NJ: Prentice-Hall)
- Czernuszenko98 Czernuszenko, Marek, Daniel Sandin, Thomas DeFanti: Line of Sight Method for Tracker Calibration in Projection-Based VR. Proceedings of the 2nd International Immersive Projection Technology Workshop, Iowa State University, May, 1998.
- Deering92 Deering, Michael. High Resolution Virtual Reality. *Proceedings of SIGGRAPH '92* (Chicago, IL, 26-31 July 1992). In *Computer Graphics* 26, 2 (July 1992), 195-202.
- Doenges85 Doenges, Peter K. Overview of Computer Image Generation in Visual Simulation. *SIGGRAPH '85 Course Notes #14 on High Performance Image Generation Systems* (San Francisco, CA, 22 July 1985).
- Fuhrmann00 Fuhrmann, A., Schmalstieg, D. and Purgathofer, W. Practical Calibration Procedures for Augmented Reality. Proceedings of the 6th Eurographics Workshop on Virtual Environments, Amsterdam, Netherlands, June 2000.
- Gibson02 Gibson, S., Cook, J., Howard, T.L.J., Hubbard, R.J., and Oram, D., Accurate Camera Calibration for Off-line, Video-Based Augmented Reality. IEEE and ACM International Symposium on Mixed and Augmented Reality (ISMAR 2002), Darmstadt, Germany, September 2002.

- Hoff00 Hoff, William A., Vincent, Tyrone: Analysis of Head Pose Accuracy in Augmented Reality. *IEEE Transactions on Computer Graphics and Visualization*, vol. 6, no. 4, 2000.
- Holloway95 Holloway, Richard. Registration Errors in Augmented Reality. Ph.D. dissertation. UNC Chapel Hill Department of Computer Science technical report TR95-016 (August 1995).
- Jain89 Jain, Anil K. Fundamentals of Digital Image Processing. Prentice Hall (1989). ISBN 0-13-336165-9.
- Janin93 Janin, Adam L., David W. Mizell, and Thomas P. Caudell. Calibration of Head-Mounted Displays for Augmented Reality Applications. *Proceedings of IEEE VRAIS '93* (Seattle, WA, 18-22 September 1993), 246-255.
- Kato99 Kato, H. and Billinghurst, M.,. Marker tracking and hmd calibration for a video-based augmented reality conferencing system. In *Proceedings of the 2nd IEEE and ACM International Workshop on Augmented Reality (IWAR'99)*, San Francisco. IEEE, October 1999.
- Kindratenko99 Kindratenko, Volodymyr: Calibration of electromagnetic tracking devices. *Virtual Reality: Research, Development, and Applications (The VRS Journal)*, vol. 4, 1999, pp. 139–150.
- Livingston97 Livingston, Mark, State, Andrei: Magnetic Tracker Calibration for Improved Augmented Reality Registration. *Presence*, vol. 6, 1997, pp. 532–546.
- McGarrrity99 McGarrrity, Erin, Tuceryan, Mihran: A Method for Calibrating See-through Head-mounted Displays for AR. *Proceedings of the 2nd IEEE International Workshop on Augmented Reality (IWAR 99)*, San Francisco, CA, October 1999.
- Milgram94a Milgram, Paul, and Fumio Kishino. A Taxonomy of Mixed Reality Virtual Displays. *IEICE Transactions on Information and Systems E77-D*, 9 (September 1994), 1321-1329.

- Milgram94b Milgram, Paul, Haruo Takemura, Akira Utsumi, and Fumio Kishino. Augmented Reality: A Class of Displays on the Reality-Virtuality Continuum. *SPIE Proceedings volume 2351: Telemanipulator and Telepresence Technologies* (Boston, MA, 31 October - 4 November 1994), 282-292.
- Oishi96 Oishi, Takashi and Susumu Tachi. Methods to Calibrate Projection Transformation Parameters for See-Through Head-Mounted Displays. *Presence: Teleoperators and Virtual Environments* 5, 1 (Winter 1996), 122-135.
- Pausch92 Pausch, Randy, Thomas Crea, and Matthew Conway. A Literature Survey for Virtual Environments: Military Flight Simulator Visual Systems and Simulator Sickness. *Presence: Teleoperators and Virtual Environments* 1, 3 (Summer 1992), 344-363.
- Press88 Press, W., Flannery, B., Teukolsky, S., and Vetterling, W.. Numerical Recipes in C. Cambridge University Press, 1988.
- Reitmayroo Reitmayr, Gerhard, Dieter Schmalstieg: OpenTracker — An Open Software Architecture for Reconfigurable Tracking based on XML. Poster, IEEE Virtual Reality 2001, Yokohama, Japan, March 2001. Extended version available as technical report TR-186-2-00-18, Institute of Computer Graphics and Algorithms, Vienna University of Technology, Austria, June 2000.
- Rekimoto98 Rekimoto, Jun: Matrix: A Realtime Object Identification and Registration Method for Augmented Reality. Proceedings of Asia Pacific Computer Human Interaction 1998, Japan, 1998.
- Schmalstieg00 Schmalstieg, Dieter, A. Fuhrmann, G. Hesina, Zs. Szalavari, L. M. Encarnação, M. Gervautz, W. Purgathofer: *The Studierstube Augmented Reality Project*. TR-186-2-00-22, Vienna University of Technology, December 2000.
- Strauss92 Strauss, P., Carey, R.. An object oriented 3D graphics toolkit, Proc. SIGGRAPH '92, pp. 341-347, 1992.

- Summers99 Summers, Valerie A., Booth, Kellogg S., Calvert, Tom , Graham, Evan, MacKenzie, Christine L.: Calibration For Augmented Reality Experimental Testbeds. Proceedings of the 1999 symposium on Interactive 3D graphics, Atlanta, GA, April 1999, pp. 155–162.
- Szalavári97 Szalavári, Zs., M. Gervautz, M. The Personal InteractionPanel - A Two-handed Interface for Augmented Reality.Proc. EUROGRAPHICS 97, Budapest, Hungary, 335-346,1997.
- Szalavári99 Szalavári, Zs., The Personal InteractionPanel - A Two-handed Interface for Augmented Reality. Ph.D. dissertation. Vienna University of Technology, Vienna, Austria, Institute of Computer Graphics and Algorithms (September 1999).
- Tuceryan95 Tuceryan, M., Greer, D., Whitaker, R., Breen, D., Crampton, C.,Rose, E. and Ahlers, K.. Calibration requirements and procedures for a monitor-based augmented reality system. *IEEE Transactions on Visualization and Computer Graphics*, 1(3):255–273, September 1995.
- Tuceryan00 Tuceryan, Mihran, Navab, Nassir: Single point active alignment method (SPAAM) for optical see-through HMD calibration for AR. Proceedings of the IEEE and ACM International Symposium on Augmented Reality, Munich, Germany, October, 2000, pp. 149–158.
- Welch78 Welch, Robert B. Perceptual Modification: Adapting to Altered Sensory Environments. Academic Press (1978). ISBN 0-12-741850-4.
- Wernecke94 Wernecke, J,. The Inventor Mentor: Programming Object Oriented 3d Graphics With Open Inventor, Release 2. Addison-Wesley, 1994.
- Whitaker95 Whitaker, Ross T., Crampton, Chris, Breen, David E., Tuceryan, Mihran, Rose, Eric: Object Calibration for Augmented Reality. Proceedings of Eurographics'95, Maastricht, Netherlands, 1995, pp. 15–28.