Dissertation

Studierstube: A Collaborative Virtual Environment for Scientific Visualization

ausgeführt zum Zwecke der Erlangung des akademischen Grades
eines Doktors der technischen Wissenschaften unter der Leitung von

A.o. Prof. Dipl.-Ing. Dr. Michael Gervautz
Institut Nr.186
Institut für Computergraphik

eingereicht an der Technischen Universität Wien
Technisch-Naturwissenschaftliche Fakultät

von

Dipl.-Ing. Anton L. Fuhrmann
Wahlberggasse 2, A-1140 Wien
Matr.-Nr. 8525331

Acknowledgements

Special thanks to my mentor Michael Gervautz, who not only was the first person to suggest that I started this thesis, but also arranged for the necessary means. His sometimes barbed comments where what kept me on track during the previous three years. His corrections of my papers ("..apply the Gervautz operator one more time...") always hit their weakest spot, thereby improving the papers and me.

Thanks also to Meister Eduard Gröller, without whose psychological and scientific support this thesis would have been impossible, to Dieter Schmalstieg, Robert Tobler and Helwig Löffelmann for their help and suggestions and to Gerd Hesina (Figure 31), Markus Krutz, Rainer Splechtna (Figure 19), Hermann Wurnig (Figure 6, right) and Andreas Zajic for their implementation work.

Thanks also to all the other guys at the institute, who created an atmosphere of creative chaos which I would not like to have missed for all the rendering power in the world.

This work has been supported by the Austrian Science Foundation (FWF) under project no. P-12074-MAT.

Table of Contents

1 Kurzfassung (German Abstract) 5
2 Abstract II
3 Problem Statement 4
4 Proposed solution 7
5 Structure of the Thesis 10
6 Introduction to Studierstube 13
7 Related Work 17
8 Workspace Concept and Design 21
8.1 Hardware Setup 22
8.2 Design Analysis of the Workspace 25
8.3 User Interaction in Workspace 28
8.4 Input and Output Contexts 31
8.5 Multi-user Aware Applications 33
8.6 Implementation 38
9 Fast Calibration for Augmented Reality 46
9.1 Overview 46
9.2 Introduction 47
9.3 Registration in Augmented Reality 49
9.4 Previous Work 51
9.5 Requirements 52
9.6 Calibration Procedure 54
9.7 Fast Calibration 58
9.8 Distortion Compensation 59
9.9 Registration of Virtual to Real Objects 61
9.10 Implementation Details 63
9.11 Results 65
10 Occlusion in Collaborative Augmented Environments 68
10.1 Overview 68
10.2 Introduction 68
10.3 Requirements for Occlusion in "Studierstube" 74
10.4 Occluding with Phantoms 75
10.5 Implementation 81
11 Studierstube as a Frontend for Scientific Visualization 88
11.1 Overview 88

11.2 Studierstube/AVS Interface 88
11.3 Collaborative Visualization 90
11.4 Visualization of Dynamical Systems 93
12 Real-Time Techniques For 3D Flow Visualization 95
12.1 Overview 95
12.2 Introduction and Motivation 95
12.3 Related Work 96
12.4 Dashtubes: Streamlines with Animated Opacity 97
12.5 Adaptive Texture-Mapping 101
12.6 Streamline Placement 106
12.7 Focussing and Context 107
12.8 Implementation 112
13 Results 119
13.1 Scientific Visualization in Studierstube 119
13.2 Studierstube as a generic user interface 123
13.3 Scientific Collaborations 131
14 Evaluation 138
14.1 Studierstube Workspace 138
14.2 Collaborative Scientific Visualization 141
14.3 Unresolved problems 142
15 Conclusion 145
16 References 146

1 Kurzfassung (German Abstract)

Diese Arbeit beschreibt den *Studierstube Workspace*, eine interaktive Arbeitsumgebung für kollaborative Visualisierung in *Augmented Reality*. Wir erarbeiten Konzepte für einen kollaborativen Arbeitsbereich, in dem mehrere Benutzer gleichzeitig verschiedene Anwendungen bedienen können. Das gemeinschaftliche Arbeiten der Benutzer innerhalb eines Raumes mit mehreren gleichzeitig ablaufenden Anwendungen eröffnet völlig neue Möglichkeiten der Kollaboration und Interaktion. Wir beschreiben, wie multiple Interaktionspfade zur Implementierung der zugrundeliegenden Mechanismen genutzt wurden, und berichten über Strategien und Erfahrungen im Umgang mit und der Entwicklung von Multi-Anwender Applikationen.

Augmented Reality besteht aus der Überlagerung von computergenerierten Bildern über die Wirklichkeit. Um die räumlichen Abhängigkeiten zwischen reellen und virtuellen Objekten – die sogenannte *Registrierung* – korrekt darstellen zu können, müssen diese Bilder unter Verwendung von Transformationen erstellt werden, die einen Punkt im virtuellen Raum genau auf seinen Konterpart in der Realität abbilden. Dazu präsentieren wir ein einfaches und schnelles Kalibrationsverfahren, welches keine zusätzlichen Meßgeräte oder komplizierte Prozeduren erfordert. Der Benutzer wird dabei interaktiv durch eine Reihe von einfachen Schritten geleitet, welche ihm eine Anpassung der Transformationen auf seinen Augenabstand und seine Kopfabmessungen erlauben.

Eine weitere Verbesserung der Registrierung wird durch unsere Methode zur Korrektur von Linsenverzerrungen erzielt. Wir benutzen dazu Standard OpenGL Hardware, welche eine Entzerrung in Echtzeit ermöglicht. Weiters wird eine Technik um bewegliche, vom Computer verfolgbare Gegenstände zu ihren virtuellen Repräsentationen zu registrieren.

Die Qualität und Konsistenz der virtuellen Umgebung hängt auch von der korrekten Verdeckung von virtuellen durch reelle Gegenstände und umgekehrt ab. Wir haben ein Verfahren entwickelt, das nicht nur für den Körper eines Benutzers und vom Computer verfolgbare Gegenstände anwendbar ist, sondern darüber hinaus noch

irritierende Effekte aufgrund mangelhafter Registrierung reduziert. Unser Verfahren basiert auf der simulierten Verdeckung von virtuellen Objekten durch virtuelle Repräsentationen realer Objekte. Benutzer werden durch kinematische Ketten simuliert, welche dann zur Verdeckung herangezogen werden können.

Wir demonstrieren die Gültigkeit unseres Gesamtkonzeptes der kollaborativen Visualisierung innerhalb einer virtuellen Umgebung durch die Einbindung eines kommerziellen wissenschaftlichen Visualisierungssystems in unsere Arbeitsumgebung. Mehrere Anwendungsbeispiele zur Visualisierung dynamischer Systeme illustrieren unser Konzept.

Um die speziellen Eigenschaften der virtuellen Realität in die Visualisierungsmöglichkeiten des Gesamtsystems einfließen zu lassen, haben wir animierte, strichlierte Strömungslinien entwickelt. Dazu präsentieren wir eine texture-mapping Technik, welche die Details entlang jeder Strömungslinie auch bei stark variierender Flußgeschwindigkeit konstant hält. Weiters wird gezeigt, wie man eine gleichmäßige Verteilung der Strömungslinien im Raum erzielt.

“Magische” Lupen und “magische” Kisten werden als spezielle Interaktionmethoden zur Erforschung dicht mit Strömungslinien gefüllter Volumina verwendet. Dadurch wird eine Überforderung des Benutzers mit visuellen Details vermieden.

Wir geben einen Überblick über die Anwendung von *Studierstube* in verschiedenen Bereichen. Weiters wird gezeigt, wie verschiedene andere Forschungsinstitutionen *Studierstube* verwenden.

Eine umfangreiche Evaluation der geleisteten Arbeit in Bezug auf ihre Anwendbarkeit auf wissenschaftliche Visualisierung und ihrer Eigenschaften als neuartiges Konzept für kollaboratives Arbeiten innerhalb der virtuellen Realität rundet die Ausführungen inhaltlich ab.

2 Abstract

This thesis describes *Studierstube Workspace*, an application framework for collaborative visualization in *Augmented Reality*. We develop a concept for a collaborative working environment that simultaneously supports multiple users as well as multiple applications and in multi-tasking. The co-presence of multiple users interacting with multiple concurrently executing applications opens up new possibilities in the field of collaborative work. We describe how multiplicity in the interaction paths is used to implement the necessary underlying mechanisms and report on strategies and experiences regarding the development of multi-user aware applications.

Augmented Reality overlays computer generated images over the real world. To correctly depict spatial relations between real and virtual objects – the so-called *registration* – these images have to be generated using transformations which correctly project a point in virtual space onto its corresponding point in the real world. We present a simple and fast calibration scheme, which does neither require additional instrumentation nor complicated procedures. This allows us to calibrate the virtual environment for specific users. The user is interactively guided through a series of simple initialization steps, which allows even inexperienced users to adapt the calibration to their eye distance and head geometry.

To further improve the registration between real and virtual objects we describe a method for correcting the distortions introduced by the camera lens in real-time using standard OpenGL hardware. A simple technique for registering tracked objects to their augmentations is also introduced.

The quality and consistency of the augmentation also depend on the correct occlusion of real objects by computer-generated objects and vice versa. We developed methods that are not only appropriate for a tracked users body and other real objects but also manage to reduce irritating artifacts due to misregistrations. Our technique is based on simulating the occlusion of virtual objects by a virtual representation of the real object. The user is modeled as kinematic chains of articulated solids which is used for occlusion. Registration and modeling

errors of this model are reduced by smoothing the border between virtual world and occluding real object.

We demonstrate the validity of our concept of collaborative visualization in a virtual environment with the integration of our virtual environment into a commercial visualization system and illustrate it with several visualizations of dynamical systems.

To integrate the properties of our environment into the process of scientific visualization we created animated, opacity-mapped streamlines as visualization icon for realtime 3D flow visualization. We present a texture mapping technique to keep the level of texture detail along a streamline nearly constant even when the velocity of the flow varies considerably and describe an algorithm which distributes the dashtubes evenly in space. Magic lenses and magic boxes are applied as interaction techniques for investigating volumes filled densely with streamlines without overwhelming the observer with visual detail.

We present applications of *Studierstube* in a number of different areas and collaborations with other research institutes already using *Studierstube*.

An extensive evaluation of the performed work regarding its applicability on scientific visualization and its properties as a new concept for collaborative work in virtual reality concludes the thesis.

3 Problem Statement

Data acquired by satellites, generated by supercomputer simulations or logged for documenting the transactions of the stock market can lead to datasets in the Terabyte range. Trying to understand underlying patterns using only the numerical output of such datasets is next to impossible. One of the main tools aiding human understanding of this flood of information produced by almost all aspects of the modern world is *Scientific Visualization*.

Computer graphics has been described in its early stages as "a solution looking for a problem". If not in anything else, at least it has found its problem in the area of *Scientific Visualization*.

In [Schroeder1996] Visualization is defined as:

> "*Visualization* is the process of exploring, transforming, and viewing data as images (or other sensory forms) to gain understanding and insight into the data."

Visualization is described not as a static result but as an ongoing process of data processing resulting not in an image but in an *understanding* of the data behind the image. Furthermore, [Schroeder1996] lists characteristica distinguishing visualization from its parent discipline computer graphics as:

> "1. The dimensionality of data is three dimensions or greater. Many well-known methods are available for data of two dimensions or less; visualization serves best when applied to data of higher dimensions.
>
> 2. Visualization concerns itself with data transformation. That is, information is repeatedly created and modified to enhance the meaning of data.
>
> 3. Visualization is naturally interactive, including the human directly in the process

of creating, transforming, and viewing data."

Thus, visualization is further defined as a highly interactive process working with high-dimensional data in an iterative process along the *Visualization Pipeline* as illustrated in Figure 1.

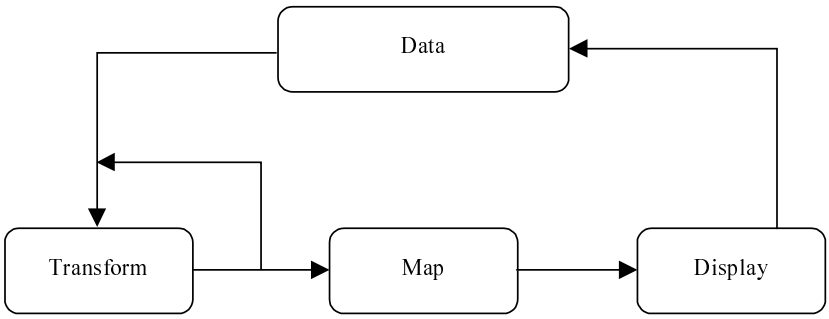


Figure 1: The visualization Pipeline (adapted from [Schroeder1996])

For complex visualizations, the process may require the expertise of more than one person. A possible scenario consists of one expert in scientific visualization performing adjustments and selections in the "map" stage – applying different visualization methods – and one or more specialists providing and analyzing the underlying datasets; for example stock market experts analyzing the performance of a portfolio. Another scenario would be a group of scientists collaborating on a project and preparing a visualization of the results for presentation to the public.

While all of the participating users have to share a common perception of the data, their different fields of expertise may make it necessary to customize their view of different properties of the visualization and/or customize their interfaces. A stockbroker may only want to be able to select which stocks to visualize in a selected time-frame, while the assisting visualization expert needs access to mapping parameters and the economist may change parameters of the underlying simulation model.

Summarizing the statements above, we conclude that a work environment for visualization should solve the following problems:

- Display of three dimensions or more, e.g. animations
- Realtime modification and display of data
- Highly interactive, complex manipulations possible
- Collaboration of multiple users
- Shared but individually customizable view of data

4 Proposed solution

The requirements of the previous section lead us directly to the concept of a shared virtual environment for scientific visualization:

The characteristic properties of a virtual environment are:

- Realtime 3D display
- Highly interactive
- Sophisticated 3D interaction

A shared virtual environment must provide the following additional features:

- Individual views and viewpoints
- Individual interfaces
- Collaboration and conflict handling techniques

Technical progress in recent years gives reason to believe that *Virtual Reality* (VR) has a good potential to become the user interface of the future. At the moment, VR applications are usually tailored to the needs of a very specific domain, such as a theme park ride or a virtual mock-up for design inspection. Like in current *Graphical User Interfaces* where a large variety of tasks – like drawing, word processing and communicating – can be executed and coordinated via a single, consistent interface at the same time, our system should support not a single, specialized interface customized for a single application but a new metaphor for general interaction.

We believe that *Augmented Reality* (AR), the less obtrusive cousin of VR where computer generated images are overlayed over reality, has a better chance to become a viable user interface for everyday application than a fully immersive *Virtual Environment*, where one can only perceive the synthetic images. It not only allows for direct social user interaction without mediating layers of software but also for the possibility of including objects of the real environment – tables, mirrors, architectural models etc.- into the interaction concept.

Collaboration should be supported not only in the form of a specialized application among others but by the very concept and setup of the virtual environment. Our basic concept includes the idea of a

shared work environment where many users are able to collaborate in a natural unhindered way. This idea is illustrated by Figure 2: a simple setup, incorporating virtual objects like the globe into a real setup – the table – and naturally incorporating social interaction into the environment.

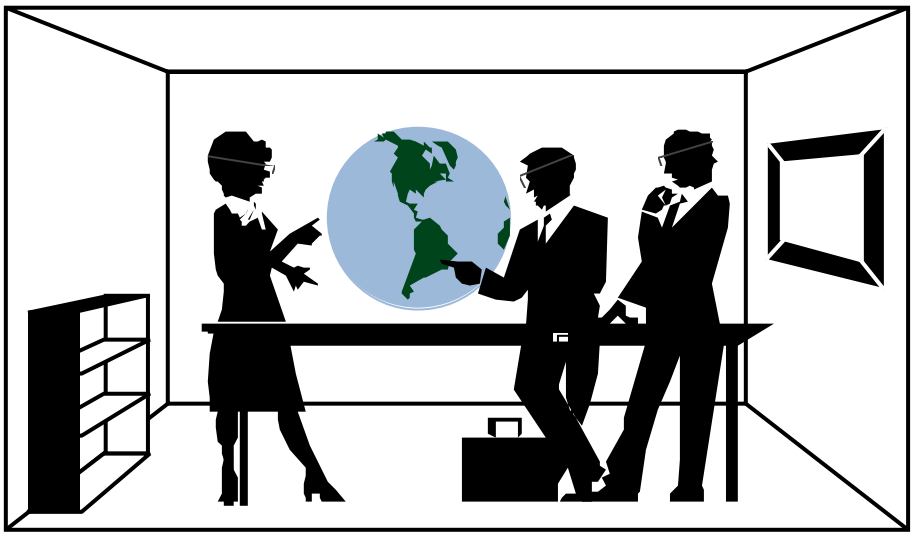


Figure 2: Illustration of basic concept: a shared virtual environment

The envisioned environment must satisfy these essential requirements:

– *Sophisticated Visualization Techniques*: Since our main target for the time is the application of our system in scientific visualization, the development of new, sophisticated visualization methods utilizing the special properties of virtual reality – realtime 3D display, fast changing viewpoints, highly interactive use – has to be a major objective in our research.

– *Collaboration*: The discussed system is designed for multiple concurrent users and must support collaborative work. While the co-presence of users in the same room allows many issues to be resolved using social protocols, the user interface design must incorporate appropriate "groupware" mechanisms so that technical concurrency issues as well as conflicts between users competing for an application can be resolved.

– *Multi-application*: A Workspace has very much in common with modern multi-tasking desktop GUIs. A user should be able to interact with multiple concurrently executing applications in turn. To further complicate requirements, multiple users must be able to work with any desired application – like for example a calculator shared by two persons in the same room – and even work with the same application (the same instance) at once, as in the case of architects working on different floors of a model building at the same time.
– *Augmentation*: It must be applicable to a shared augmented reality environment. This has two important consequences: Firstly, the organisational principles for the interface must be appropriate to 3D (and it is not trivial to transfer user interface elements from 2D). Secondly, there can be only one common three-dimensional space, which is shared among all participants and imposes some spatial constraints on the interface design.

Clearly, such a demanding set of features for a user interface requires a powerful user interface metaphor that is equally expressive in 3D as the desktop metaphor in 2D. We propose as our metaphor a ”Work-space” which acts as an interface layer between applications and user and can be customized by the users for their needs.

5 Structure of the Thesis

After an introduction into the basic concepts of our virtual environment Studierstube in chapter 6 we give an overview over relevant related work in chapter 7. Chapter 8 contains an in-depth explanation of the conceptual basis and resulting design decisions of the Studierstube Workspace, the framework of our system. Chapters 9 and 10 describe new approaches we developed to solve problems in the areas of calibration and occlusion, which we encountered in the course of our work. The integration of scientific visualization into our system is described in chapter 11, followed by a detailed description of new visualization techniques in chapter 12.

The thesis concludes with an evaluation of results and a short overview over resolved and unresolved aspects of our work in chapter 13.

This thesis contains material partially previously published in:

> A. Fuhrmann, H. Löffelmann, D. Schmalstieg: Collaborative Augmented Reality: Exploring Dynamical Systems. Proc. of IEEE Visualization 1997, pp. 459- 462, November 1997.
>
> A. Fuhrmann, H. Löffelmann, D. Schmalstieg, and M. Gervautz: Collaborative Visualization in Augmented Reality. IEEE Computer Graphics and Applications, 18(4): pp.54-59, July/August 1998.
>
> A. Fuhrmann and E. Gröller: Real-Time Techniques for 3D Flow Visualization. Proc. of IEEE Visualization 1998, pp. 305-312, November 1998.
>
> A. Fuhrmann, G. Hesina, F. Faure, and M. Gervautz: Occlusion in Collaborative Augmented Environments. Proceedings 5th Eurographics Workshop on Virtual Environments, ISBN 3-211-83347-1, pp. 179-190, Springer-Verlag, Wien, 1999.

A. Fuhrmann, D. Schmalstieg, W. Purgathofer: Fast Calibration for Augmented Reality. To appear in: Proceedings of ACM Virtual Reality Software & Technology '99 (VRST'99), short paper, London, December 20-22, 1999. Extended version available as technical report TR-186-2-99-16, Vienna University of Technology, 1999.

A. Fuhrmann, D. Schmalstieg: Concept and Implementation of a Collaborative Workspace for Augmented Reality. Technical report TR-186-2-99-04, Vienna University of Technology, 1999.

A. Fuhrmann, D. Schmalstieg: Multi-Context Augmented Reality. Technical report TR-186-2-99-14, Vienna University of Technology, 1999.

G. Hesina, D. Schmalstieg, A. Fuhrmann, W. Purgathofer: Distributed Open Inventor: A Practical Approach to Distributed 3D Graphics To appear in: Proceedings of ACM Virtual Reality Software & Technology '99 (VRST'99), London, December 20-22, 1999.

During the development of the thesis the following additional papers have been published:

D. Schmalstieg, A. L. Fuhrmann, M. Gervautz, and Zs. Szalavári: 'Studierstube' - An Environment for Collaboration in Augmented Reality'. 1996.

Z. Szalavári, M. Gervautz, A. Fuhrmann, and D. Schmalstieg. Augmented: Reality Enabled Collaborative Work in 'Studierstube'. EURO-VR '97, 1997.

A. Fuhrmann, D. Schmalstieg and M. Gervautz: Strolling Through Cyberspace with Your Hands in Your Pockets: Head-Directed Navigation Virtual Environments '98 (Proceedings of the 4th

EUROGRAPHICS Workshop on Virtual Environments), pp. 216-227, Stuttgart, Germany, June 16th-18th, Springer-Verlag, 1998.

D. Schmalstieg, A. Fuhrmann, Z. Szalavari, M. Gervautz: Studierstube - An Environment for Collaboration in Augmented Reality Extended abstract appeared in proceedings of Collaborative Virtual Environments '96, Nottingham, UK, Sep. 19-20, 1996. Full paper in: Virtual Reality - Systems, Development and Applications, Vol. 3, No. 1, pp. 37-49, 1998.

Z. Szalavari, D. Schmalstieg, A. Fuhrmann, M. Gervautz: Studierstube - An Environment for Collaboration in Augmented Reality, Virtual Reality: Research, Development & Applications, 1998

6 Introduction to Studierstube

We propose a system that allows multiple collaborating users to simultaneously study three-dimensional scientific visualizations in a „study room“ - German: „Studierstube“ (inspired by the classic play „Faust“ by J.W.Goethe).

Each participant wears an individually head-tracked see-through Head-Mounted Display (*HMD*) providing a stereoscopic real-time display (Figure 3).

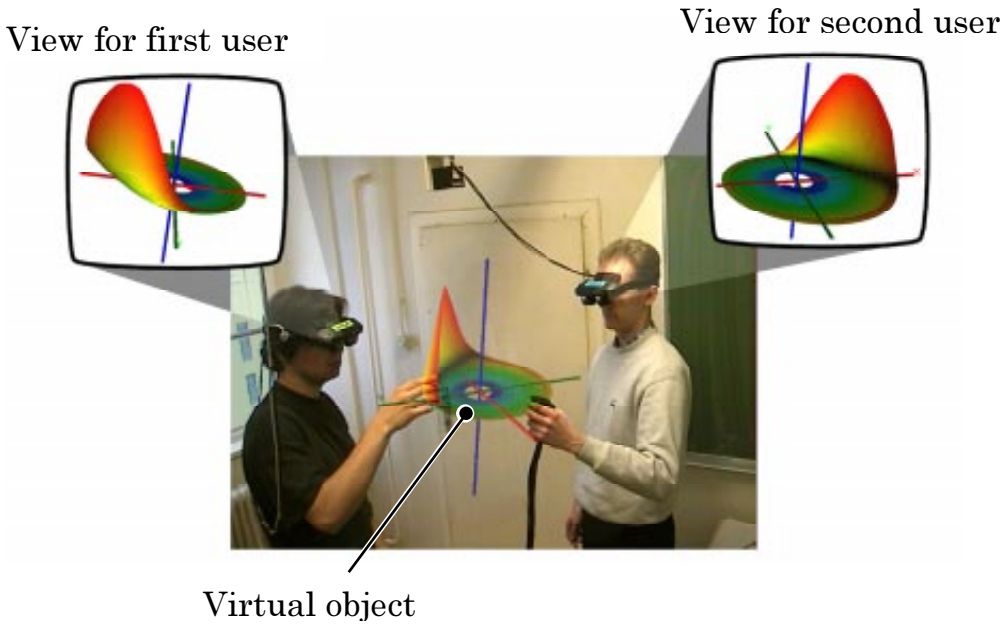


Figure 3: Illustration of Studierstube setup: A shared virtual space

The use of individual displays and head-tracking for each participant (for a description of the details of the hardware setup see section 8.1) allows stereoscopic, undistorted images to be presented to multiple users. There are no implicit constraints regarding the viewpoint. Unlike to other setups, users may stand face to face, which in combination with the see-through property of the HMDs allow users to see each other and interact directly. This also provides a direct feedback for han-eye coordination, since the user may directly perceive his hands instead of only seeing a simulation of their real position. The

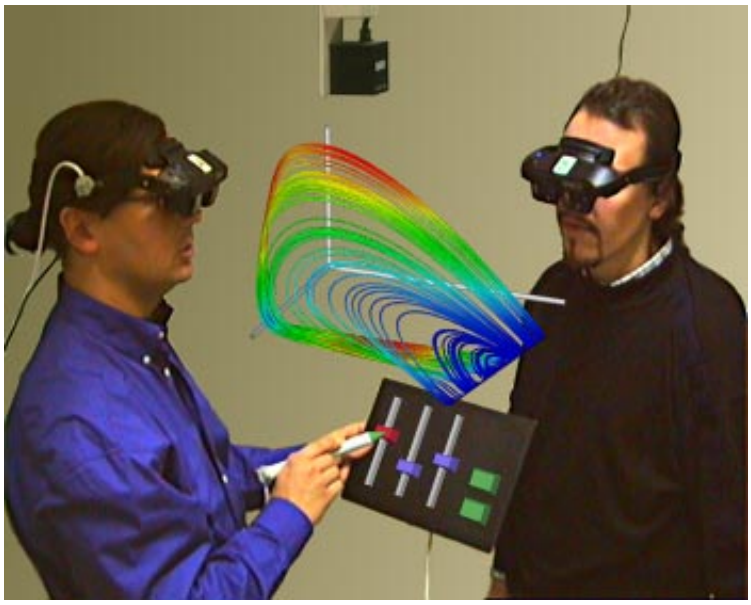


Figure 4: Adjusting the parameters of a visualization with the *Personal Interaction Panel* (PIP).

ability to directly see the surroundings also reduces the fear and disorientation new users perceive in a completely immersive setup. The possibility of bumping into obstacles is also reduced.

Rendering separate images for each user gives great flexibility in the choice of viewpoint, but also makes the rendering effort proportional to the number of users, whereas CAVE [Cruz-Neira1992] and (single-user) Workbench [Krüger1994] require only a constant rendering effort.

The defining features of Studierstube are:

– *Augmented props*: We exploit the capabilities of AR to construct a three-dimensional user interface needed for controlling the presentation and possibly the simulation by introducing tracked real-world objects that combine physical items and overlaid computer graphics, such as the Personal Interaction Panel (PIP) [Szalavári1997]. The PIP is used as a device for handling virtual objects (Figure 4) or as a cutting plane, but mainly as basis for conventional 2D user interface elements - historically one of the weak points of VEs. The demands of scientific visualization – for example when visualizing properties of a real model in a virtual

windtunnel – may make it necessary to provide the geometry and position of real objects into to the virtual environment.

- *Customized views*: In addition to individual choice of viewpoint, customized views of the data are possible, for example one user may want to see stream lines added to the basic image, while another may not. Two users in the same room may see different aspects of the same object at the same time, for example different visualizations of an airflow around the same model.
- *Usage of space*: The space in the Studierstube can be used similarly to a CAVE [Cruz-Neira1993a] (multiple users looking out at the environment), but also allows a workbench setup [Krüger1994] (users gathering round a desk, interacting with objects on it).
- *Organizational advantages*: While the cost of the Studierstube's hardware components are certainly higher than a conventional desktop visualization station consisting of only a graphics workstation, they are moderate compared to a setup like the CAVE. This is particularly important as the potential users of the Studierstube - research groups - are typically operating on a tight budget. Furthermore, the setup consumes little space and is relatively easy transportable.

Scientific visualization serves not only as a testbed and demonstration application for the features of our proposed environment but as a viable benchmark for our progress in the direction of a true shared virtual environment as general user interface concept.

On one hand, scientific visualization has a high demand for sophisticated interface techniques for user input and display, on the other hand many visualizations need a tight collaboration between experts in different fields. Its need for interaction on many different levels, like configuration of a simulation and visualization pipeline, input or adjustment of purely numerical parameters of the simulation and finally direct interaction with the visualization and feedback via computational steering make a monolithic implementation as single application unpractical. A modular concept of interacting applications and interface elements seems necessary.

In this thesis we address the following problems, which we identified as crucial for the implementation of our concept of a collaborative virtual environment for scientific visualization:

- a scalable system architecture with a consistent Application Programmer Interface (API)
- interaction elements and techniques supporting collaborative work
- calibration methods to improve precise interaction
- solving occlusion problems in the augmentation
- visualization icons and visualization focussing methods specialized for our virtual environment

In this thesis we address the following problems, which we identified as crucial for the implementation of our concept of a collaborative virtual environment for scientific visualization:

- a scalable system architecture with a consistent Application Programmer Interface (API)
- interaction elements and techniques supporting collaborative work
- calibration methods to improve precise interaction
- solving occlusion problems in the augmentation
- visualization icons and visualization focussing methods specialized for our virtual environment

7 Related Work

The design of our collaborative virtual environment Studierstube – first introduced in [Schmalstieg1998] - builds upon legacy knowledge from very different fields. In the following, we summarize some of the work that we consider most influential for our work.

Several research groups have created other augmented reality applications, either using video composition [Bajura1992] or see-through HMDs [Feiner1993b], like us. While those systems are intended for individual users, the shared space project [Billinghust1997] has focused on building collaborative augmented environments.

Other prominent attempts to create collaborative semi-immersive settings are the CAVE™ [Cruz-Neira1992] and the virtual workbench [Krüger1995].

Both systems present stereoscopic images to the user via large display screens and LCD shutter glasses:

The CAVE is a small room, composed of three back-projected walls and a front-projected floor, in which computer-generated images are displayed. The Workbench is essentially a table on which computer generated images are projected, resulting in a typical setup used by e.g. surgeons, engineers and architects. The resource requirements are less demanding than those of the CAVE, and the horizontal workspace is very useful for manipulation with hand-held tools.

Common advantages of both systems are high resolution, wide field of view, insensitivity towards lag for rotational head movements, and a strong feeling of immersion.

While these systems are frequently used simultaneously by several users, they are not strictly group systems: Only one “leading” user sees correct head-tracked stereo graphics, while for the remaining users the images are often severely distorted. Recently, a workbench extension for two users was presented [Agrawala1997], which - unlike the Studierstube setup - does not easily scale beyond two users because of inherent limitations of the display hardware.

A different area which has inspired our design are systems for collaborative work. While such groupware [Marca1992] has already been

integrated into commercial desktop environments, support for collaboration in virtual environments is currently an active area of research. However, most work focuses on remote training such as NPSNET [Macedonia1994] and social interaction (Diamond Park/SPLINE [Waters1997]), collaborative visualization such as [Wood1997] and C-Spray [Pang1997] [Gerald1993] and tele-meetings such as DIVE [Carlsson & Hagsand, 1993]. While such remote systems are not directly related to our approach of physical co-presence, they implement many useful ideas on how collaboration can be supported.

Graphical user interfaces are another field which influences our work. The desktop metaphor that was originally conceived in the Xerox STAR project [Smith1983] is ubiquitous in today's work environments such as X-Windows [Gettis1990]. The challenge lies in bridging the gap from the widely accepted document-centric 2D work style to a spatial 3D Workspace. While the use of 3D makes radically new interaction styles possible [Conner1992], judicious borrowing from 2D often eases the transition from the desktop to the virtual world. In particular, there are some attempts to organize content as well as re-use existing 2D developments [Angus1995] [Dykstra1993] [Feiner1993a]. In some sense, the PIP [Szalavári1997] used in our system also falls into that category as a container for 2D as well as 3D user interface elements.

A few researcher have dealt with the problem of spatially arranging and managing data from different contexts. The cognitive coprocessor architecture [Robertson1989] uses so-called 3D Rooms to provide multiple virtual Workspaces for a user. [Feiner1990] describes a system of nested volumes for the visualization of high-dimensional data. The most directly related approach to ours is the CRYSTAL system [Tsao1997] that allows a user to organize his or her work in 3D windows. The taxonomy in Figure 5 was originally introduced for the CRYSTAL system, which supports multiple contexts and applications, but lacks true multi-user support. Note that "CRYSTAL in the CAVE" is not really a multi-user application, as only one user can be active and the other users are passive observers. In contrast, the Studierstube Workspace design which this thesis focuses on aims at the construction of a system that is scalable in all three properties.

Early adopters of virtual reality (VR) systems soon realized that one of the immediately useful applications comes from the field of scientific visualization, where scientists try to understand complex data sets and can benefit from true 3D, stereoscopy, and interactive exploration, e.g., GROPE [Brooks1990], the nanomanipulator [Taylor1993], and the virtual Windtunnel [Bryson1991].

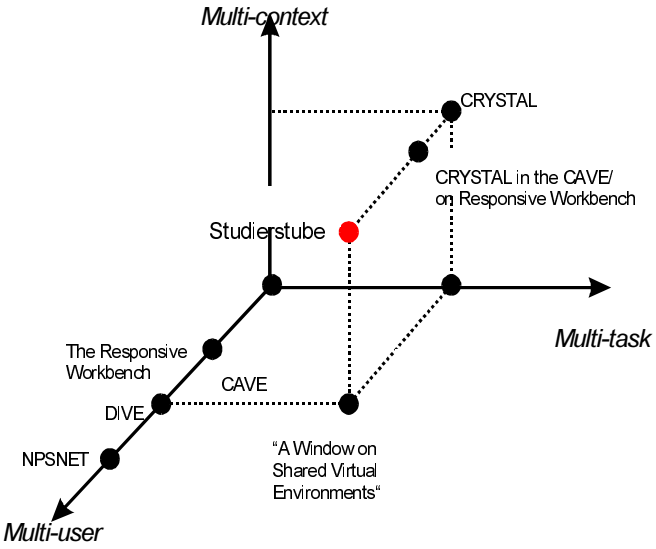


Figure 5: Taxonomy of virtual environments (adapted from: [Tsao1997])

The need to support collaboration of human users lead in two directions: remote collaboration [Bryson1993a, Pang1997] and collaborative virtual environments (VEs) where users come together in one place, and can interact and communicate in a natural way.

In the latter category, two approaches have been successfully used for scientific visualization: The CAVE [Cruz1993b], and the workbench, with two variants - Responsive Workbench [Krüger1995] and Virtual Workbench [Obeysekare96].

The combination of CAVE and a supercomuter (CM-5) for computational steering of scientific visualizations in [Roy1994] demonstrated impressively how virtual reality utilizing high-performance equipment can alter the way data is perceived and manipulated.

Finally, from a software engineer's perspective, development frameworks for user interfaces are notoriously huge and complex. A modular implementation as well as support for rapid prototyping via scripting is essential. In the area of 3D user interfaces, current prominent examples with features that have guided our design are ALICE [Pausch1993], MR [Shaw1993], Java3D [Sowizral1998] and the VRML97 ISO standard [Carey1997].

8 Workspace Concept and Design

In this chapter we present *Studierstube Workspace*, an application framework for augmented reality. We develop a concept for a collaborative working environments that simultaneously supports multiple users as well as multiple applications and in particular multi-tasking. The co-presence of multiple users interacting with multiple concurrently executing applications opens up new possibilities in the field of collaborative work. We describe how multiplicity in the interaction paths is used to implement the necessary underlying mechanisms and report on strategies and experiences regarding the development of multi-user aware applications.

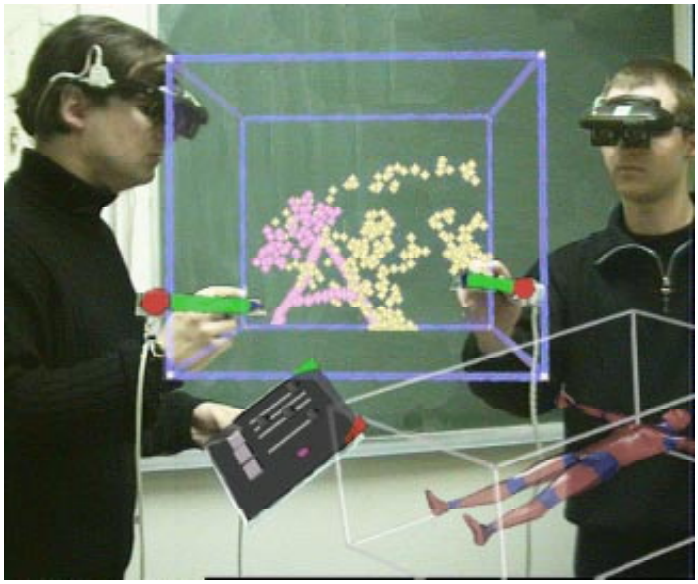


Figure 6: Collaborative work in STUDIERSTUBE Workspace (composited image): 3D painting application window (with focus, middle) and object viewer window (without focus, lower right)

As a user interface, Studierstube enables multiple concurrent users to interact with each other and with the application via the shared space provided by augmented reality as illustrated in Figure 6. Studierstube is a software toolkit which distinguishes itself from a dedicated environment by ist ability to execute a number of completely different

applications concurrently. Previously only one application could execute at a time, and operate only on one data context - like most VR systems, the user interface was neither capable of multi-tasking nor of multi-data context operation.

Studierstube is intended to be a collaborative AR user interface in which a variety of tasks can be performed. Such a user-interface is opposed to a dedicated application that is designed for only a single purpose (e. g., a driving simulator19).

We introduce an extension of our previous concept [Schmalstieg1998] that combines all these properties: *multi-user*, *multi-tasking* and *multi-data context*.

We give an analysis of the design space for such a collaborative AR Workspace. As a result of that analysis, we present a user interface design for such a Workspace that fulfills the three requirements listed above. Finally, we discuss our prototype implementation of a development framework for the Studierstube Workspace together with initial experiences and observations.

8.1 Hardware Setup

A typical Studierstube per-user setup consists of one semi- transparent headmounted display (HMD), one tracked pen and one tracked PIP (Figure 6).

The HMDs we use - Virtual I-O i-glasses - are very lightweight and unobtrusive, but only of limited resolution (230×263) and small field of view. As input devices we use a 6DOF pen with multiple buttons and the Personal Interaction Panel (PIP) [Szalavári1997] (see section 8.1.1 below).

Rendering is performed by SGI and InterGraph workstations, an additional Linux workstation services the magnetic tracker and distributes the tracker data via a LAN to the graphic workstations.

8.1.1 Personal Interaction Panel

The Personal Interaction Panel (PIP) [Szalavári1997] consists of a magnetic tracked pen and clipboard which contains augmented information presented to the user by see- through HMDs. It allows 2D interaction and three dimensional direct manipulation to be done in parallel. Unlike many other interfaces it implements a 2D interface in 3D rather than 2D besides 3D. Using the PIP is similar to using a notebook's flat surface in the real world. Our evaluations have shown that most test persons were familiar with the interface in a very short time and found the two- handed interaction metaphor natural and appealing.

8.1.2 Setup for Video Documentation

The augmented nature of our VE can also be exploited for documentation purposes:

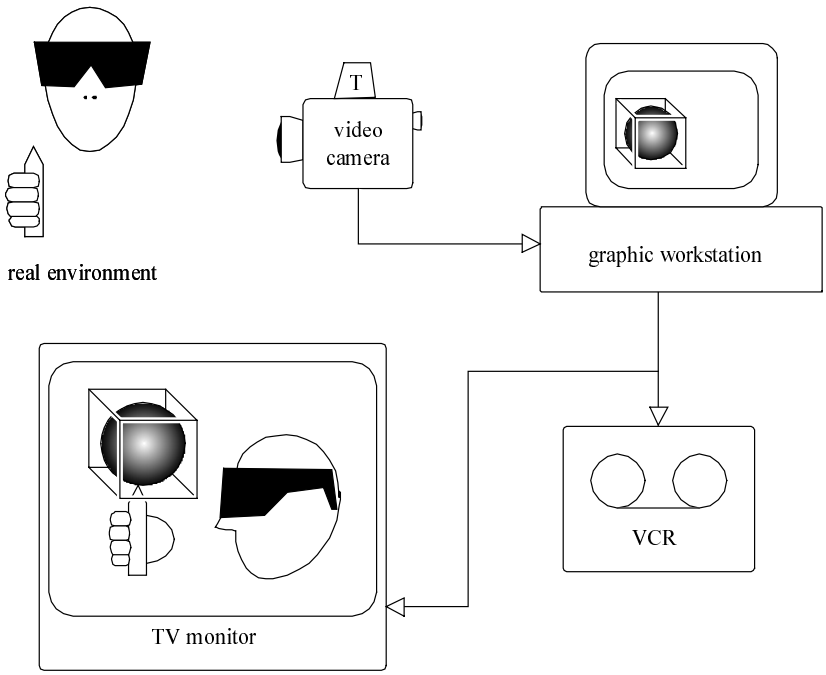


Figure 7: Setup for video documentation (digital overlaying)

We replaced one HMD by a video camera with attached tracker (Figure 7). The resulting images were overlaid in a workstation with the scene rendered from the viewpoint of the camera and the resulting images output directly to video tape. This method – which was also used to generate most of the pictures from the point of view of a user (e.g., Figure 11, Figure 14, Figure 29) – also enables the audience to view the augmented environment via a video screen. The resulting pictures can be transferred at NTSC resolution (640×480) to video tape.

Another, more straightforward method for capturing augmented reality on video tape is to mount a video camera behind a semi-transparent display as used in the HMDs (Figure 8). This is easier to implement, but results in a loss of picture quality, since the overlaying of computer generated images is performed using the optical system of the HMD, thereby reducing contrast and introducing distortions. We use this documentation setup only when we want to document properties of the optical system as for example in Figure 24.

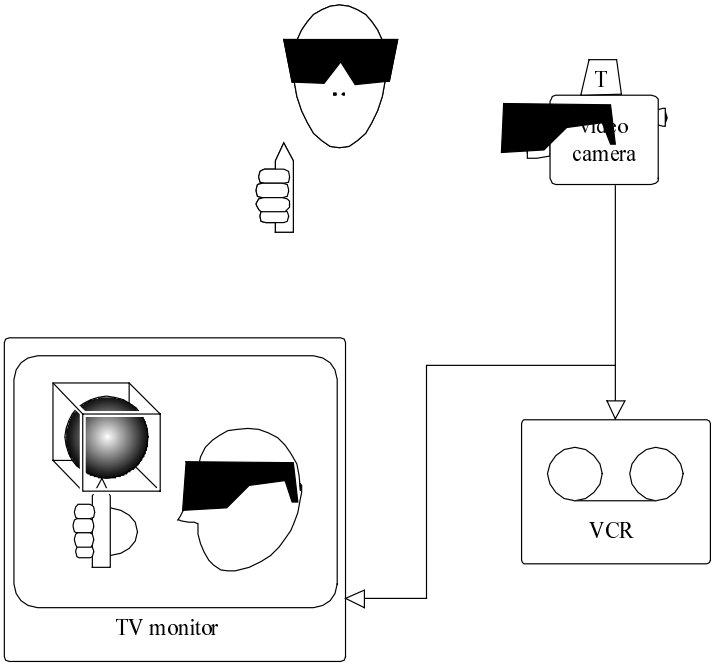


Figure 8: Setup for video documentation (optical overlaying)

8.2 Design Analysis of the Workspace

This section provides an analysis of the design space for a user interface that satisfies the requirements stated in the introduction, namely support for augmentation, collaboration, and multiple applications. To illustrate our choices, we will first describe a few example applications, that were chosen to characterize everyday activities that users may perform in the proposed work environment. From these examples, general guidelines for the design of the Workspace can be derived.

– Calculator: The calculator has a desktop-like interface on the PIP (Figure 9). Only one user may enter data at one time. Results are displayed on the PIP, and so no additional interaction or output elements are necessary.

Figure 9: Calculator application overlayed onto PIP

– 3D object viewer: The 3D equivalent to an image viewer application on a desktop is a simple object viewer. It supports the loading and display of multiple static objects in the Workspace (Figure 10). It does not support any interaction with the displayed objects, but allows to resize and move their representations via their surrounding 3D frames. Every object is self-contained in such a frame, and every user may move or resize it. Only one user at a time may open a new object, but the "object open" operation is accessible to every user.

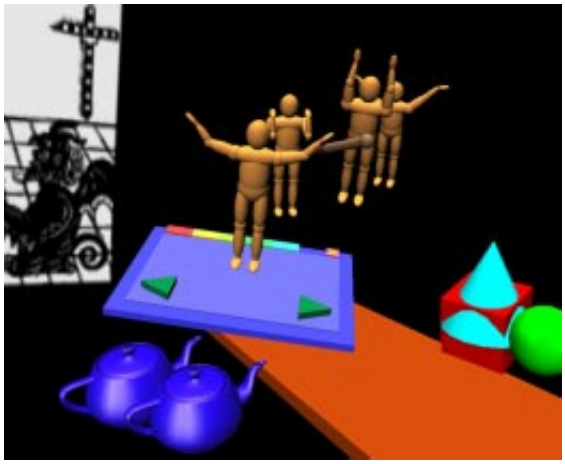


Figure 10: Viewing objects using the PIP

- 3D Paint Application: Contrary to the "object viewer" application, where one of many static 3D objects can be manipulated only by one user at a time, a 3D paint application should allow many users to manipulate a shared "scratch volume" at the same time (Figure 6). It shall be the equivalent to a blackboard: a space where a lot of users may display their ideas and add or modify the ideas of others. Each user should be able to specify his own painting tools properties - colour, width etc. - without modifying the painting mode of others.
- Collaborative games: A typical collaborative application is an implementation of any multi-user (as opposed to a solitaire) game (Figure 11). In a game of chess, white and black pieces may only be moved alternating, but every user can see the complete game state. A different example is a war game scenario consisting of terrain and different forces, where the (static) terrain exists as a real miniature model on which the forces are augmented. In this case all users can interact with each other and the forces simultaneously. Every user may only move his own forces. Collaborations between different parties may result in shared access to forces and to information on enemy territories outside of the own range.

Figure 11: Playing (silly) games in *Studierstube*

– Educational demonstration: In an educational setting a Workspace can be thought as an extension of the classroom into virtuality. The standard setting of one presenter and many viewers/students is of course a collaborative one, albeit a highly asymmetrical. Examples for such an asymmetry are a presentation where only the presenter may change parameters or a test situation where the teacher may see the work of each student, but where students should not be able to copy each others results.

The mentioned examples are intended to illustrate that the semantics of applications in the Workspace can be vastly different. Some applications are mainly oriented towards a single user or a group of isolated users, whereas others only make sense with multiple present users. In some cases the roles of the users are symmetric, while other cases exhibit asymmetry. Nevertheless, the design paradigm underlying these applications can easily be summarized: Every interaction path should support multiple instances.

Not only does the Workspace supply the junction point, but it also acts as an implicit "traffic control". To perform this task, the underlying framework has to support multiplicity in four distinct ways:

– Multiple users: Clearly the basic requirement for a collaborative setting

– Multiple applications: Only the support of multiple active applications at the same time enables a productivity scenario comparable to desktop environments.
– Multiple input contexts: By input context we mean the interaction state for input as perceived by a particular user. The same application may present the same interface elements - normally on the PIP - to different users, but the state of the elements (e.g., a selected colour) is presented on a per-user basis. Different users then see different colours on their PIP, and a change of selected colour by one user does not affect the selction performed by another user.
– Multiple output contexts: A single application may have more than one output context, normally represented as a 3D window. These contexts may be homogeneous (i.e. representing the same object in the applications context, e. g., a different view of the same building in a CAD system), or heterogeneous (e. g., a windows that shows attributes of an object selected in another window). Both the homogeneous and the heterogeneous case will use multiple 3D windows to present the different sets of information.

In the following sections we will detail how the Workspace implements these properties.

8.3 User Interaction in Workspace

A 3D Workspace requires interface elements for efficient manipulation and customization by the user. Here we give an overview how input and output is organized from the user's perspective. The relationship among the user interface components below will then be explored from a more technical side in the next sections.

One of our design goals was the extension of as many 2D GUI mechanisms into our VE as possible. As pointed out previously [Angus1995], there are two important reasons for this approach:

Firstly, since many 2D elements have been developed and refined over the last years and have proven their worth, it would be counterproductive to completely discard the well-developed solutions for these elements.

Secondly, most users are accustomed to the use of 2D GUIs. An approach that builds upon widely accepted user interface elements is more likely to ease the migration from conventional desktop computing to VE Workspaces.

While we do build upon the legacy of 2D GUIs, we do not want to impose unnecessary constraints upon our interface: we use 2D techniques where they prove to be of advantage - either by the nature of the data to be input or by the conventions associated with a specific operation - and extend them or completely abandon them when appropriate.

8.3.1 2D widgets in AR

For the straightforward integration of conventional 2D interface elements like buttons, sliders etc. we use PIP, a simple board with attached tracking sensor, which functions as base for the virtual 2D interface elements (Figure 12) in two ways: it allows the user to position the 2D interface conveniently using the non-dominant hand and additionally provides the necessary haptic feedback when using a slider or button. “Floating menus” or similar virtual interfaces lack this feedback and can therefore only be used when in the field of view. Every application may display its own interface in the form of a PIP “sheet”, which appears on the PIP when the application is in focus. The PIP and pen are our primary interaction devices. Both of them are tracked with 6DOF and provide the means for flexible interaction in 2D and 3D. The PIP cannot only be used as passive base but also as interaction and selection tool in itself. Its use as “virtual camera” to make 3D-snapshots of the scene as well as “fishnet” metaphor for selecting objects by sweeping it through space has been demonstrated [Schmalstieg1999].

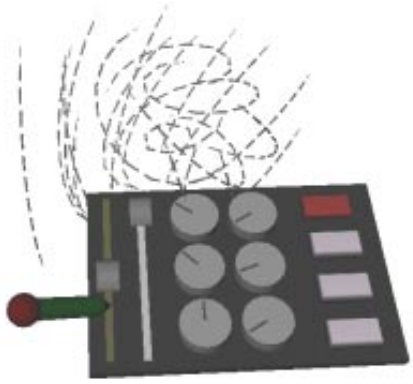


Figure 12: 2D interaction elements on the PIP used for parametrization of flow visualization

8.3.2 3D Windows in AR

The use of windows as abstraction and interaction metaphor for an output context is a long-time convention in 2D GUIs. It's extension to three dimensions seems logical [Feiner1990, Tsao1997] and can be achieved quite straightforward: Using a box instead of a rectangle seems to be the easiest way of preserving the well- known properties of desktop windows when migrating into a VE. It supplies the user with the same means of positioning and resizing the display area and also defines its exact boundaries (Figure 1). Obvious differences of these 3D windows ("boxes", see Figure 14) to their desktop counterparts can in many cases be resolved easily. Positioning a box by grabbing a designated part of its geometry may of course include the rotation of the window to an arbitrary orientation. Resizing is achieved by grabbing a corner and repositioning it with 3DOF, thereby changing all measurements of the box in one movement.

Differences appear in a case easily resolved in two dimensions: overlapping display contexts. On the desktop this is easily handled by a 2½D approach, rendering one window "on top" of another, effectively implementing a stacking order of windows. In three dimensions the equivalent solution would be a similar explicit order in which the windows are rendered - possibly based on importance or least recently used criteria - where each window clears the volume it defines before rendering its contents. This leads to an unambiguous assignment of

each point in the working volume of the virtual environment to at most one box.

8.4 Input and Output Contexts

The difficulty in designing a user interface for multi-user multi-tasking comes from the fact that the semantics of applications in the Workspace can be vastly different. A setup where multiple users are sharing the same work volume and the same set of applications significantly increases the combinatorial possibilities of interaction. This situation is opposed to the desktop world, where each user in a multi-user system has a separate desktop.

Some applications in the multi-user Workspace are rather oriented towards a single user or a group of isolated users, while others only make sense with multiple present users. In some cases the roles of the users are symmetric (such as in a game of chess), while other cases exhibit radical asymmetry (such as a teacher with students).

Nevertheless, the design paradigm underlying these applications can easily be summarized: *Every interaction path should support multiple instances.*

By *interaction path* we mean the way data flows from the user to the application and vice versa. To control this flow of data, the Workspace framework introduces input contexts and output contexts. The relationship among multiple users, input contexts, applications and output contexts is depicted in Figure 13.

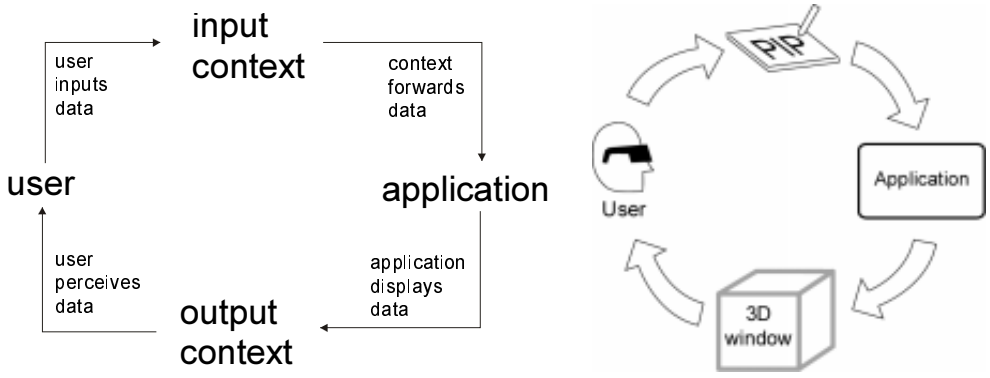


Figure 13: Relationship of Workspace components (left) and their implementations (right). Note how the human user acts in a loop with the application, mediated by input and output contexts.

8.4.1 Input Context

By input context we mean the interaction state for input as perceived by a particular user. There are two distinct ways in which a user may input data to a Workspace application:

1. The application presents a number of user interface widgets like buttons or sliders on the surface of the PIP - a so-called *PIP sheet*, somewhat akin to a conventional dialog box (Figure 61). The input context in this case is the state of a particular instance of a PIP sheet.
2. The application accepts 3D input (e. g. clicks and drags) in the *client volume* of a 3D window belonging to the application - this input style allows the design of direct manipulation of the application's data. Every user may lay a *focus* on one particular window at any time (note the color-coded focus on the left window in Figure 14), this focus is equivalent to an input context.

8.4.2 Output Context

An output context is manifested in workspace as a 3D window - an application may display its data in one or multiple windows. Multiple output contexts can be either homogeneous or heterogeneous:

– *Homogeneous output contexts* display information of the same kind in the same way. The difference between two homogeneous output contexts of one application is only in the actual data context being displayed. In the desktop world, this style is known as multiple document interface (MDI) - multiple data contexts can be opened at once, and the same interaction possibilities are available for each data context (Figure 6).

– *Heterogeneous output contexts* display information structurally different. For example, a 3D view of a data context may be displayed in one window and a schematic view in another.

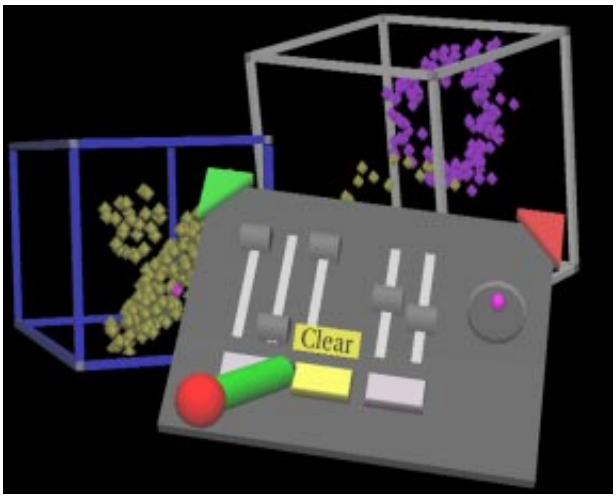


Figure 14:Painting application with multiple output contexts (windows) and color selector displayed on the PIP. The painting application has two open windows, input is directed to the left window, which has the focus

8.5 Multi-user Aware Applications

This section explains how a multi-user aware application (MUA) is designed by sup-porting multiple instances of input and output contexts. Every user can potentially interact with every application, which creates a matrix of potential interaction paths (Figure 15). Every such interaction path requires an instance of an input and output context that routes information between user and application. By allowing multiple instances of the same context type, the required multiplicity and independence of interaction from and to multiple users can be guaranteed.

Note that this multiplicity is not equivalent to running multiple instances of one application concurrently (e.g., one application instance

per user), as any user may interact with any data of the application at any time, and the actions of multiple users may influence each other. Figure 15 shows the multiplicity relationships of Workspace components.

Output contexts are represented as 3D windows. For Multiple Document Interface (MDI) style interaction, a separate window is opened for each data context. However, a window not only represents an output context, but also defines an input context. Input regarding a particular window may either be by direct manipulation of the data in the window, generally via a user's pen, or indirect via a PIP sheet.

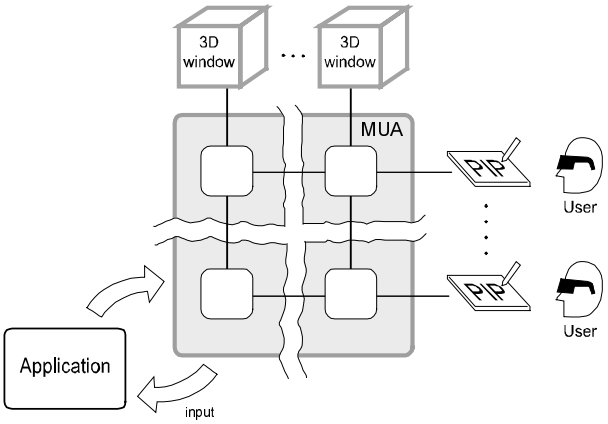


Figure 15: Multiplicity relationships among Workspace component - a context for a particular application is defined by every pairing of a user and a window

The proposed multiplicity of interaction paths implies some changes in application design. Full support of all possible interaction paths leads to some extensions to the applications interface. A multi-user aware application (MUA) has to take into account additional information concerning which user is interacting with it and keep user-dependent information. While in some cases multiple instances of single-user applications are sufficient (e.g. calculator and object viewer as described in section 3.) only a MUA can support a fine-grained collaborative workstyle where both users can interact with the application without extensive context switches.

While simple single-user applications shall be able to function in the Workspace without any knowledge of the multi-user setup, a multi-user aware application has to recognize and support one or all of the following situations:

8.5.1 Different Internal States for Each User

Depending on the applications requirements possibly extensive user-context information has to be kept for each user. Many standard operations allow different semantics when extended to multiple users: Shall a cut-and-paste sequence access a shared clipboard or should there be a clipboard for every user? Is there one common or are there many individual command histories?

In these abstract cases both variations make sense and can only be decided on a per-application or per-environment basis. The Workspace supplies all relevant user-specific information to the application program via one of its manager classes (section 7.2) and leaves this policy decisions to the application.

8.5.2 Multiple Input Foci on One 3D Window

The same window receives at the "same" time input from more than one user. While idempotent operations are possible, in many cases input from different users requests special consideration. An example would be two "dragging" operation at the same time. Our Workspace implementation supplies all MUAs with all necessary information to differentiate between actions executed by different users via the 3D event system (section 7.1). The application is left to decide whether to support a concurrent workstyle or lock the application when one user is already accessing it.

8.5.3 Multiple Users Input on the Same PIP Sheet

Since the PIP as our primary input device has to be used in MUAs too, special considerations for its behaviour are necessary. Normally, every application displays all of its 2D interface elements on its own

"sheet" on the PIP, meaning a combination of widgets only displayed on a users PIP when the users inputs focus resides within the application. Since a user can acquire an additional focus of a MUA while another user is already working with it - e.g. by selecting an output window of the application - the same PIP sheet will appear on the PIPs of both users.

A MUA has to manage a different state of its associated PIP sheet for every user. Since this state may be visible - e.g. in the position of a slider - it is not sufficient to create multiple renderings of the same sheet on different PIPs, but also to supply each sheets instance with its own state. Again, a Workspace manager exists which allows to distinguish between PIPs on a per-user basis.

8.5.4 Direct Manipulation in the 3D Window

Multiple users may simultaneously lay their input focus on the same window and manipulate the contained data. The application receives the users' activities in the form of 3D events (see section 8.6.1 on the 3D event system). Depending on the operation, such concurrency may or may not make sense. As the application receives events in serialized order, it is straight forward to implement mechanisms to control concurrency.

Applications relying on direct manipulation generally offer several activities at once. As a result, users perceive their doing as concurrent work on the *same* data, while really they are engaged in two *separate* activities in the same general context. In contrast, actual engagement of two or more users in the very same activity is mostly undesirable: What should be the outcome of two users trying to drag the same object into two different directions?

As the content of the window and hence the possible operations will be very different from application to application, Workspace provides locking mechanisms of different granularity:

- For coarse-grained concurrency control, the window focus policy can be decided by the application: It may either allow multiple users to focus a particular window simultaneously, or only one user is allowed to have a focus. In the case of a single user focus, a second

user may either ”steal” the focus from the first user holding the focus, or is forced to wait until the first users voluntarily gives up the focus (usually by focusing on another window).

- On a finer grain of interaction, some utility interaction classes, such as object draggers, are implicitly locked during operation by one user. This effectively disables all other users to gain control while a user is manipulating e.g. a slider.

Finally, should true multi-user interaction on one widget be desired (e. g., the joint definition of a rubber box by two users holding opposite corners), it is the applications responsibility to implement the desired behavior from the 3D events sent to the widget.

8.5.5 Indirect Manipulation via PIP

For homogeneous windows (windows representing different instances of the same class), the structure of the PIP sheet is the same for every window, but the state of the interface (e. g., the position of a slider widget) in the most general case depends on the window and the user. For example, a selected color may be presented on the PIP. The color will differ from user to user and can be manipulated individually. However, when a user switches focus from one window to another, a different interface state on the PIP becomes active - the state is also individually different from window to window. In other words, the degree of sharing can be set on a per-widget base. In general, four configurations make sense:

– Different interface state for every window and every user

– Different interface state for every user, but for a given user the interface state is the same for all windows

– Different interface state for every window, but for a given window the interface state is the same for all users

– Only one state for all windows and users

States may also be shared by groups of users or groups of windows, but we have found little use for this option.

8.6 Implementation

The Workspace software development environment is realized as a collection of C++ classes which extend the Open Inventor (OIV) Toolkit [Strauss1992] (Figure 16). The rich graphical environment of OIV allows rapid prototyping of new interaction styles. We also use the file format of OIV to enable convenient scripting of the properties of an application and to include our custom classes. A further advantage of OIV is its availability on IRIX and Windows NT, which allows for some flexibility in the selection of equipment.

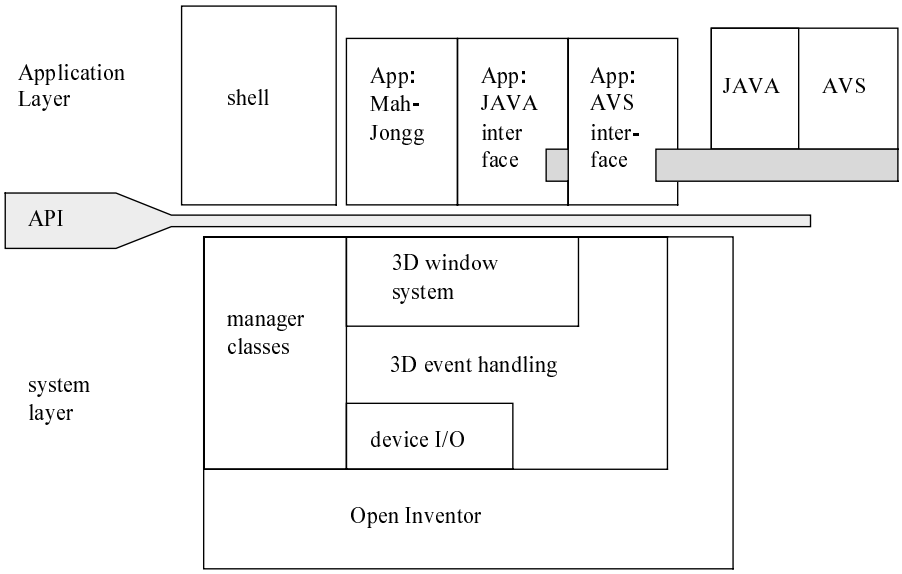


Figure 16: *Studierstube Workspace* architecture schematic

OIV represents scenes using scenegraphs - directed acyclic graphs - which can not only contain geometry but also active interaction objects. We make use of this property by implementing most elements of the workspace as scene graph nodes that are capable of processing 3D events. This allows efficient design of user interfaces like for example PIP-sheets by simple writing an OIV ASCII file which includes the necessary widgets.

8.6.1 3D Event System

Open Inventor's event system has been extended to process 3D events, which is necessary for choreographing complex 3D interactions. Since OIV does not support 3D interaction with 6DOF but normally projects 2D input into 3D space, we had to implement a new 3D event class, which distributes events containing 6DOF information through the scenegraph using bounding box information. A new class hierarchy of 3D interaction objects allows for the easy integration of active components in a scenegraph.

These objects respond to events generated by a 6DOF input device - in most cases the pen - by altering their appearance, position or internal state. They can filter events depending on user or type or "grab" an input device, redirecting all input to only one object.

This basic 3D interaction classes have been subclassed for the PIP to implement standard 2D widget behaviour (buttons, radiobuttons, sliders and dials, see figure 3). Additional features like highlighting and flyover help are supported for these widgets.

The more general mechanism for full 6DOF interaction are in most cases implemented by the Studierstube applications. This is done similar to 2D GUIs by opening a 3D window object and attaching a window callback function which receives all events generated in the windows volume (move, drag and click). Some commonly used types of 3D interaction like move and resize of passive objects have been integrated into special scenegraph components, which allow the scripting of simple applications in OIV ASCII files.

8.6.2 Output Contexts

Output contexts in Workspace are implemented as 3D windows. The 3D window class is a container associated with a user-specified scene graph. This scene graph is normally rendered with clipping planes set to the faces of the containing box, so that the content of the window does not protrude from the window's volume. Nested windows are possible. The window is normally rendered with associated "decoration" that visually defines the windows extent and allows it to be manipulated with the pen (move, resize etc.). The color of the decoration

also indicates whether a window has a user's focus (and hence becomes an input context for that user).

The *representation* of a 3D window can be maximized, 3D window, or minimized:

- *Minimized windows* are only accessible via its application icon in the application loader and consume no space in the working volume.
- *3D (normal)* window is displayed with a surrounding frame, which allows positioning and resizing via dragging of the edges respective corners.
- *Maximized* windows do not have the frame of 3D windows, effectively consuming the whole display volume. Since only one application window may be displayed in this state, maximizing forces all users to work with the same application.

Output context multiplicity is straight forward as all users can see all windows. An application wishing to display multiple output context simply creates multiple windows and associates the desired context with it. Maintaining homogeneous output contexts simply requires that those application methods that manage window content are parameterized by window. To simplify MUA development, application programmers can subclass a foundation class that already encapsulates such behavior.

8.6.3 Input Contexts

A PIP sheet as an input context has both an important function as a means of interaction as well as a visual representation. PIP sheets are therefore realized as OIV scene graphs mostly composed of Studierstube interaction widgets (such as buttons etc.). However, the scene graph may also contain passive geometry (e. g., 2D and 3D icons) that are useful to convey user interface state or merely as decoration.

In general, the internal state of widgets depends on the context, i. e. it is parameterized by user and window. A *context switch* can have an impact on the visual representation of a widget (e. g., current position of a slider) and therefore requires a widgets to be rendered using a

different internal state. Active context switches occur if a user changes the window focus. Passive context switches occur - unnoticed by the users- if the system progresses from one user to the next as all users' views of the environment are rendered in turn.

Input received from a widget (such as triggering of a button) will usually be addressed at the currently focused window. However, the semantics of any interaction via widgets is dependent on the application and can therefore not be predetermined. For example, an operation may affect *all* windows rather than only the active one.

Upon the creation of a PIP sheet, an application may also indicate that the state of particular widget is to be shared by all windows (per user), all users, or all windows and users. Any of these sharing options simply results in fewer states per widget and is trivially implemented.

Note that in principle the same options for sharing not only apply to interface state, but also to data representing the internal state of the application. However, our framework currently only supports multiple contexts on the interface level. It is the applications responsibility to manage context switches of internal state.

Naturally, any widget shared by multiple users is implicitly locked during its operation by the user who initiated manipulation, to prevent errors and undesired behavior resulting from contention and race conditions.

8.6.4 Studierstube Manager Classes

The manager class in Studierstube Workspace give access to high-level interaction concepts. While the basic interaction element classes implement widgets like sliders, buttons or 6DOF draggers and do not depend on any interaction concepts besides the 3D event system (section 7.1), these classes implement the specific Workspace concepts like support for multiple application, multiple windows and multiple users.

8.6.5 Application Manager

The Workspace job management allows loading a new application into Workspace and starting and stopping of applications. This function is mostly used by the application loader sheet (section 8.1) but may be used by any application to start a helper application like an object viewer or to stop another application. Starting and stopping applications has to be executed synchronously (between screen updates) with the application, which the application manager achieves this by sending an EXIT message via the message manager (section 7.6). Upon receiving this message, the application may perform necessary clean up functions and then exit.

8.6.6 Resource Manager

The Workspace resource management implements inquiry and setting of Workspace and device attributes as listed in Table 1.

resource	attributes
workspace	dimensions, number of users
pen	associated user, geometry
HMD	associated user, geometry, calibration
PIP	associated user, geometry, sheets, active properties: (fishnet, snapshot)

Table 1: resource attributes

Some of this attributes, like PIP sheets and pen geometry, are obviously used by most applications, but some of them, like number of users, concern only MUAs. HMD geometry for example is only set when an application wants to attach augmented information to a user and

HMD calibration only is accessed by the Workspaces calibration utility.

The ability to attach active components to a PIP sheet as demonstrated in the landscaping application (section 8.6) in the form of a "fishnet" - to select by sweeping (Figure 65)- or to use it as a magic lens (Figure 62) can be accessed via this manager.

8.6.7 Window Manager

The Workspace window management implements the creation and destruction of window objects and the setting of window attributes.

Callback functions for rendering and event processing and extensions like "drag-and-drop" between windows can be specified via this manager.

window attribute	content
focus	on / off
title bar display	on /off
representation	minimized, 2D, 3D, maximized
cursor	cursor geometry

Table 2: 3D window attributes

The window manager furthermore manages the states of the displayed windows. The most important attributes of 3D windows are listed in Table 2.

The representation of a 3D window can be maximized, 3D window, 2D window or minimized.

- Minimized windows are only accessible via its application icon in the application loader and consume no space in the working volume.
- 2D windows are displayed as a flat frame through which the applications geometry can be seen.

– 3D (normal) window is displayed with a surrounding frame, which allows positioning and resizing via dragging of the edges respective corners.
– Maximized windows do not have the frame of 3D windows, effectively consuming the whole display volume. Since only one application window may be displayed in this state, maximizing forces all users to work with the same application.

Displaying an application in a 2D window reduces space consumption too, but still allows viewing the applications display in realtime. All application output is still rendered in 3D, thereby generating the effect of watching it through a “window in space” or “magic mirror”, which can be placed like a framed picture somewhere in the Workspace. This mechanism is very efficient when an applications output has to be watched while working in another window. It is implemented using our SEAM interaction element25. Interaction with applications in this state can only be accomplished via the PIP.

8.6.8 Message Manager

The Workspace application-level message passing implements a general communication mechanism between studierstube Workspace objects, mainly between applications themselves or between applications and managers (table 3).

System events implement task management and are generated by the application manager (section 7.3). They are routed to special methods of the application. *Window* and *application messages* (Table 3) are generated by the window manager or another application and are passed through the specified receiver windows window-function (section 7.5).

Most of these messages are familiar from conventional 2D window managers, only their additional qualifiers distinguish them from 2D events. The introduction of real 3D coordinates and user identification enable the full functionality of Mult-User aware Applications (section 8.5).

Message	**Description**	**Receiver**
WM_RESIZE	window has been resized	window
WM_CLOSE	window has been closed	window
WM_EVENT	3D event has occurred in window	window
WM_WINDOW_GOT_FOCUS	window has lost a user's focus	window
WM_WINDOW_LOST_FOCUS	window has got a user's focus	window
AM_SHUT_DOWN	application is being shut down	application
AM_REMOVE_APPLICATION	request to shut down an application	App. manager
AM_APP_TO_FOREGROUND	called when a user changes to another application	application
AM_APP_TO_BACKGROUND	called when a user changes to another application	application
MM_DELIVERY_REPORT	delivery result of a message sent by an application	application

Table 3: selected Studierstube Workspace's messages

9 Fast Calibration for Augmented Reality

9.1 Overview

Augmented Reality overlays computer generated images over the real world. To correctly depict spatial relations between real and virtual objects, these images have to be generated using transformations which correctly project a point in virtual space onto its corresponding point in the real world (Figure 17).

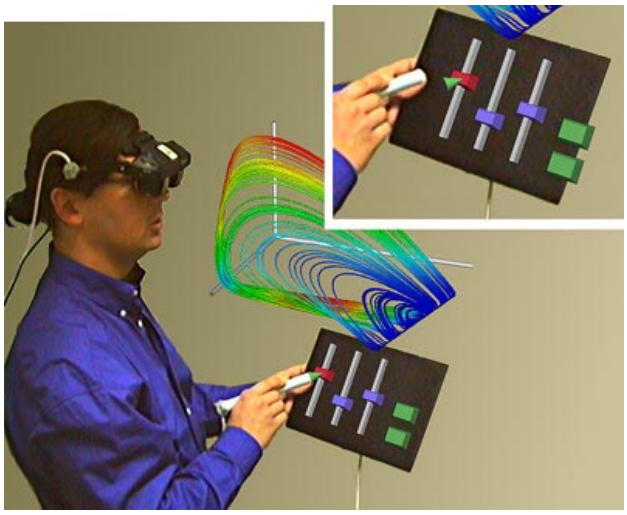


Figure 17: Augmented Personal Interaction Panel
(inset miscalibrated)

This requires precise knowledge of the viewing projection of the head-mounted display (HMD) and its position. Methods to calibrate the projection parameters of the virtual cameras to that of the HMD have been presented in previous work (section 9.4). Most of these methods are complicated or use special equipment for the calibration process.

We present a simple and fast calibration scheme, which does not require additional instrumentation or complicated procedures. This allows us to calibrate HMDs for specific users. The calibration process is presented to the user as interactively guided initialization step, which

enables even inexperienced users to adapt the calibration to their eye distance and head geometry.

The calibration is stable - meaning that slight errors made by the user do not result in gross miscalibrations - and applicable for see-through and video-based HMDs. Additionally we show a method for correcting the distortions introduced by the camera lens in real-time using standard OpenGL hardware and for registering tracked objects to their augmentations.

9.2 Introduction

Virtual Environments (VEs) present to the user computer generated images in the same way a real environment would be perceived: each eye perceives an image depending on its position and the direction of view. These images are generated according to the users heads position in space. The position is tracked by special sensors and used to move the virtual cameras inside the VE accordingly.

Slight errors in this process - introduced by erroneous measurements or wrongly defined camera parameters - result in distortions of the view presented to the user, like a wrong perspective or parallax, giving a false apparent position of the virtual objects. Additional errors are introduced when the camera model used for generating the virtual images does not contain parameters for simulating the distortion of the real cameras lens (Figure 18). Most hardware accelerated rendering – e.g, OpenGL [Woo1997] – is done using a simple camera model which does not take lens distortions into account.

Figure 18: Lens distortion (red lines rendered with idealized camera)

While these distortions are only annoying in an immersive setup - where the user only perceives the virtual environment - they can in many cases be overcome. The hand-to-eye coordination of the user for example adapts to the altered visual feedback given by the virtual image of the hand. Nevertheless resulting differences between the visual and the kinesthetic sense may result in motion sickness [Pausch1992].

In *Augmented Reality* [Feiner1992], where computer generated images overlay the users view of the real surroundings, such distortions cannot be tolerated. When a user perceives two different loci of interaction, one given by the real image of his hand and one by the virtual image on a different position, the perceived clues conflict and hand-to-eye coordination is severely impaired.

In Figure 17 the Personal Interaction Panel [Szalavári1997] is shown, 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 slight misalignment as described above. 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.

To guarantee the necessary correct alignment of real and virtual environment, some prerequisites have to be fulfilled. In this chapter we present simple solutions for two of the most important goals:

- HMD calibration (calibration of virtual camera)
- alignment of virtual to real objects

Several approaches have been proposed in the past to attack the problems of calibration and registration (section 9.4). However, they are generally not designed to be operated by untrained users. This is a fundamental problem as every user is different, and consequently some per-user setup is inevitable for high-resolution AR. However, a complex or cumbersome process will lack user acceptance and will mostly be ignored.

If augmented reality is to be deployed outside of research labs, procedures must be simple and may not require trained operator intervention.

In this chapter, we present a simple, stable and interactive method to implement camera calibration. The method requires little effort on the side of the user and should thus be more appealing to novice users and experts in proliferated lab practice alike. It is usable for see-though HMDs, does not require additional hardware, and is numerically stable. Furthermore a fast solution for the correct alignment of real and virtual objects is presented.

9.3 Registration in Augmented Reality

The alignment of virtual objects and their associated real counterparts is called registration. The desired result is correct registration of the images of the real and virtual objects. To achieve this alignment, a couple of different criteria have to be fulfilled.

9.3.1 Correct Sensoric Input

Tracking data - especially data acquired by magnetical trackers as in our case - is, like all measurements, prone to errors. Time lag, nonlin-

ear distortions and noise have to be compensated, otherwise no registration is possible.

This lies beyond the scope of this thesis, relevant publications are [Azuma1994, Bryson1993b].

9.3.2 Registration of Real and Virtual Objects

Stationary and moving real objects in the working volume which are to be augmented have to be registered to their virtual counterparts. E.g. to attach a virtual label to a specific part of a real model, the system would have to know the models position and dimensions. For moving objects this knowledge has to be updated in real-time using tracking data.

A simple interactive method for registering virtual to real geometry is discussed in section 9.9.

9.3.3 Camera Calibration

To achieve image registration, meaning alignment of real and virtual world when projected on the retinae of the user, we also need a precise description of the projection from the real world onto each retina. This step, called camera calibration for the remainder of this chapter, has to determine the intrinsic and extrinsic parameters of the virtual camera which has to mimic the projection of the real environment.

In the course of this chapter, a simple, stable and interactive method to implement camera calibration is presented. While specifically developed for our shared augmented environment Studierstube, the calibration procedure is generally applicable wherever quick adaptation of camera parameters to a specific user is asked for.

9.4 Previous Work

9.4.1 Photogrammetric Approaches

In photogrammetric camera calibrations [Tsai1986, Faugeras1993, Janin1993, Tuceryan1995], as used in data extraction from aerial photos and computer vision, data points in 3D and their 2D projection are measured. From these quintuple an optimization algorithm produces camera parameters, in many cases via the intermediary form of an projection matrix, from which camera parameters have to be extracted afterwards.

The main disadvantage of these methods when applied to our scenario lies in the relatively large sample of data points they need to converge (50-100) and tendency to behave unstable when presented with erroneous data.

9.4.2 Direct Measurement of Camera Parameters

Some previous approaches use additional hardware for calibration:

Oishi et.al. [Oishi1995] use a "shooting gallery", which presents calibration patterns at varying distances to the user, whose head has to be fixed in a previously defined position. The user has to tag all these points to achieve calibration.

Azuma et.al. [Azuma1997] use orientation markers on a calibrated box, which require the user to align at the same time two pairs of these markers to each other and a virtual to a real crosshair to define the eye position and orientation. Additional markers allow the measurement of FOV and aspect ratio.

9.4.3 Object Registration

Stationary objects in the real environment can be registered offline by measuring their position and dimensions in the tracker coordinate system.

By using image processing techniques [Berger1997, Whitaker1995, Kutulakos1998] this can also be done for moving (rigid) objects during the simulation, but the necessary processing power and the low precision [Whitaker1995] and the lack of the for interaction necessary depth information [Berger1997] preclude us from following this approach. Furthermore image processing methods are only easily applicable for video-based augmented environments and suffer under line-of-sight problems.

Best results have been achieved when using pointer-based object registration Whitaker et. al., where users have to touch predefined landmarks on a real object with a tracked pointing device.

Like Whitaker et. al. we are using magnetical trackers on our movable augmented objects (PIP and the associated pen which also doubles as a 6DOF mouse). Methods of registering these movable tracked objects in the Augmented Environment are discussed in section 9.9.

9.5 Requirements

In the following, we discuss the requirements for a calibration method that can be used for augmented reality setups in everyday situations. More concretely, this section analyses requirements for a method that which allows untrained users to calibrate the HMD to their personal parameters, resulting in improved registration.

9.5.1 Reduced User Effort

As stated before [Azuma1994, Bajura1995] 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 achieve (e.g. calibration of field-of-view) and complex interactions with the system (see section 9.4). We want to reduce user interaction to a guided approach, which in 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.

As an example for a similar calibration procedure may serve the joystick calibration of many computer games: "Move the joystick to the four corner positions and press button afterwards". In other words: we want to maintain interactivity while reducing user effort.

9.5.2 Usable for See-Through HMDs

When using see-through HMDs, which optically overlay the computer images over reality, the users eyes - specifically their position relative to the display surface of the HMD - are an integral part of the registration problem and have to be taken into account when calibrating the virtual cameras. To achieve correct registration for a specific user we have to calibrate the HMD while it is being worn by this user. Even slight differences in eye-distance and distance between eye and optical system of the HMD lead to misregistration.

9.5.3 No Additional Hardware

Previous approaches [Azuma1994, Oishi1995] make use additional hardware for calibration. They deliver high-quality calibration 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 raises additional calibration problems (How to register the calibration hardware itself?) and reduces mobility of the whole system.

9.5.4 Numerically Stable

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 calibration 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.

9.6 Calibration Procedure

We use the usual pinhole camera model for calibration, since non-linear lens distortions are to be rectified independently (section 9.8).

The following parameters have to be calibrated:

- eye position
- position of (middle of) image plane
- orientation of image plane
- aspect ratio
- (horizontal) field of view

Our camera model takes into account the physically decoupled nature of eyepoint and image plane (Figure 21) in a see-through HMD. Since the users 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.

We have implemented a two-step optimization procedure, which optimizes these cameras 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 procedure 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 viable solution is determined geometrically.
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.

9.6.1 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 [Bajura1995] only determines the parameters of the video camera. Differences between the cameras parameters (FOV, interpupillary distance, etc.) and the users eyes are not taken into account, since the alignment of real and virtual images can be guaranteed in the first step. The discrepancies in the complete system only result in the same effects as in an immersive VE, as discussed in (Why calibration).

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 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 users eye, the resulting calibration will only be valid for the position of this camera. The data gathering stage of our calibration scheme, in which we have to acquire 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 turn the image processing approach - presenting a real calibration pattern and identifying points on its projection - around and present the user with a virtual calibration pattern, on which real points have to be aligned. The user sees a real marker on a tracked pen (Figure 19), which he has to align with a virtual marker presented via the HMD. When the alignment is achieved (Figure 20) 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 is transformed in the coordinate system of the HMD tracker sensor,

which eliminates influences of the head position. The resulting 3D point gives us - together with the known 2D location of the virtual marker - one quintuple of calibration data.

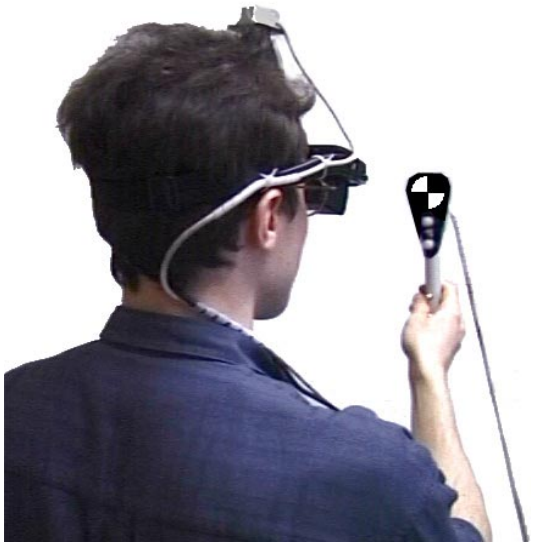


Figure 19: physical setup for calibration

Figure 20: View through the HMD - virtual marker overlays real marker

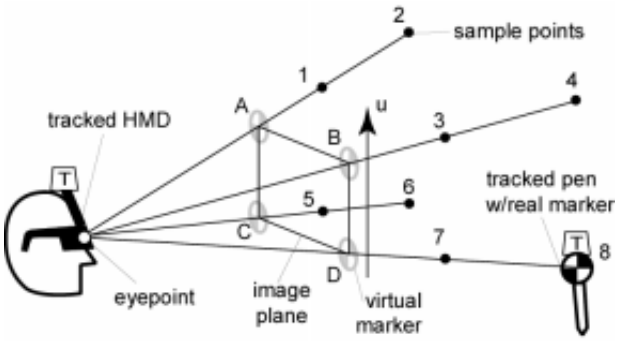


Figure 21: geometric determination of HMD parameters

9.6.2 Geometric Determination of Camera Parameters

Since we want to keep the number of sampled data points low (section 9.5.1), 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 21 the distribution of sample points for correct calibration is depicted as black circles (1-8). Every pair of samples lies on a line connecting one corner of the image plane with the eye point, essentially defining in this way the viewing pyramid.

In a first optimization step this gives us the location of the eyepoint as a least-squares solution for the point lying nearest to all of these lines. Averaging their direction gives us a good approximation of the viewing direction. As a first approximation at this stage we assume that the viewing direction is normal to the image plane. Intersecting this approximated image plane with the planes defined by the lines (1/2), (3/4), (5/6) and (7/8) gives us approximations for the horizontal, respective vertical directions in image space. Averaging the verticals (A/C, B/D) and the normals of the horizontals (A/B, C/D) gives us an approximation of the up-vector (u) for our camera model.

If the aspect ratio of the display area is not known it may be approximated in a similar manner from the intersection points of the lines through the sample pairs with the image plane (A, B, C, D), but in most cases this parameter is known for a given HMD.

This intermediate solution already gives a good approximation of the calibration problem. Differences between calculated and measured projections of the data point are in the range of 1-2%.

Since this solution only holds for eye positions on the axis of the optical system, we have to append a optimization procedure to account for off-axis positions of the eye.

9.6.3 Numerical Optimization of Parameters

Since - as already mentioned above - the solution at this stage is already very good, we do not have to apply sophisticated optimization techniques to it. A simple multi-dimensional least-squares optimization [Press1988] is being applied to the geometric solution reached in the previous step.

The parameters optimized in this step are the normal vector of the image plane, field of view (FOV) and - if needed - aspect ratio. The optimizations reaches a stable solution after 20-100 iterations and leads to errors in the range of 0.5-1%.

9.7 Fast Calibration

The procedures described above implement a full user-specific calibration of an unknown HMD with attached tracker sensor. It needs user input of 8 sample points per eye (a total of 16 samples per HMD) and yields a virtual camera pair which is optimized for a specific user, but may be used satisfactory by others, if they are not willing to repeat the full procedure. Ideally we want a calibration which separates the user-specific parameters (eye position) from parameters fixed for a given HMD/tracker sensor setup similar to the one used in Studierstube.

Given a sufficiently linear behavior of the optical system of the HMD which projects the display surface - LCD or CRT - onto a virtual image somewhere in front of the user, we can assume that each point on the display surface corresponds to a point fixed in space relative to the HMD (Figure 21). It should then be possible to determine this projection of points in image space (pixels) on a plane in the HMD coordi-

nate system. This would reduce the calibration problem to the determination of the eye position. It eliminates the need for more than one sample point along a edge of the sample frustum, since the (now) known 3D location of the virtual marker in real space in combination with the position of the real marker gives us a line in space through the eyepoint. This reduces the number of sample points per eye from 8 to 2.

We are using the full calibration data gathered from a sufficiently large user population to determine not only the orientation of the image plane but also its positions in real space in a least-squares optimization step similar to section 3. When the eye distances of the users vary sufficiently (ranging from children to adults) this optimization terminates with a solution applicable for a wide range of users.

While theoretically applicable with only 2 sample points per eye we normally use all 4 corner locations. This reduces the user effort versus the full calibration from 16 to 8 points while keeping the error margin low. The resulting errors (0.7-1.5%) are slightly higher than when full calibration is applied but the solution is still better than an "averaged" camera without user-specific calibration.

9.8 Distortion Compensation

A further problem for correctly registering and Augmented Environment are non linear distortions introduced by the optical system of the HMD. Especially in video-based HMDs the short focal length of the cameras lens can introduce radial distortions as the typical barrel distortion depicted in Figure 22. These distortions result from different focal lengths of the optical system for different points in the image plane. Equivalent distortions may be introduced by the optical system inside a see-through HMD.

Photogrammetric calibration schemes like [Tsai] mostly include some parameter(s) for describing a radial distortions, which allows a rectification of image data on a point-by-point basis.

Since this approach does not work on a standard OpenGL graphics hardware, where only linear and perspective distortions are possible, we use an approach similar to the one used in [Bajura1995], where

video images are non-linearly distorted before being fed into a video-based HMD.

To implement a general image distortion using OpenGL hardware acceleration we take advantage of the OpenGL texture mapping mechanism. To rectify an image we use it as a texture and map it onto a screen-aligned segmented plane, the rectification grid. By using the inverse of the distortion function the image becomes rectified (Figure 23).

In most cases a precise definition of the distortion function is not available, therefore we have to acquire it by measuring the distortion via a calibration pattern. Photogrammetric approaches deliver closed form solutions for the distortion function, [Tsai1986] for example gives a first-order radial-lens distortion parameter for rectification.

Since we are sampling the inverse distortion function at discrete points - the vertices of the rectification grid - we do not need to determine the closed form of the function at all. All we have to do is take a distorted image of the calibration pattern in Figure 22, measure the positions of the intersections and map these positions as texture coordinates onto the undistorted rectification grid. The texture hardware then rectifies the distorted images, using linear interpolations inside the grids squares.

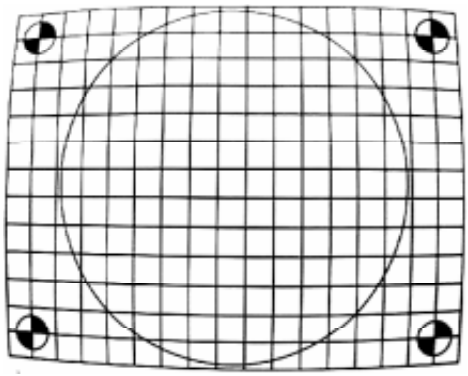


Figure 22: distorted video image of calibration pattern

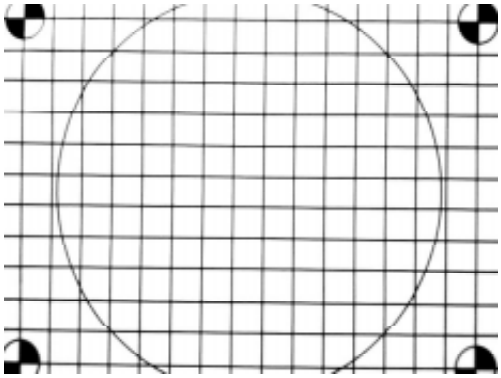


Figure 23: distortion compensated image

To apply this method to see-through HMDs one just has to distort the computer generated image instead of a video image. This is implemented by rendering OpenGL graphics into a separate region of the frame buffer and then using this region as a texture map. In this case of course the distortion has to be the same as the one seen through the HMD. Calibration data for this case can be gathered by letting the user click a lot of on markers lined up like the intersections of the rectification grid or - a simpler and faster approach - by capturing an image of the calibration pattern through the HMD.

The coordinates of the intersections can then be measured by hand or with a feature recognition algorithm.

9.9 Registration of Virtual to Real Objects

Registration of real objects to virtual counterparts has several uses. In the case of a physically based simulation for examples a virtual ball has to bounce on a real table, or annotations and instructions have to appear aligned to components of a real object []Feiner1992]. We have used tracked approximations of the users head and limbs to simulate the occlusion of virtual objects by the users body [Fuhrmann1999].

We have used an approach similar to [Whitaker1995] for registering static objects, but use a simpler method for registration of tracked objects. Tracked objects are objects like the PIP, or the occluding body geometry in section 10, which in the virtual world are represented as

rigid objects moving according to the tracker sensors attached to their real counterparts.

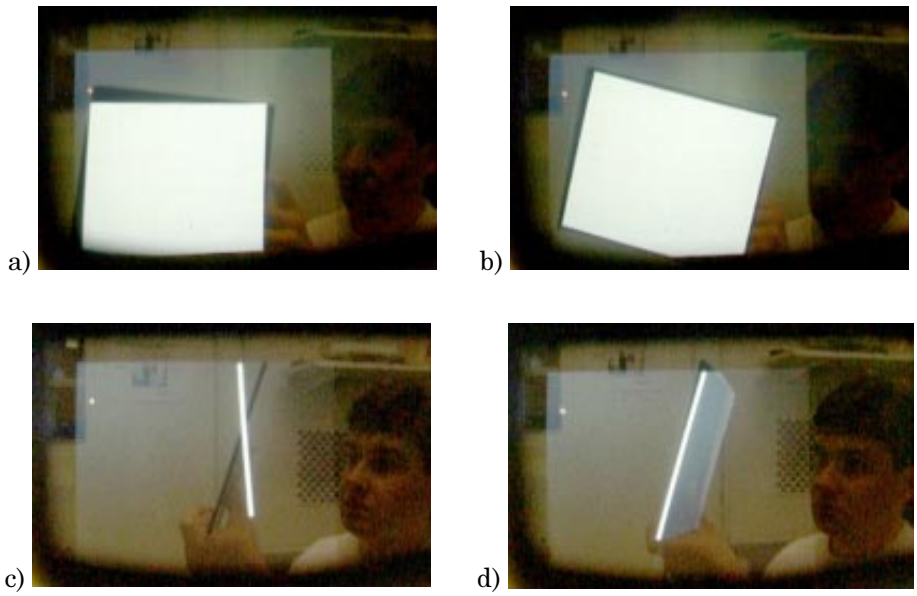


Figure 24: registration of virtual to real object: a) front view - misaligned
b) front view – aligned, c) side view – misaligned, d) registered

In our approach the user is presented with a augmented view of the real object, overlaid with its (non-aligned) virtual representation (Figure 24a). Using keystrokes, the augmented geometry may be move screen-aligned to cover the actual object. Rotation around the center of the object in the image plane are supported also. The user aligns the virtual object with the real from one viewing direction as shown in Figure 24a and Figure 24b, then turns the physical object around to look at it from another direction (Figure 24c). The procedure is repeated until satisfactory registration is achieved (Figure 24d).

This procedure is easily implemented and the user interface is intuitive enough for most users. Since the registration process has to be performed for each tracked object only once - baring changes in the attachment position of the tracker sensor - the necessary user effort is

kept small. It allows for fast registration of arbitrarily shaped objects without the need to attach markers on the object.

9.10 Implementation Details

The HMD calibration has been implemented in Open Inventor (OIV) [Strauss1992], the same graphics toolkit in which Studierstube has been implemented. The extended camera model has been implemented as an OIV extension, making it possible to save user-specific calibrated cameras in the standard OIV file format. The calibration runs as a separate module and allows the user to calibrate the HMD on the fly anytime during his work in the Augmented Environment. While this is normally done only once per user, passing a HMD to another user may make this necessary during a working session.

The procedure itself has been implemented as user-guiding step-by-step "calibration wizard" comparable to the "setup wizards" of commercial software. The user is shown virtual markers directly corresponding in size and shape the real marker, therefore not only prompting for a correct alignment in two dimension, but also differentiating between different distances. Figure 25 a and b show the markers according to the upper right corner at near and far location respective. When the optimization procedure detects one or more measurements beyond the tolerated error margin the user gets prompted to repeat the necessary steps. During the calibration process the user may repeat botched measurements by using a "back" button on the pen.

As an unforeseen advantage proved the separation between calibration of left and right eye. Since some people have problems closing a single eye, they expected to have difficulties with having to align markers. These problems did not occur, since the virtual marker is only presented to one eye at a time.

The calibration method has been applied to stationary cameras, thereby fixing their position in space without the expense of an additional tracker sensor. We use this setup to generate videos for documentation purposes.

All the augmented illustrations in this chapter have been captured with a digital camera directly through the optical system of our HMD - virtual-io i-glasses - and have been generated using a SGI Indigo 2 graphics workstation.

The distortion compensation has been implemented and tested on a SGI O2 R5000 workstation, using the O2 system camera as video source.

a)

b)

Figure 25: guiding the user through differently sized virtual markers. a) far sample point, b) near sample point

9.11 Results

Figure 26 shows the results of a full registration procedure. The virtual marker is in this case tracked and follows the real marker on the tracked pen over the whole working volume, independent of the users heads position and orientation.

The video images in Figure 27 demonstrate the real-time distortion compensation. The texture mapping method described in section 9.8 has been applied to a life video stream captured by a Silicon Graphics O2 system camera. The distinct barrel distortion in Figure 27a has been compensated in Figure 27b.

A first simple implementation of this method rectified the video at a rate of 22 frames per second (including video capturing time) on a SGI O2 R5000.

Figure 26: resulting registration of virtual to real marker

a)

b)

Figure 27: a) distorted video image,
b) distortion compensated image

The advantages of the presented calibration method lie mainly in its simplicity of application and its tolerance to user errors. While testing other methods we observed a certain reluctance in many participants to apply the calibration procedure when presented with a reasonable - but not user-specific - calibrated HMD. Our method - especially when embellished by the user guidance interface - seems to be more tolerable to the ordinary user. The simple task of matching one distinctive image with another proved to be easier to accept than any more taxing schemes.

The disadvantages of the method lie mostly in its dependence on precise user input and tracker precision for high-quality results. We are planning to improve our measurements by applying appropriate

filtering directly to the marker measurements already transformed into head-tracker coordinates. Since tracking delays and loss of high-frequency information are not relevant during the calibration procedure we expect to improve the precision considerably.

The use of gradient methods for optimization are another topic for improvements although the numerical optimization step seems to be stable and is fast enough for interactive expectations.

10 Occlusion in Collaborative Augmented Environments

10.1 Overview

Augmented environments superimpose computer enhancements on the real world. Such augmented environments are well suited for collaboration of multiple users. To improve the quality and consistency of the augmentation the occlusion of real objects by computer-generated objects and vice versa has to be implemented. We present methods how this can be done for a tracked user's body and other real objects and how irritating artifacts due to misalignments can be reduced. Our method is based on simulating the occlusion of virtual objects by a representation of the user modeled as kinematic chains of articulated solids. Smoothing the border between virtual world and occluding real reduces registration and modeling errors of this model. Finally, an implementation in our augmented environment and the resulting improvements are presented.

10.2 Introduction

One of the main advantages of using an augmented environment [Feiner1993, Azuma1997] for collaboration as opposed to an immersive setup is the direct interaction of participants in reality. While the collaborators in an immersive setup always have to rely on more or less satisfying representations of each other, ranging from disembodied hands or heads to complete bodies visualized in plausible poses, users of an augmented scenario always are able to directly see each other and the interface devices they are using. This combination of reality and virtuality leads to the problem of correct occlusion between real and virtual objects, which of course does not exist in an immersive environment.

Even when using semi-transparent HMDs, where virtual objects only appear as transparent overlays over reality, wrong occlusion can hide gestures or facial expressions of participants. When applied in a

video-based augmentation setup virtual objects can completely hide real objects if not occluded properly. In Figure 28 the geometry intended to be displayed between the users hands and the PIP seems to be floating above the PIP. More confusion arises when moving the pen: ist virtual counterpart correctly occludes the PIP, while the real pen is almost invisible.

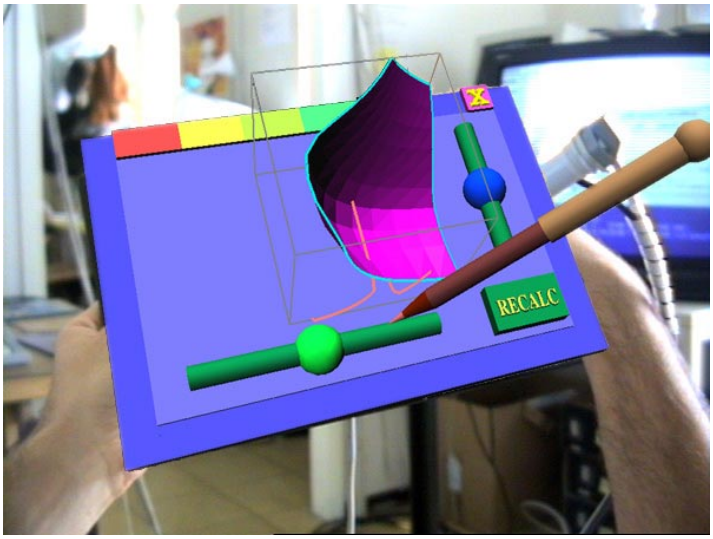


Figure 28: PIP geometry wrongly occluding users hands

Even discounting the importance of the social interaction, wrongly occluding virtual objects subject the users brain to conflicting depth clues: the parallax of the stereo-rendered objects gives a farther distance than the one inferred by the occlusion of real objects. This not only leads to misconceptions of spatial properties by the user, resulting in errors when trying to grab objects, but also increases eyestrain and the probability of motion sickness.

10.2.1 Influences of the Display System

As already mentioned, the properties of the display system influence the severity of the occlusion problem, as outlined in the following table:

Table 4. occlusion order vs. display system

occlusion order / display system *(example)*	virtual object occluding real object	real object occluding virtual object
back-projection/ screen based *(CAVE [Cruz-Neira1992],responsive workbench [Krüger1994])*	impossible	inherent
semi-transparent HMD *(Studierstube [Schmal-stieg1998])*	inherent	semi-visible / software solvable
video + immersive HMD *(UNC [State1996b])*	inherent	software solvable

In screen-based augmented environments - fishtank scenario - or projection based setups - like the CAVE - occlusion of virtual objects by real ones is simple and straightforward: real objects are always between the display surface and the eye and therefore always occlude virtual objects. This yields excellent results as long as no real object is placed placed behind (nearer the projection screen) a virtual one. In this case the real object incorrectly occludes the virtual object in front of it. The only exception for this would be the projection-based virtual office [Raskar1998], where front projection is used. Since the virtual objects are projected on top of the real ones - normally on walls or desktops - real objects can be excluded from this projection by projecting nothing (displaying black) in the relevant portion of the display area. This could for example be used to exclude pictures or windows from being projected onto, but also - when using the proposed optical tracking mechanism - to exclude the users hands from being projected on. Nevertheless the occlusion of the display surface by real objects still manifest: users hands and arms for example may drop shadows on the surface, thereby occluding parts of objects virtually in front of the shadow.

When using HMDs, the display surface is always between the eye and real objects. Without further processing virtual objects always occlude--- real ones. The only difference - albeit only gradual - exists between see-through HMDs utilizing semi-transparent mirrors and immersive HMDs being fed video images by headmounted cameras: the first only overlays semi-transparent computer-generated images over reality while the second one may display completely opaque objects.

Since the only case where the occlusion problem is solvable we concentrate on setups where the displayed virtual objects overlay images of reality. This can of course also be a desktop-based system, where video images are displayed overlaid by graphics.

10.2.2 Influences of the Tracking System

Tracking of users (heads), input devices and real objects to be augmented is a major task which influences strongly the quality of the augmentation. Immersive environments can tolerate discrepancies between reality and computer generated images which would be impossible to ignore in an augmented setup. Since in an immersive situation the user only relies on computer generated images for hand-eye coordination, errors between hand position in reality and projected hand/cursor position in the environment almost never lead to problems when interacting with the virtual environment.

In an augmented environment however, misalignment between tracked real objects and their representations in virtuality - which do not have to be visual representations: an input wand may only be represented by its "hotspot", the point in space where its function takes place - can cause severe problems for the user to operate in the environment. Additionally, lag between reality and the computer-generated environment is much more noticeable than in an immersive situation, since the position of virtual objects in respect to the real surroundings can be immediately compared. This results in "swimming" behavior of the virtual scene, which may also lead to motion sickness.

When addressing the problem of occlusion, another quality of the tracking system used matters: the ability to supply the simulation

with additional information regarding the occluding objects. Ideally, we would like the tracking system not only supply us with the position and orientation of one or more reference points on the occluding object, but also to deliver complete geometric information which enables us to determine which parts of virtual objects to occlude.

This yields the following classification by tracking system:

Table 5: occlusion order vs. display system

Supplies only positional data	supplies additionally geometric information
Magnetic tracking Mechanical tracking Optical tracking using beacons [State1996b]	video tracking delivering depth map from stereo [Wloka1995] video tracking delivering contour data [Berger1997] video range tracking [Raskar1998] using invisible structured light laser range tracking

Tracking systems providing occlusion data

A very efficient and self-contained approach would be video-based augmentation using stereo headmounted cameras and a HMD. Stereo video data could be used for inside-out tracking of the users position and orientation and also to generate a dense depth map of the visible scene usable for occlusion. While this approach seems to be the most promising in respect to versatility, it has still to overcome some drawbacks. The computational complexity of the depth map reconstruction allows only coarse approximations of the scene depth to be computed in real-time. Wloka and Anderson [Wloka1995] only produce coarse approximations of occlusion in acceptable time. This will probably be solved with increasing computing speed of future systems. Another drawback, which cannot be solved without additional tracking information is the line-of-sight problem: objects not visible in the video images cannot be tracked. While tracking of these objects is not needed for solving occlusion, tracking of the users hand outside the viewing frustum may be necessary. Especially when using two-handed interaction as we do when using the Personal Interaction Panel (PIP), the user must be able to use his proprioceptic sense alone

to manipulate virtual input devices without relying on his hand-to-eye coordination. This manipulation outside the users field of view is one of the essential advantages of two-handed interaction and avoids display cluttering and occlusion by interaction elements.

Berger [Berger1997] presented a contour-based approach to occlusion which delivers outstanding results while using essentially only 2D image processing methods. This reduces the computational costs drastically, but also suffers from the line-of-sight problems mentioned above.

Approaches using special hardware, as for example laser range scanning devices, while providing excellent data are in most cases prohibitively expensive and suffer in many cases from line-of-sight problems similar to the ones cited above. This holds especially in a collaborative situation where users standing close together examine and object between them, as in Studierstube.

Tracking systems providing only positional data

At the moment most virtual environments -immersive and augmented- use tracking systems providing only position and orientation. This includes commercial magnetic and mechanical tracking devices and advanced beacon-based optical tracking systems like the one developed at UNC [Ward1992] or the structured-light approach of [Raskar1998].

Sophisticated optical tracking systems like this, or hybrid optical/magnetic solutions like the one used in [State1996b] deliver high precision tracking data but unfortunately no additional data usable for occlusion.

10.2.3 Registration Problems

The problem of registering the users head position with the position of the virtual camera and the virtual objects with the real environment has already been covered extensively [Taylor1993, Azuma1994, State1996a, Whitaker1995]. Here - like in the previously cited approach to occlusion by [Berger1997] - a 2D image-based approach pre-

sented by Kutulakos [Kutukalos1998] seems to be an adequate solution for video-based systems.

The registration errors present us with especially annoying artifacts when occluding virtual objects. Slight errors produce a visible gap between virtual and real object (Figure 29) or overlapping effects which occlude a slice of the real object that is supposed to appear in front of the virtual.

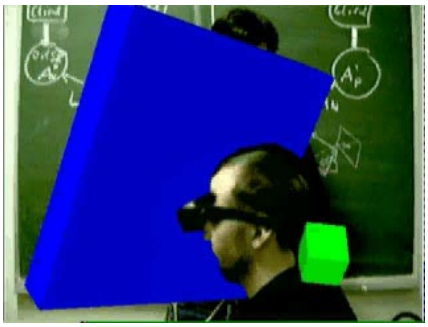
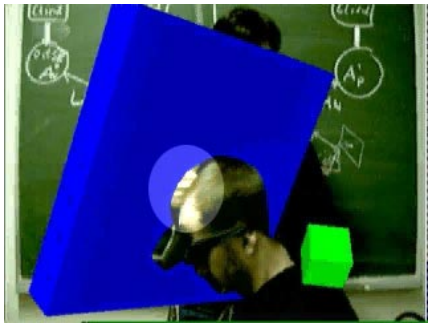
a)  b) 

Figure 29: (a) Occlusion with head phantom,
(b) misregistration (in circle) due to time lag.

Since registration of the commonly used magnetical trackers is notoriously instable, these errors tend to appear in most augmented environments. We try to present a solution for reducing the visual impact of these artifacts.

10.3 Requirements for Occlusion in "Studierstube"

The collaborative augmented environment we have developed allows multiple collaborating users to simultaneously study three-dimensional scientific visualizations in a "study room" - German: "Studierstube". Each participant wears an individually head-tracked see-through HMD providing a stereoscopic real-time display. The use of individual displays and head tracking allows providing stereoscopic, undistorted images for every user. The see-through property of the HMDs allows users to see each other and avoids the fear of bumping into obstacles.

Objects in Studierstube may appear - unlike in projection based display systems - both between and beside users. Interaction and collaboration within arm reach of the users is possible and supported by scientific visualization applications we have presented in [Fuhrmann1997] and [Fuhrmann1998]. Since the resulting scenarios lead often to situations where users heads, hands or bodies or the PIP had to occlude virtual objects, we had to find a method for efficient handling of these occlusions.

The properties of Studierstube and our chosen hardware and software setup presented us with the following requirements for solving the occlusion problem:

- View independence supporting multiple users

Studierstube has to support multiple users. Occlusion methods therefore have to support a view independence of methods, ideally holding the computational cost proportional to the number of users.

- Minimize number of trackers

Representation of users bodies has to be supported with a minimal number of additional trackers. Otherwise the tracking requirement of multiple users could quickly exceed our hardware capabilities and the users tolerance to setup overhead for use of the environment.

- Minimize rendering overhead

Since occlusion is only annoying when it does not work, the tolerable overhead for correctly simulating it is relatively low. We have to keep rendering passes, additional geometry or image-processing to a minimum.

Since our Studierstube uses - like most virtual environments at the moment - commercial magnetic trackers we need another approach to the occlusion problem which does not rely on tracking of geometry or depth.

10.4 Occluding with Phantoms

Since we are using a tracking system, which does not supply geometric information of occluding real objects, we have to acquire this in-

formation previously. This can easily be done by modeling or digitizing sufficiently precise representations of the objects offline and placing them inside the virtual scene.

We call these mockups of real objects "phantoms". By rendering them in an appropriate way occlusion of the virtual objects by real objects can be simulated (Figure 30). Rendering of phantoms can be performed differently, depending on the display system used:

Semi-transparent HMDs only need black areas where reality should be visible. To correctly occlude objects in this setup phantoms have therefore only to be rendered in black, without any special rendering order. This is fast and easy to implement.

When combining images for a video-based approach externally using a luminance- or chroma-keyed system the same method can be applied. For presentation purposes or when generating movies of our environment we usually apply digital compositing using one machine - an SGI Octane - for simultaneously digitizing video data and rendering virtual objects overlaying the video information.

The previous rendering approach would only produce a black avatar in front of the video image. To correctly render occluding geometry we had to restructure the scenegraph to guarantee all phantom would be rendered before any visible geometry. This can be done in a preprocessing step and does not influence rendering times. The following OpenGL sequence is executed for each frame:

Table 6. OpenGL rendering sequence for occlusion

reset z-buffer ↓
render video background ↓
enable z-buffer testing & writing, disable RGB writing ↓
render phantoms ↓
enable RGB writing ↓
render visible part of scene

This results in "invisible" phantoms, which only are registered in the z-buffer of the rendering hardware. Normal geometry is only rendered where it lies nearer to the viewpoint as a phantom.

10.4.1 Occlusion of Static Objects

The simplest case of occlusion of real by virtual objects is when the real objects are previously known and static over the duration of the simulation. Examples for this would be furniture and fixtures of the room used as workspace.

Examples for occlusion of static objects can be found in [Bree1996]. We have applied this method to simulate occlusion of virtual objects by laboratory furniture in Studierstube.

10.4.2 Occlusion of Tracked Rigids

The occlusion of virtual scenery by non-stationary rigid real objects can be handled in much the same way as the above case of static objects. The position and orientation of the phantom - since they are no longer constant over the duration of the simulation - have to be coupled to the real object, in our setup by the mounting of an additional magnetic tracker sensor on each occluding object.

We have used this approach to model occlusion of virtual objects by tracked real objects like the PIP in Studierstube (Figure 30).

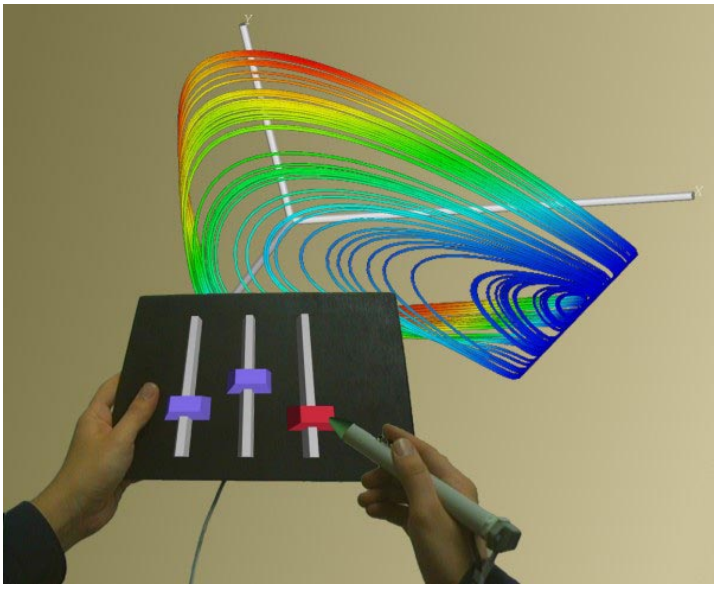


Figure 30: Personal Interaction Panel (PIP) occluding scientific visualization.

10.4.3 Occlusion for Tracked Articulated Objects

A more general case than tracking simple rigid objects for occlusion purposes is the use of tracked articulated objects for occlusion. The primary application of this in our environment was the occlusion generated by participant's bodies moving in front of virtual objects.

To achieve this under the requirements stated above we modeled a coarse articulated representation of a user. This approximation is

used as a "phantom avatar", which is supposed to mimic all poses and gestures of the tracked user to allow rendering of correct occlusion. It consists of rigid segments corresponding to body parts, which have to be animated in real-time according to incoming tracker data. To achieve this we have to mount additional trackers on the user. (Figure 31) shows how this was done. Since the head position is already tracked via the HMD, only three additional tracker sensors were used: two on the lower arms and one on the back of the user, a scheme comparable to the one used by Badler [Badler1993]. Similar results have been obtained by Waldrop [Waldrop1995] with the use of three sensors per arm for implementing avatars in a distributed virtual environment.

Figure 31: User with 4 sensors and HMD.

Based on a reduced number of sensors our simulation has to implement a number of additional constraints regarding joint stiffness and degrees of freedom to correctly simulate the posture of the user. Since our system is based on rigids we also are not able to simulate deformations of the body, but are planning to implement this in a later version based on a simplified, segmented spine.

10.4.4 Compensate Registration Errors by "Blurring" the Phantom

As already stated before, slight registration errors between phantom and real object result in annoying artifacts when occlusion occurs. This can be caused by errors between tracker data and real position of the sensor, due to time lag or misregistration of real and virtual environment or by differences between the shape of the user and phantom geometry. The first two causes can be addressed in one of the ways cited above, whereas the last case defines a new problem.

While theoretically possible, a pixel-perfect representation of for example the users forearm may be possible to model, such precision is in most cases unnecessary. Even discounting remaining errors from the tracker misregistration deformation of body and even changed clothes can invalidate the precision in modeling the shape of a body part in a fixed posture. Furthermore a precise and therefore finely tessellated model of the avatar would consume too much of our polygon budget.

We are therefore proposing to represent this margin of error in our phantom avatar. We do not render a hard edge between occluded object and occluding phantom but a soft transition covering the error margin (Figure 32). This transition shall enable the user to perceive details of both virtual object and occluding real object, while not significantly reducing the visual occlusion clue.

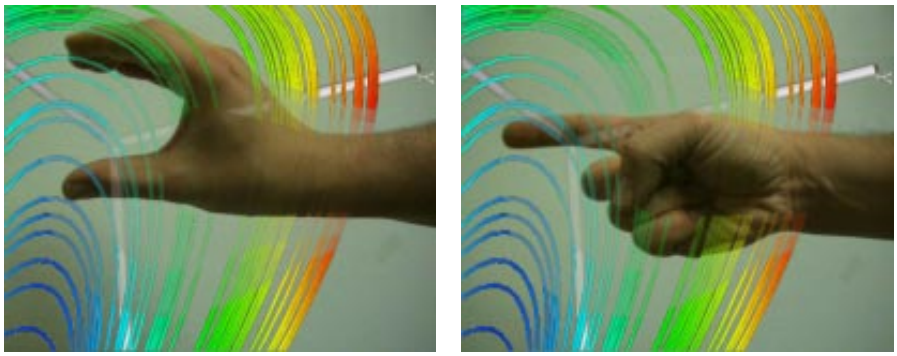


Figure 32: Use of a blurred phantom with different gestures.

A further application of this transitional "blurring" effect are body parts where much additional tracking effort would have to be spent just to implement precise occlusion: the users hands. Without using

the almost always cumbersome and inaccurate gloves to track exact hand and finger positions, not enough data is available to correctly occlude this part of the users body. When using some kind of "probability blurring" in this area, which renders the avatar the more occluding, the higher the probability of a part of the hand appearing in it is, we are able to provide satisfactory occlusion (Figure 33). Since finger and hand gestures are desirable extensions of collaboration in our environment we want each user be able to see them.

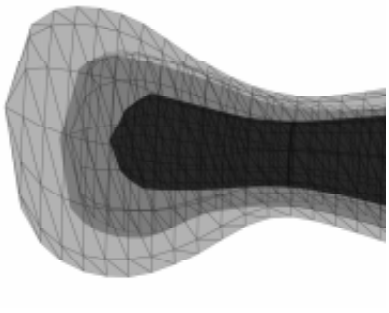
a)  b) 

Figure 33: a) "Probabilistic" function of hand,
b) corresponding phantom

10.5 Implementation

10.5.1 Integrating Occlusion into Studierstube

The integration of the necessary extensions for occlusion into Studierstube proved to be relatively straightforward. The whole system is based on the OpenInventor [Strauss1992] Rendering/Simulation library, which reduces the addition of new functionality to the Studierstube environment to the simple process of adding a new dynamically loadable module. The necessary tracker data is delivered by the Studierstube tracker interface, which delivers positions and orientations in OpenInventor fields, a high-level interface element, which allows easy connections to the separately developed simulation library [Löffelmann1997a].

10.5.2 Kinematic Simulation of Avatar Posture

The phantom avatar used for occlusion is modeled as articulated solids representing a human body. A geometry used for occlusion is attached to each solid. Some of the solids are directly connected to the trackers attached to the user (Figure 34).

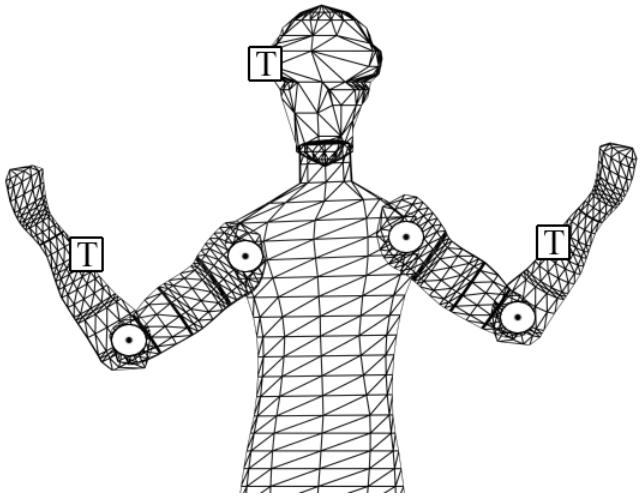


Figure 34: Phantom avatar with position of tracker sensors ("T") and joints (points).

Dynamics is used to animate the avatar. This allows us to intuitively rigidify the degrees of freedom of the joints using damped springs. An upward force applied to the body maintains a standing position. The resting positions of the springs define the natural, comfortable posture of the avatar.

Our simulator, detailed in [Faure1998], solves the inverse kinematics problem while taking into account masses and forces. Forces are applied at the beginning of each time step. The motion is then integrated over time, leading to new positions. In addition, the position of the solids bound to the trackers are set accordingly to the input data, regardless of forces or velocities. Possible violations of geometric constraints induced by forces, time integration and tracker input are then cancelled by computing appropriate displacements of the other solids.

The displacements are computed by solving the dynamics equation

$$(JM^{-1}J^{T})\lambda = b \qquad (1)$$

where matrix M is a block-diagonal matrix which represents the mass of the system and J a sparse matrix which codes for the geometric constraints. Vector b represents the geometric errors to cancel. Vector λ gathers Lagrange multipliers of the constraints. They act much like integrated forces. The corresponding motion corrections are are thus M-1JTλ. Velocity updates are deduced from position corrections by applying Stoermer's rule [Press1992]. This integration scheme considers displacements instead of velocities. This avoids us to differentiate the possibly noisy tracker data.

An iterative solution of equation (1) is performed. A minimization of the norm of the error is performed over the search space defined by the components of vector λ. This approach has two useful features for our application. First, in presence of overconstrained solids, such as the upper arms of the character in figure 5, a compromise between the constraints is found. Second, the iterative minimization allows the user to trade-off accuracy for computation time, making the approach suitable for the real-time animation of complex scenes.

10.5.3 Implementing "Blurred" Phantoms

To implement the "blurring" we want to exhibit the avatars, we have essentially two approaches of realizing this effect:

Image-based blurring

The first method that comes to mind when an operation like "blurring" is requested is of course an image-based approach. Rendering an avatar blurred by some kind of convolution operation seems to be the simplest way to implement the desired effect.

But a normal convolution using only software for a video-resolution image may not be completed in real-time (e.g. for frame rates >10Hz) without the use of special hardware. Convolution techniques utilizing OpenGL hardware as presented in [McReynolds1998] may be fast enough, but depend in our case on the ability of moving data fast be-

tween image buffer and texture buffer, which is not present in all OpenGL [Woo1997] implementations. Using the OpenGL accumulation buffer would provide an elegant solution for this problem, but unfortunately would require N^2 rendering passes of the whole scene for an N×N convolution matrix.

Furthermore certain properties of the image-space approach may not result in the desired appearance of the occlusion: we want the margin of error to be specified in absolute, real-world measurements, to allow for a fixed error produced by e.g. different clothing or hairstyles. An image-based approach would deliver the same amount of blurring for near and far parts of the avatar. The above mentioned "probabilistic blurring" would also not be easily implementable using a single convolution over the avatar.

Object-based blurring

Another approach to the implementation of smooth transitions for occlusion is to implement a "blurred geometry" of the avatar. This consists not of a single surface representation of the geometry representing a body part, but a layered approach of successively bigger and more transparent shells around the geometry (Figure 33a). These shells can be modeled as the same geometry rescaled around the centroid of the object, since most of the avatars geometry is convex.

Of course this approach does not yield a "smooth" transition, but one consisting of a discrete number of steps in occlusion, but it should be sufficient to implement a reasonable small number of steps - say three to ten - to obtain satisfying results in most cases. Since this geometry undergoes the same perspective transformations as everything else, the error margin can be specified in real-world measurements as opposed to the image-based approach above.

A further advantage of this method is the ability to implement "Levels of Detail" for phantoms: with increasing distance we can "switch off" more and more of the intermediate shells, to reduce geometric complexity and therefore polygon count for objects farther away. This allows us to vary the number shells and of transition steps according to distance. In distant phantoms only one or two transitions, covering maybe an equivalent amount of pixels of distance between two shells

in image space are rendered. Phantoms nearer the observer are represented by more shells, resulting in approximately the same number of pixel per transition step. This behavior is readily implemented using the standard OpenInventor LOD-node, which references more or less of the scaled versions of the same geometry representing the avatar.

10.5.4 Results

Registration of the avatar geometry to the user proved to be a relatively time consuming process. Since misregistrations of the tracking sensors' position relatively to the geometry resulted not only in visual artifacts (Figure 29b) but also in artifacts of the simulation, registration had to be done very carefully. Once registered, the main problem was an additional time-lag introduced by the kinematic simulation, which could only partially be corrected in this version. The visual results in a typical scientific visualization situation as in (Figure 30) were satisfying and gave users a much better overall impression of the situation and the spatial relationship between real and virtual scene.

The object-based implementation of "blurred" occlusion was easy to implement and gave in the cases where hand-gestures made rigid modeling difficult a satisfactory compromise between correct occlusion and tracking effort (figure 2 and 3). It produced noticeable artifacts along the intersections between rigid limbs, which can be corrected by using stencil masks when rendering the different shells, a correction which will be implemented in the next version of our occlusion system.

The main difficulty for the animation of the avatar is to have it accurately overlap the image of the user (Figure 35). As mentioned before, the geometry has to be carefully adjusted to the body of the user (Figure 36). Additionally, the positions of the trackers and of the joints with respect to the geometry have to be precisely modeled. This is done interactively before performing the animation. Joint positions are difficult to adjust because they are inside the body of the user, thus invisible. We adjust them iteratively using different postures.

Figure 35: User with overlaid phantom avatar geometry.

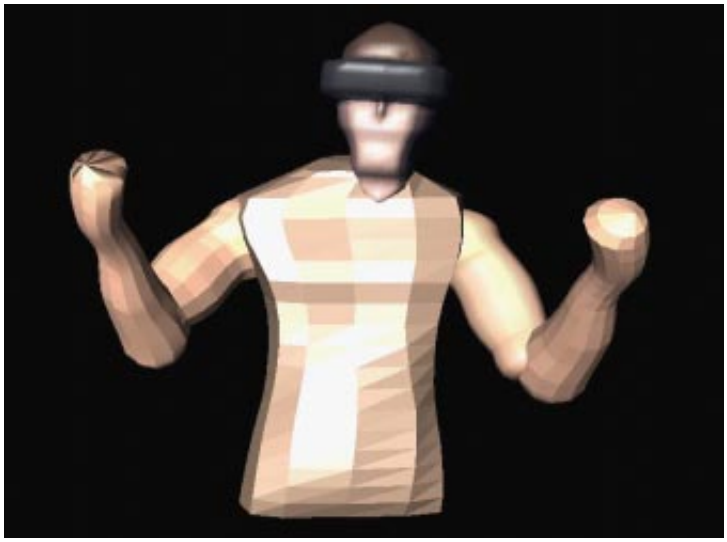


Figure 36: Same user as above represented as avatar.

Figure 37: Occlusion of a background polygon.

There are several sources of error in the final result (Figure 37). The geometric models do not fit perfectly to the body of the user. The human body is a much more complex structure than our avatar. Limbs are not rigid, especially when covered with cloth, and the joints of the human body differ from our simple joints. If the calibration of the virtual camera does not fit perfectly to the real camera used for video input, the avatar may overlap the user in some postures but not in others. Future work include a more realistic body structure, especially for shoulders, back and neck, and semi-automatic calibration of the virtual camera.

11 Studierstube as a Frontend for Scientific Visualization

11.1 Overview

Scientific visualization is a tool, it provides the means to extract specific information from gathered data or a simulation of real phenomena. But not every person capable of formulating a problem and interpreting visualization results is also capable of wielding this tool. Therefore scientific visualization frequently requires experts with different background to cooperate closely, in particular because many valuable insights only occur in face-to-face discussions over the relevant data. We believe that augmented reality (AR) [Feiner1992], which combines a familiar physical surrounding with the visualization of synthetic data, represents an ideal working environment for collaborative visualization.

In this chapter we show how a combination of AR and a visualization system [AVS92], results in STUDIERSTUBE becoming a three-dimensional user interface for scientific visualizations. Several examples constructed in DynSys3D [Löffelmann97a] - developed for the visualization of complex dynamical systems in AVS – conclude this chapter.

11.2 Studierstube/AVS Interface

To combine augmented reality and scientific visualization, a new integrated solution could be developed, but employing an existing, general-purpose scientific visualization system - in our case, the Advanced Visualization System (AVS) - allows a wider spectrum of applications and eases development [AVS1992]. Since this desktop-based system is not designed for the real-time requirements of AR, we use decoupled simulation [Shaw1993]: The visualization system and the AR user interface (called display server in an analogy to X-Windows) run as completely independent processes, typically executing on separate machines connected by a LAN. As shown in figure 2,

the system is composed of two loops: the display loop, a tightly coupled human-in-the-loop component, where real-time response is essential, and a loose coupling between display server and visualization application for the exchange of visual information. Since the display server continually updates the images sent to the HMDs, delays of several seconds for recomputation of the visualization do not impair the use of the VE.

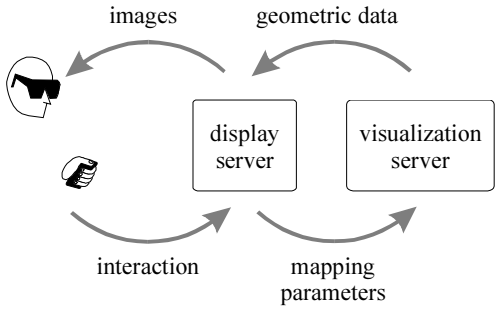


Figure 38: Decoupled simulation model

Our objective was to enable users familiar with AVS graphical way of constructing dataflow networks to quickly integrate the AR system into their visualizations, similar to the work presented in [Cruz-Neira1993b, Roy1994]. We implemented a set of interface modules which automatically establish network connections to the display server and transfer data between the AVS dataflow-network and Studierstube (figure 4). These included an AR-output module and AR-input modules for every interaction method: slider, dial, button and pen click- and drag-events. To reroute output geometry to Studierstube, the user drag an AR-output module into his AVS-network and connects it like a standard AVS component (figure 3) to any output supplying geometry data. For the generation of different visualization icons which are updated independently, more than one AR-output module is used. When using customized views we can change the default behavior of this modules from displaying data on all HMDs to a different selection of users per module. Similarly, input modules can be connected to the input parameters of the visualization. For sliders or other 2D input elements in this way we can specify which user may access them via his PIP. The LAN-connections to the display server and the administrative exchange of parameters like slider extrema are performed behind the scenes by a separate process in an AVS co-routine. Changes in the AVS-network are therefore instantly reflected

in the configuration and display of the corresponding elements of Studierstube.

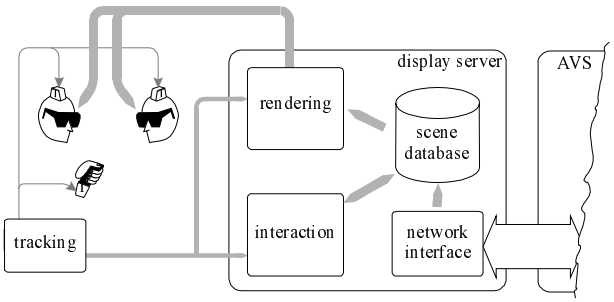


Figure 39: Display server and hardware configuration

11.3 Collaborative Visualization

As stated before, scientific visualization often implies collaboration of a group of experts. A common scenario in our visualization department is for example a visualization-expert and a mathematician working together on the visualization of a dynamical system, like depicted in Figure 40. The mathematician supplies his knowledge of the simulation and knows what features he is looking for and the visualization expert applies various modeling methods and parameterizes the mapping. In case of a commercial project the group often contains - at least in the last phase of work - one or more customers to whom the finished work is presented.

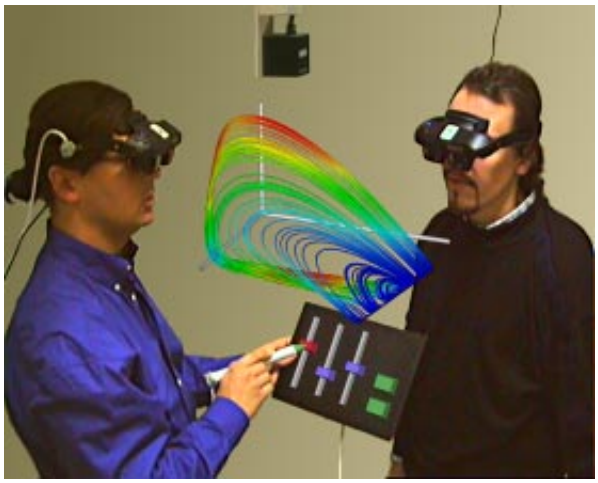


Figure 40: Collaborative work in *Studierstube* (Mixed Mode Oscillations)

To support this working method, our environment has to supply the following features:

- social interaction
- input devices and paradigms which allow for shared and private interaction
- interaction with the visualization
- customized views to display only data relevant for a specific user

In most collaborative situations, direct conversation between persons is preferable over every other medium. When a group of people has to work together on one problem, every electronic layer between them acts as an obstacle to efficient exchange of ideas. Due to the augmented nature of our VE, users can see each other even if parts of the faces are hidden by the HMD - and can perceive their relative positions and gestures in a way which cannot be efficiently replicated by an avatar. "Can you turn the object this way?" accompanied by an appropriate gesture is sufficient to convey the request to another user. A completely immersive environment frightens and disorients novice users more than AR.

11.3.1 Interaction with the Visualization

All geometrically interpretable input e.g. seedpoints for streamlines or other position information is done by clicking or dragging the 3D mouse in the Studierstube's workspace. In this way a direct manipulation of visualization output can be performed. The display server provides instant local feedback to the user in form of points or lines as markers for the completion of the gesture (figure 5).

11.3.2 Personal Interface

Our main input device for non-geometrical interaction with the visualization is the PIP. All numerical or state information is input via conventional 2D user elements on the PIP. These are allocated by

connecting the corresponding input module to the AVS dataflow network. An additional parameter specifies on which users PIP the input element appears, thus putting the input element literally into the hands of the appropriate user. Thus every experts PIP is supplied only with the input elements that are relevant for him.

11.3.3 Customized Views

Since our scenario postulates the collaboration of different experts, it is likely that not the whole information necessary for one expert should be presented to all users. The visualization-expert for example may want to display a streamsurface as wireframe to check the tesselation, whereas the mathematician may want to map local properties like torsion to the color of the surface. Our AVS output modules take as additional parameter whether the data routed to the environment should be presented to all or only to some users.

11.3.4 Reconfiguration of Visualization

During most visualization sessions at least a partial reconfiguration of the dataflow network takes place. It may be the introduction of a new visualization-icon or only the wish to manipulate a former not directly accessible parameter via the PIP. Such reconfiguration cannot be accomplished by using interaction from within STUDIERSTUBE but must be done on the AVS desktop interface. Since all changes can be made on the running system, the other users may continue working on the visualization while e.g. the visualization expert sits down on the workstation to reconfigure the network. Even in this case interaction between users is possible: Firstly small changes are possible while wearing the HMD, so a quick look over the shoulder into the VE shows the results, secondly comments from the other users can steer the reconfiguration work.

11.4 Visualization of Dynamical Systems

DynSys3D is a multi-purpose workbench for the rapid development of advanced visualization techniques [Löff97b, Löff97c] in the field of three-dimensional dynamical systems, designed to support incremental and parallel implementations of new ideas in this field.

The system is based on AVS [AVS92], which is a general purpose visualization system based on data flow paradigm.

DynSys3D modules are usually built from at least three principal components: Dynamical System, Numerical Integrator, and Visualization Technique. They are separated by rather narrow interface specifications. By this separation, for example, new dynamical systems can be visualized with approved visualization techniques without recompiling them. Similarly, a new numerical integrator can be plugged into this system without touching any visualization module. Currently, our most important visualization icons – streamlines, streamsurfaces and dashtubes [Fuhrmann1998] - and a set of specialized input modules – 2D sliders and buttons and 3D feedback - are implemented.

One design guideline of DynSys3D, namely that all of its modules have to produce standard AVS output (geometry), was very important for the integration of DynSys3D and Studierstube: A simple utility that converts AVS geometry into the Studierstube's format (Open Inventor) was sufficient to export geometry. Interaction messages from the VE are sent to DynSys3D's input modules as AVS geometry items, points for 3D mouse positions, lines for mouse drag events (color plates 2-4).

Governed by the data flow paradigm underlying AVS, the user's commands are routed via the display server to input modules of an AVS network, whereas the results of the computation are sent to output modules, and from there onward to the user (Figure 42). However, AVS' execution scheme invokes modules only when the user generates input events. Consequently, the network communication with the display server had to be implemented as a coroutine inside AVS, which forwards the user's commands to an AVS input module. This explicitly triggers re-evaluation of the AVS net; the coroutine also collects

the results (usually new or modified geometric data) and forwards it to the display server.

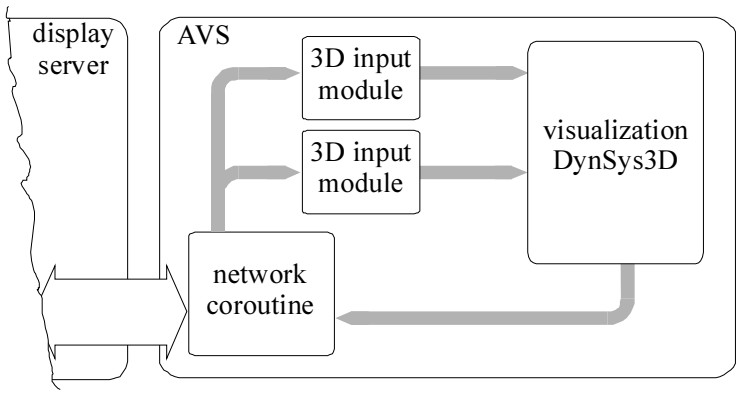


Figure 41: AVS/DynSys3D configuration

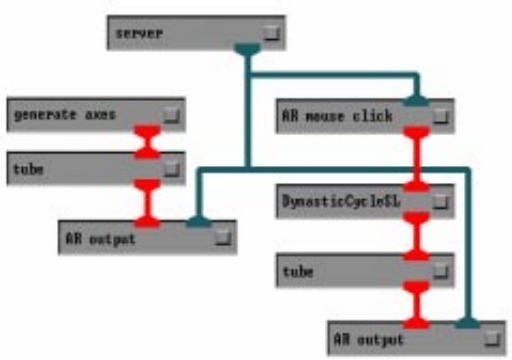


Figure 42: An AVS network for interaction via Studierstube

12 Real-Time Techniques For 3D Flow Visualization

12.1 Overview

Visualization of three-dimensional steady flow has to overcome a lot of problems to be effective. Among them are occlusion of distant details, lack of directional and depth hints and occlusion. In this chapter we present methods which address these problems for real-time graphic representations applicable in virtual environments. We use dashtubes, i.e., animated, opacity-mapped streamlines, as visualization icon for 3D-flow visualization. We present a texture mapping technique to keep the level of texture detail along a streamline nearly constant even when the velocity of the flow varies considerably. An algorithm is described which distributes the dashtubes evenly in space. We apply magic lenses and magic boxes as interaction techniques for investigating densely filled areas without overwhelming the observer with visual detail. Implementation details of these methods and their integration in our virtual environment conclude the chapter.

12.2 Introduction and Motivation

Many visualization techniques for two-dimensional flows have already been investigated in detail [Post1993]. Visualization of 3D flow phenomena, however, tend to produce complex images with often heavily overlapping geometry. Occlusion, ambiguities in depth and orientation of flow strain the viewer's abilities to interpret the visualized data.

The main point of this chapter is to show how real-time graphics in a virtual environment can be used to overcome some of these problems. Stereo cues, interactive and intuitive changes of the viewpoint and the feeling of immersion allow users to get a better impression of the structure of the 3D flow in a virtual environment as compared to a desktop system.

We present a combination of selected visualization and interaction techniques, which enable the user to rapidly explore complex 3D vector fields. Fast texture-based visualization techniques, which utilize the graphics hardware to get real-time performance, are applied to streamlines. A new parameterization scheme allows a direct mapping of a wide range of flow velocity to texture velocity without loss of detail. These techniques together with interactive 3D focussing enable the user to quickly identify and explore areas of interest. The focussed volume is selected with magic lenses and magic boxes that also use mainly hardware accelerated features. Animation is realized in the texture coordinate domain with moving opacity maps. This reduces occlusion and cluttering by simulating particle traces. An automatic streamline placement algorithm [Jobard1997] is extended into the third dimension to generate an even distribution of streamlines in the virtual environment.

12.3 Related Work

Several techniques for the visualization of 2D and 3D flows inspired this work. Some examples of texture based techniques for the visualization of 2D flows are [Cabral1993], [Wijk1993], [Stalling1995]. We already applied texture-based visualization techniques in [Löffelmann1997].

FROLIC [Wegenkittl1997a] is a variation of OLIC (Oriented Line Integral Convolution, [Wegenkittl1997b]) based on LIC [Cabral1993]. OLIC uses sparse textures and an asymmetric convolution kernel to encode the orientation of the flow in still images. Costly convolution operations as done in LIC and OLIC are replaced in FROLIC by approximating a streamlet by a set of disks with varying intensity. The visualization icons we call dashtubes are basically 3D streamlines with animated texturing and apply similar techniques to 3D flows.

[Interante1997] use LIC for 3D flow volumes. Halos around streamlets offer additional depth and ordering cues. The high cost of volume rendering, however, precludes an interactive exploration. Texture splats as discussed in [Crawfis1993] encode direction and orientation of 3D flows. Fast splatting operations are realized with hardware supported texture mapping. Animated texture splats illustrate the

flow dynamics. While these techniques produce good results, their rendering times are prohibitive for real-time applications.

[Max94] presented various techniques for visualizing 3D flows close to contour surfaces. Motion-blurred particles are generated in the vicinity of surfaces. Particles are started automatically on a lattice. Generation and deletion of particles is density based. Line bundles are realized as texture splats with antialiased lines as texture. Hairs are 3D particle traces originating on the surface. Additional information is encoded in the color, length and transparency of these hairs.

Streamline placement is an important task to achieve an approximately uniform coverage of phase space. An image-guided streamline placement has been presented in [Turk1996]. Another approach for creating evenly spaced streamlines uses a regular grid [Jobard1997]. For each grid element a list of passing streamlines determines whether there is still space for the placement of another streamline.

Queues of streamline vertices administer possible seedpoints for new streamlines. Tapering of streamline widths produce hand-drawing effects. Directional glyphs illustrate flow orientation. A 3D variation of the streamline placement in [Jobard1997] was used in our approach (see section 3).

12.4 Dashtubes: Streamlines with Animated Opacity

Streamlines are an intuitive way of visualizing flow. Their applicability in 3D-space however is limited, since they do not provide the visual cues needed. Normally, lines are rendered with the same width regardless of distance to the viewpoint so they lack perspective distortion, which is a significant cue for judging distance.

Additional techniques like halos [Interrante1997] are necessary to resolve the ambiguities of overlapping lines. When visualizing flow, streamlines need to be enhanced to convey the direction of the flow. This can be done by directional color variations or by placing icons along the streamline as shown in [Jobard1997]. Texture based techniques like LIC can be modified to include directional variations as we have shown in [Wegenkittl1997a] and [Wegenkittl1997b]. A more direct approach however is the visualization of flow by animation. In

the 2D case we use FROLIC combined with lookup-table animation to do this in real-time. [Bryson1991] has successfully used streaklines - 2D particles moving along the vector field - in the virtual windtunnel to animate flow in space. This technique depends on continually updating the position of all particles with every animation step, leading to a considerable consumption of processing power.

In this chapter we present dashtubes as visualization tool for steady 3D-flow. Dashtubes are generalized cylinders extruded along the direction of the flow (Figure 43). Their geometry is displayed with animated, opacity mapped texturing to visualize velocity and direction of flow. They appear as "dashes" - short opaque segments - moving along the direction of the flow.

Our requirements for dashtubes are:

- *Volume-filling properties*: the dashtubes have to exhibit an even distribution over the volume of interest. Otherwise the omission of interesting features would be probable.
- *Reduced occlusion*: we need a method which reduces the occlusion of distant features by features near the observer. This demand is almost the opposite from the volume-filling properties mentioned above.
- *Animation of flow*: the velocity and direction of the flow should be visible in the animation. Dash velocity should directly correspond to flow velocity.
- *Visibility of dashes*: the length of the dashes should not vary too much with velocity. High velocity areas would otherwise produce long dashes and gaps, which do not give the desired appearance, while low velocity would lead to very short dashes and eventually to aliasing artifacts.
- *Fast Rendering*: Since one of our main points is the real-time applicability, we want a fast, hardware-assisted rendering method. Like FROLIC we would like to use the graphics hardware to do the animation, leaving the CPU time for simulation and interaction. This is advantageous as the graphics hardware has anyhow to update the image continuously when rendering a virtual environment.

The design of dashtubes meets the mentioned requirements. To avoid the occlusion of distant parts of the visualization by closer features and to generate the desired effect of particles moving along the dashtube we render it partially invisible. This is done by using an opacity texture, which includes transparency information for the rendering hardware. Since semi-transparency does not work well in combination with z-buffered visibility resolution, we only map completely opaque or complete transparent values to the geometry. Thereby we avoid artifacts produced by the order in which we render different parts of the scene. The dashtubes are assigned texture coordinates, which correspond to a temporal parameterization along streamlines.

Figure 43: Streamline geometry without texturing

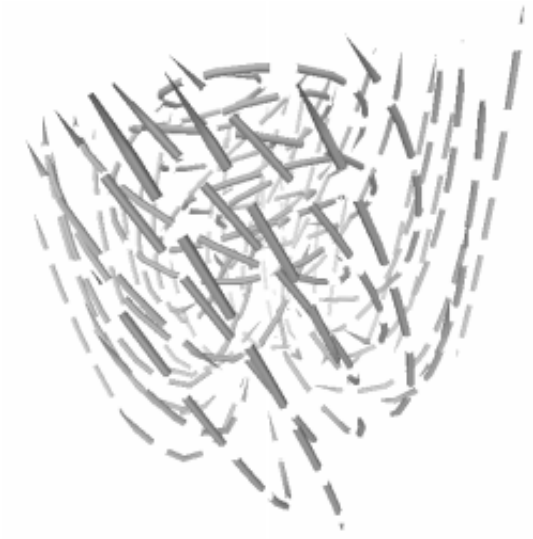


Figure 44: Dashtubes with opacity-texture

When combined with an appropriate opacity texture, this leads to the desired dashed appearance, with opaque dashes intermitted by empty sections (Figure 44). Since animating the texture image itself is a relatively time-consuming operation on most graphics hardware, we just transform the texture coordinates along the direction of the tube. In OpenGL this can be done by modifying the texture transform matrix, which has the additional advantage that it works even when more than one texture map is used. Animation is an essential part of the method, since otherwise structural information visible in Figure 43 would be lost in the opacity-mapped representation (Figure 44).

Early tests showed that the animated texture produces annoying visual artifacts at the ends of the dashtubes. The opaque segments entering and leaving the surface of the extrusion exhibited an irritating ”blinking” behavior, comparable to the pulsation we had to overcome in FROLIC. We treated this by reducing the radius of the first and last cross-section of the dashtubes to null, thereby tapering the extrusion at the ends (Figure 43 and Figure 44). This yields smooth transitions at both ends, comparable to a ”fade-in” effect.

12.5 Adaptive Texture-Mapping

As most texture mapping techniques, our method is prone to aliasing. When the length of one dash in image-space is reduced to a single pixel width or below annoying artifacts make the distinction of dashes difficult. Even worse, the speed and direction of the visualized flow becomes impossible to observe.

These effects appear when the viewpoint moves away from the texture-mapped tube or when the flow velocity is very low. In both cases a large texture area is mapped to a small region of the screen.

Therefore we need a method to reduce aliasing while preserving the essential properties of dashtubes: a high contrast texture moving along the tube at the speed of the visualized flow which contains distinctive opaque and completely transparent sections. Furthermore the maximum length of a dash and gap sequence must not be too big. Otherwise the appearance of the dashtube as a line of moving particles would suffer. Long dashes, while giving a good impression of the direction of the flow, occlude much of the scene farther away and reduce the impression of a volume flow. Ideally we want approximately uniform spaced dashes in image space, independent of viewpoint and flow velocity.

12.5.1 Mipmap Method

Most commonly the mipmap-method [Williams1983] is used to reduce texture aliasing. Thereby the texture is filtered to consecutively lower resolutions. A single texture is represented by a series of texture maps with decreasing size and texture-frequency (Figure 45a). Since this method does the filtering in a pre-processing step, the additional expenses at runtime are relatively low making it the method used by most real-time graphics hardware.

To apply mipmaps on dashtubes we have to alter the standard algorithm of producing the reduced texture maps. Normally each sub-map of a mipmap contains a filtered version of the level above, thereby halving resolution and maximum frequency of the texture (Figure 45a) This clashes with our rejection of semi-transparent objects in section 12.4: the contrast of lower resolution maps is reduced due to

filtering, which also produces semi-transparent areas. The flow velocity is preserved, but the dashes get shorter in areas with reduced velocity or a more distant viewpoint.

So we have to produce sub-maps which do not exhibit this undesired properties. Figure 45b shows an example how such maps could look like: They posses the fractal property of having the same amount of detail on every level.

Using such a mipmap produces the following effects: If the viewpoint moves farther away, the texture (the dashes) shrink continuously in length until a new mipmap level is reached, where the switch to a coarser resolution is performed and the dashes again have a distinguishable length. Sections of dashtubes where the low flow velocity would produce very short dashes are also mapped to coarser resolutions and therefore exhibit longer dashes (Figure 46).

An important aspect of this approach is that only the contents of the texture are switched, not the mapping of texture to object. This leads to the intended effect when animating the texture: the velocity of the texture along the tube directly corresponds to the flow velocity independently of the length of the dashes.

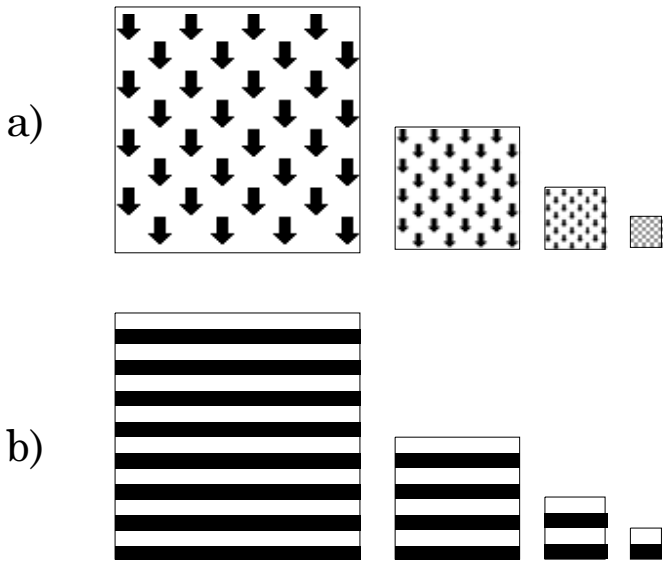


Figure 45: Mipmap textures:
a) conventional filtering scheme
b) mipmap for adaptive texture mapping

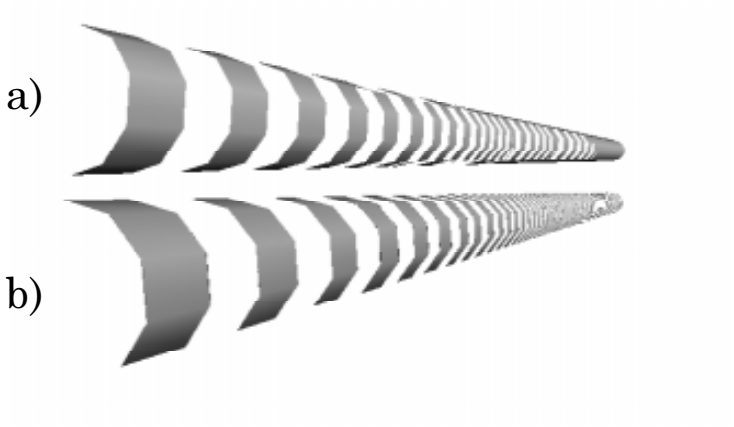


Figure 46: dashtubes with (a) and without (b) mipmap-texturing

The main problem when using mipmaps for adaptive texture-mapping arises from the small number of available mipmap levels: Since every two-dimensional sub-map has exactly the double resolution of the next lower level, the memory consumption exponentially rises with the number of levels. Additionally the switch between one level and the next leads to discontinuities in the texture appearance. Since the textures move along the tubes these discontinuities are especially annoying when they appear along a single tube.

Most mipmap implementations - including the one used in OpenGL - reduce the artifacts due to level switching by interpolating between two adjacent sub-maps. We can not apply this method in our case since this strongly reduces contrast because the interpolation is performed between two essentially different maps, not between maps only differing in the high-frequency components. Furthermore the interpolation produces semi-transparent areas causing the already in section 12.4 discussed problems when rendering into the z-buffer.

These problems occur since the transition between one level and the next is relatively coarse. If we could reduce the difference between level resolutions and increase the number of levels we would be able to exactly specify how switches between longer (less) and shorter (more) dashes are performed. The vast memory consumption of mipmaps with a high number of levels denies us the trivial solution of this problem, so we have to approach it differently.

In section 12.5.2 we show a different approach to adaptive texture-mapping, which overcomes some of the problems mentioned above.

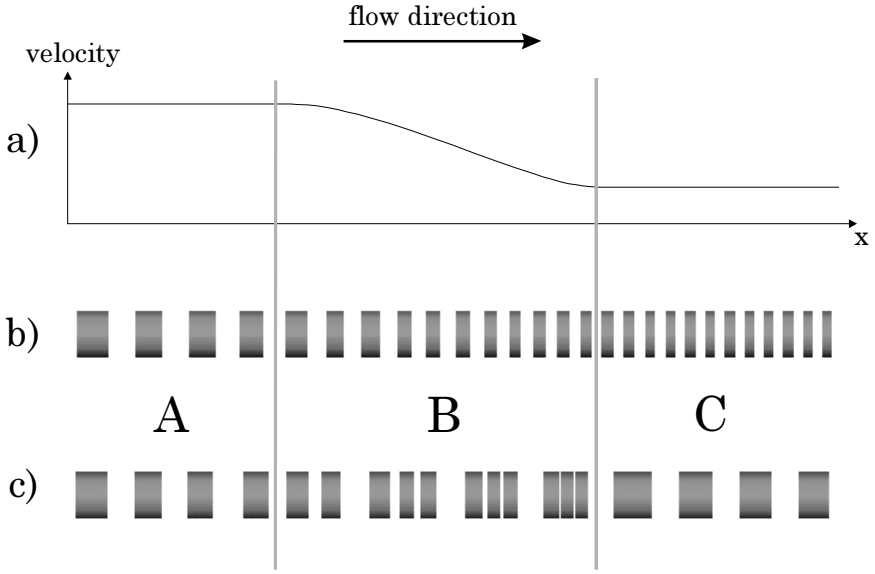


Figure 47: Adaptive texturing with texture-coordinate method

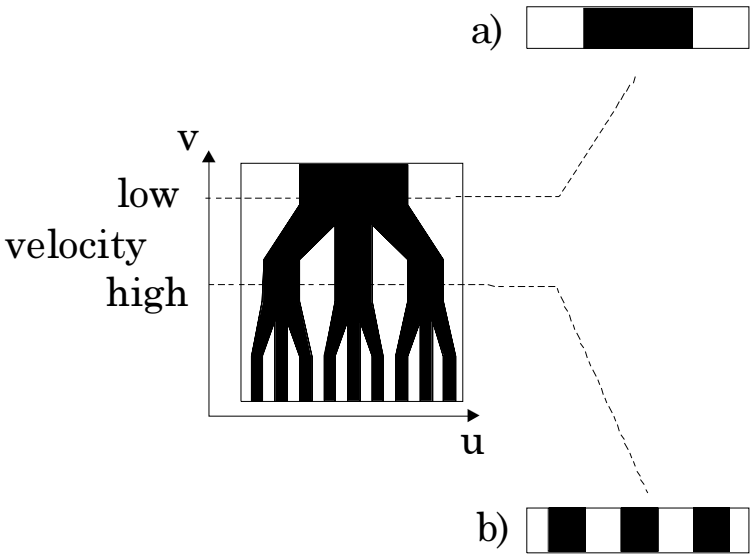


Figure 48: Texture usage of texture-coordinate method

Figure 49: dashtubes with (a) and without (b) adaptive texturing

12.5.2 Texture-Coordinate Method

If the flow velocity along a dashtube is falling as in Figure 47a, the resulting texturing leads to short dashes in the low-velocity region of the dashtube (Figure 47b, region C). This behavior violates our demands from section 12.4, where we require the dashes to be distinguishable independently of flow velocity. Since these short dashes can lead to undesirable aliasing artifacts, we would prefer longer dashes in region C, which should still reflect flow velocity. If we want longer dashes moving with the same speed as the shorter ones, we have to somehow reduce the number of dashes travelling from region A to region C. A possible solution, which reduces annoying artifacts shows region B in Figure 47c: every three dashes leaving region A get joined to a single dash while moving through region B. The low-velocity part (region C) of the dashtube contains one long dash in Figure 47c for every three short dashes in Figure 47b. When animated, this shows three dashes leaving the high-velocity region A of the dashtube, which reduce their gaps until they merge into one longer dash in the low-velocity region C.

To implement this behavior using conventional (OpenGL) texture-mapping hardware we have to alter the way we map a texture to the geometry of the dashtube. In ordinary texture-mapping applications two spatial texture coordinates are mapped onto the 2D surface of an object. Since the rotationally-symmetric dashtubes represent essentially 1D objects - the underlying streamlines - we only need one tex-

ture coordinate for the spatial mapping. We use the remaining texture coordinate to smoothly vary the dashes appearance.

Figure 48 shows a texture map, which maps velocity and time along the dashtube onto opacity. For constant velocity horizontal rows of the texture are mapped along the tube. Constant low velocity maps the lowest row of the texture onto a short segment of the dashtube (Figure 48c), whereas constant high velocity maps the uppermost row of the map onto a long segment of the tube (Figure 48a). Changing velocity along a segment samples the map along a slanted line (Figure 48b), thereby producing the effects depicted in Figure 47c, region B.

The merging dashes in the transitions exhibit slightly irritating artifacts when the gaps shrink to one pixel width. To reduce this effect, we want to map the transitions only to short segments of the dashtubes. Between this transition segments we allow the dashes to lengthen or shorten a bit, without altering their number. This gives the motion of the dashes a more uniform appearance.

The texture map in Figure 48 is designed to restrict the transition segments (Figure 47c, section B) to small velocity ranges. In the areas where the texture contains parallel, vertical stripes the sampled texture is independent of the velocity. Only when the sampling passes into one of the branching areas of the map dashes are joined or split.

12.6 Streamline Placement

When using streamlines for flow-visualization, the quality of the result depends heavily on the placement of the streamlines. Even when visualizing two dimensional flow fields a uniform distribution is desirable, but when extending the flow visualization to three dimensions the added complication of occlusions make an even placement essential. The start- and endpoints of streamlines introduce distracting artifacts into the visualization so we want to keep their number small. Therefore we want to populate our flow volume with evenly distributed streamlines of maximum length.

To accomplish this we extend the algorithm of Jobard and Lefer [Jobard1997] to three dimensions: They place streamlines using only local criteria. A new streamline can only be placed if the distance to

already existing streamlines does not fall below a certain minimum. Since the speed of the algorithm depends mainly on this distance test, certain techniques are applied to accelerate the test. Each streamline is approximated by a set of evenly spaced sample points. The distance between two streamlines is defined as the minimal distance between any of their sample points. This works reasonably well when the sample points are always closer spaced than the minimum distance between lines. A regular grid is used to reduce the set of points to be tested to the ones in the immediate neighborhood of a new point. The distribution of seedpoints depends on the desired density of the resulting images. For densely placed streamlines the seedpoints are distributed randomly, while for sparsely placed streamlines the seedpoints are introduced near the sample points of existing streamlines.

The adaptation of this algorithm to our needs was quite straightforward. We extended the grid to three dimensions, making it necessary to check now a maximum of 27 cells per distance test. Another technique presented in [Jobard1997], the agglomerative seedpoint placement for sparse distributions mainly produce visually appealing results in 2D. In 3D, where streamlines can pass in front of each other and their visual distance depends mainly on the viewpoint it produces no distinctive advantage opposed to random startpoint placement.

For this reason we chose to distribute the seedpoints on a jittered grid, a process which works faster than agglomerative seedpoint placement and produces acceptable results. Since streamlines which are short with respect to the dash length can produce irritating ”blinking” artifacts, we reject them as soon as they are introduced, therefore allowing other streamlines to grow.

12.7 Focussing and Context

One of the main problems when visualizing 3D flow fields is finding the correct information density. Too much information per volume occludes features further away and too little information may hide important details. When using streamlines for 3D-flow visualization, the amount of information in a given volume is directly related to the number of streamlines through it. To a lesser degree it also depends on the number of sample points along the streamline. As described in

section 12.6 we place our streamlines approximately equidistant to each other. Therefore density of the streamlines in the resulting images depends mainly on this user selected distance. Other factors contributing to the general appearance are width of the streamline and - in case of dashtubes - the length ratio of opaque to transparent sections.

When investigating a 3D flow we first try to get an overview of the flow field. This includes investigation of global features, the identification of areas of special interest, like vortices, separatrices, and cycles. Then, when an interesting feature has been identified, we want to single out this feature and investigate it. We want to view it in great detail, without distractions or occlusions from other features.

In many practical cases, these two different goals are difficult to achieve simultaneously. Therefore we tested techniques where we first use the context of a coarse representation to identify interesting regions. Then we use one of the mechanisms described in sections 12.7.1 and 12.7.2 to focus our attention on these regions and investigate them with finer detail.

Magic lenses, as presented in [Bier1997] are transparent user interface elements for conventional 2D windowing desktop environments. They are represented by special windows, which do not display their own independent content but rather change the representation of the underlying information. They can be used for filtering or otherwise modifying underlying image data but also for more abstract operations like showing additional information like comments. In [Viega1996] magic lenses are used in virtual environments. This work also deals with volumetric lenses, an extension of magic lenses in three dimensions.

We use these interface elements to view a higher resolution representation of the flow field. This representation contains more streamlines per volume and the streamlines are thinner and generated with closer spaced vertices than the representation used for coarse navigation. We found that both focussing techniques - lenses and boxes - have specific advantages.

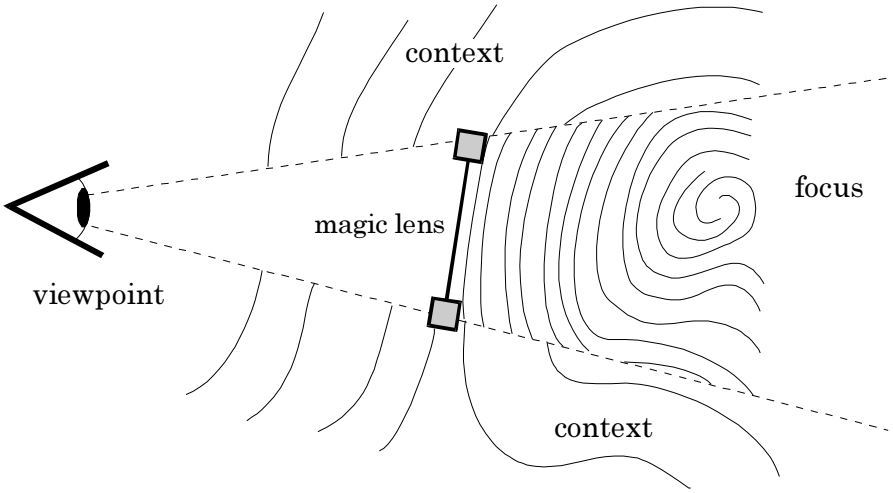


Figure 50: Volume defined by magic lens

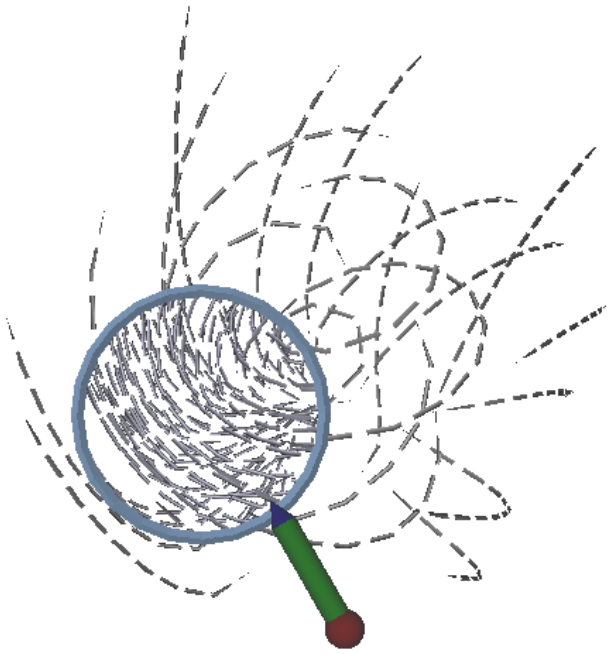


Figure 51: Focussing with a magic lens

12.7.1 Magic Lenses

A magic lens is a planar polygon with arbitrary boundaries (e.g., circle, square) which can be positioned with a 6DOF input device, normally a 3D mouse or tracked pen (Figure 51). When looking through the lens, the user focuses on the high-resolution representation.

The main difference to 2D magic lenses is that our lens additionally acts as a clipping plane, allowing only the parts of the high resolution scene behind the lens to be seen. Without this clipping plane, the lens would also display features of the detailed representation between lens and viewpoint, resulting in the same occlusion problems as if the entire detailed flow visualization were to be investigated. Together with the current viewpoint a magic lens effectively defines a viewing frustum with its near clipping plane lying in the plane of the lens and its cross-section defined by the shape of the lens (Figure 50).

Working with the magic lens is easy and intuitive. The user positions it in front of interesting features and views them through it like through a magnifying glass (Figure 51). The frame of the lens masks the border between focus and context. While presenting an effective and visually appealing investigation mechanism, magic lenses have one distinctive disadvantage compared to magic boxes (section 12.7.2): the focussed volume depends strongly not only on the position of the lens but also on the viewpoint. This does not matter when a single user is looking for a local feature. The user typically sweeps the lens through space, positioning it and himself until the area of interest is located. When this has been accomplished, the investigation technique normally changes: Now, that the detail has been located, it has to be examined from different angles, a procedure for which magic lenses are not well suited. The lens has to be dragged around the feature together with the changing viewpoint. This is a cumbersome process, which may lead to accidental loss of the focus area and the contained feature.

12.7.2 Magic Boxes

Magic Boxes - volumetric magic lenses - overcome the above mentioned disadvantages of viewpoint dependency. Instead of only im-

plicitly defining the focussed volume depending on the current viewpoint (Figure 50), they explicitly define a volume of interest (Figure 52). In the interior of a magic box the detailed representation of the flow is displayed.

The user positions the box with a 6DOF input device until it contains the local feature (Figure 53). Then this feature may be viewed from all directions. This is especially important when there are several users viewing the same focus like in Studierstube. When using magic lenses every user has to position his own lens according to his position, or different users have to trade places when looking through a single lens.

The border between focus and context is more noticeable when using boxes instead of lenses, since it consists of the whole surface of the box and cannot be masked by a frame like the image-aligned border of the lens. On the other hand the confined volume of the box allows focussing in all three dimension, whereas magic lenses do not inherently define a far plane of the focus. Additionally the box occludes features of the context behind it, thereby reducing distraction.

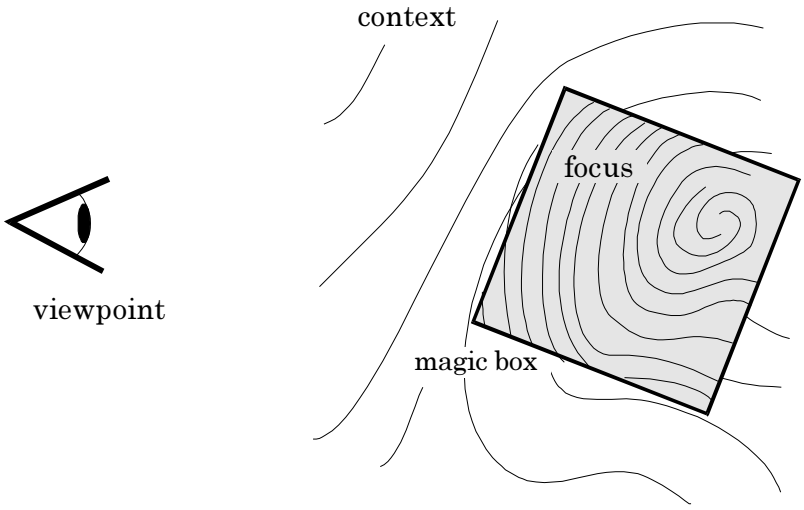


Figure 52: Volume defined by magic box

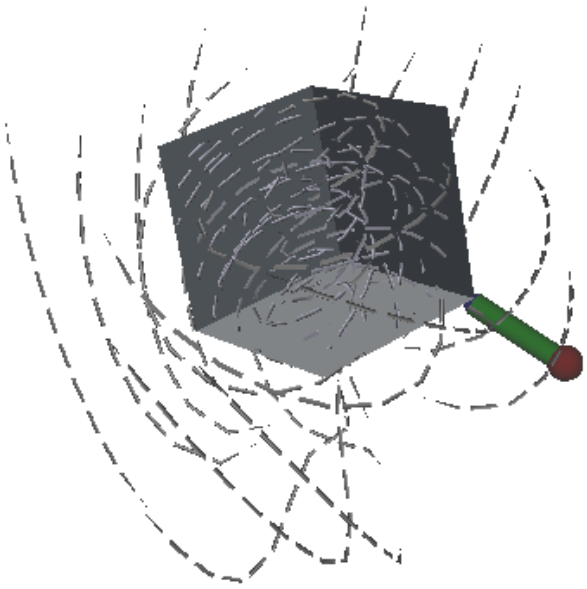


Figure 53: Focussing with a magic box

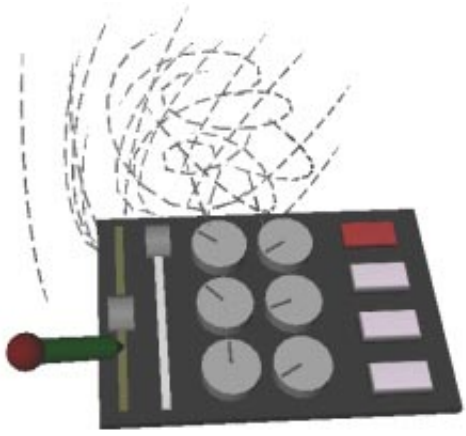


Figure 54: Interaction using the PIP

12.8 Implementation

The visualization and investigation methods described above were implemented in C++ using Open Inventor [Strauss1992]. This OpenGL based graphics toolkit enabled us to efficiently realize our

methods providing high-level graphics concepts like a scene graph and sophisticated desktop interaction elements, which we used in the early phases of our tests. The main advantage when implementing our methods was Open Inventors ability to supply these high-level concepts while simultaneously enabling direct access to all OpenGL functions. This was essential when manipulating rendering sequences for magic lenses and magic boxes. Since our virtual environment STUDIERSTUBE is also based on Open Inventor, the transfer from a desktop evaluation-implementation to the application in our virtual environment was straightforward. The following sections describe implementation details of the techniques and their integration into our virtual environment.

12.8.1 Interaction in Studierstube

In our application we used the PIP to adjust parameters of the dynamical system which provided the flow field as well as properties of the dashtubes and the magic box. The speed, length of dashes and distance between dashes was adjustable with dials. Sliders on the PIP adjust the overall size of the magic box and allow independent scaling of one dimension of the box. This transforms the box to a ”slab”, allowing the user to use it to cut slices of arbitrary width out of the flow field. Buttons on the PIP were used to switch between magic lens and magic box and to disable the coarse representation on demand.

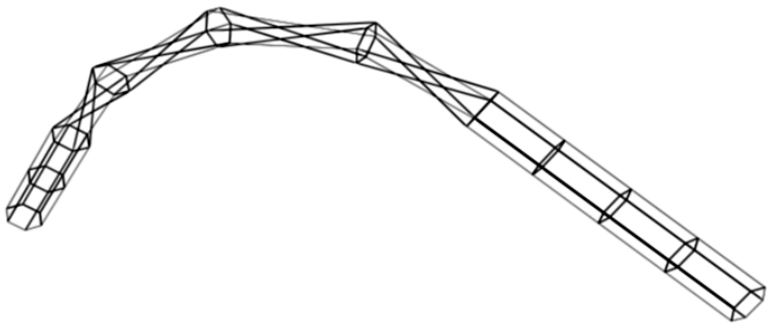


Figure 55: Contraction artifacts due to torsion

12.8.2 Dashtubes

Dashtubes are realized as textured polygonal extrusions along the direction of the flow. Ideally the cross-section of the extrusion should be a circle to provide a symmetric appearance from all directions, but we found that the polygonal approximation can be reduced down to 3 to 6 edges depending on the resolution of the display and the required quality of the image. By using Gouraud shading the resulting discontinuities of the approximation are only visible along the silhouette edges. Coarse tesselations like these are prone to generating artifacts when the geometry is twisted along the extrusion axis. The resulting radial contractions lead to irritating variations in the width of the dashtube (Figure 55). To avoid this, we generate the segments of the extrusion not by following the Frenet-frame along the streamline. We use a algorithm similar to [Bloomenthal1990], which reduces the torsion by aligning the orientation of the polygonal cross-sections along the segments.

The geometry of the dashtubes was implemented as Open Inventor Shapekit, containing fields for the vertices of the extrusion axis, the texture parameters and the geometric parameters of the cross-section. The Shapekit produces OpenGL trianglestrips, which give a better rendering performance than other OpenGL primitives. Rendering the dashtubes with culled backfaces produces a ”halfpipe” appearance at both ends of the opaque segments as visible in Figure 49. Since this is only evident in extreme close-up, we decided that the rendering speedup justifies this artifact.

12.8.3 Magic Lenses

Magic lenses act as window from context to focus. Our implementation uses SEAMs [Schaufler1998], a mechanism to connect two virtual worlds by "windows" of arbitrary geometry. Our magic lens has the appearance of a magnifying glass, using a circular SEAM inside a ring geometry providing the frame (Figure 51). According to the nomenclature of [Schaufler1998], the context outside the lens would be

the ”primary world” and the focus seen through the lens the ”secondary world”. The geometry of both worlds is given as a directed acyclic graph (scene graph). The scene graph of the context is traversed and rendered. When a SEAM is encountered, the associated polygon - in our case the ”lens” - is passed to the rendering hardware for scan conversion.

To restrict the rendering of the focus to the area covered by the SEAM we use the OpenGL stencil buffer, an additional layer for masking areas of the screen during rendering.

For all pixels that the z-test for the SEAM polygon finds to be visible:

- The frame buffer is set to the background color of the secondary world (clear screen),
- the Z-buffer is set to infinity (clear Z-buffer),
- the mask (stencil buffer) is set to 1.

Note that these image modifications are only carried out for the visible portion of the SEAM surface. After this preparation step, rendering the focus is performed inside the stencil mask created in the previous step. This prevents that the focus is drawn outside the SEAM area. A clipping plane coincident with the SEAM polygon prevents the focus from protruding from the SEAM. Finally - before rendering of the context proceeds - the SEAM polygon is rendered again, but only the computed depth values are written into the z-buffer. Thereby the SEAM is ”sealed”. The resulting z-values are all smaller than any z-value of the focus. This asserts that no geometric primitive of the context located behind the SEAM will overwrite a pixel generated by rendering the focus.

This gives the desired impression of a "window" behind which only the focus is displayed (Figure 51).

12.8.4 Magic Boxes

We found that displaying only the contents of the magic box without visual representation of its boundaries makes it difficult to locate and position the box and tends to confuse the user. Therefore we added a cube as geometric representation of the focussing volume. The front

faces of the cube are culled, leading to an ”open front” appearance regardless of the viewpoint.

Magic boxes are rendered using the same SEAM algorithm as described above, but use a cube instead of a plane to define the ”windows” between focus and context. Six clipping planes coincident with the faces of the cube clip the secondary world (the detailed representation). The main difference between magic boxes and the 3D windows described in section 8.3.2 lies in their application, not their structure: while the 3D windows contain a scene anchored to the window and independent from its position, the magic box represents a moveable volume inside a scene anchored outside the box. Thereby moving a window moves ist contents, while moving a box shows a new section of space inside the scene. Since the scene inside the box normally corresponds in some way to the scene outside – in our application it is simply a high-detail version of the same flow-field – the user can navigate through the inner scene by using hints taken from the outer scene. For example a vortex barely visible in the outer scene – the coarser version of the flow-field – can be investigated in detail when the magic box is moved over it.

Our implementation renders the complete scene (focus and context) only once, while [Viega1996] needs six rendering passes, one for each halfspace derived from a cube face. This leads to a significant improvement in the frame rate. Our method does not display any parts of the context that lie behind the box, which for our application would anyway only be distractive.

12.8.5 Results

Animated dashtubes produce an intuitive visualization of a 3D flow-field. The main problem when applying our focussing techniques lies in finding the correct density for focus and context. When testing lenses and boxes with different densities of dashtubes in the focus we found that magic boxes work better than lenses with densely placed dashtubes. Since lenses do not clip distant parts of the detailed scene they are only applicable to scenes of higher density when they are rendered with strong depth cues (haze, fog).

Most users applied magic lenses without any problems, but needed some experimentation to grasp the concept of volumetric magic boxes.

The method of slicing the flow field with ”slabs” - magic boxes of small height - as mentioned in section 12.8.1 was implemented after users started to experiment with the distances of near and far clipping plane of the view volume to achieve this "slicing" effect in a view dependent manner.

When using magic boxes, most users applied the following technique: position coarse representation conveniently; position box until interesting features visible; switch off coarse representation; magnify box with included details for investigation

Since our application allowed independent positioning of focussing element and flow field, some users preferred positioning the flow field and keeping the box or lens stationary. During the design phase of the dashtubes we used shutter glasses to produce stereoscopy. While this works very well for the examination of the flow field, interaction with magic lenses and magic boxes using the 2D desktop mouse is cumbersome and non-intuitive compared to the interaction in the virtual environment.

The newly introduced adaptive texture mapping method shows that texture hardware can be efficiently used to produce dashtubes with uniform spatial resolution. The approach ensures that velocity variations are still encoded in the animation. Dashtubes are automatically positioned in phase space to produce an even representation of the underlying 3D flow.

Interactive tools like magic lenses and magic boxes proved to be valuable in the investigation of local features. They enable the user to interactively select finely detailed features and reduce distraction by context. In our investigations 3D phase space contains spatially complex structures, which are difficult to interpret. The added cues of a virtual environment (e.g., stereoscopic viewing, interactive and intuitive viewpoint change) are quite helpful when inspecting these structures.

13 Results

This chapter consists of applications and results of the work described in the preceding chapters.

The first section of this chapter describes the application of our basic concept of performing scientific visualization in augmented reality. It focuses on the visualization of dynamical systems. An evaluation of our new interaction methods and visualization icons concludes this section.

In the course of the development of our collaborative virtual environment we soon realized that *Studierstube* presented more than a tool for scientific visualization. Applications ranging from simple demos to Computer Aided Design and cloth simulation have already been implemented in Studierstube. These applications as well as a general evaluation of the *Workspace* concept and collaborations with other research facilities are presented in the second section of this chapter.

13.1 Scientific Visualization in Studierstube

13.1.1 Visualization Applications

To study the performance of Studierstube as a collaborative visualization tool, we chose dynamical systems as an application field.

AVS networks composed from DynSys3D modules were customized to exploit AR capabilities, and usability was informally evaluated in multiple sessions with a varying set of participants (researchers and students in mathematics, visualization, and augmented reality).

In the following, we give some examples for the applications we have investigated. All these examples immersed the user in the phase-space of the dynamical system and allowed direct manipulation of position, orientation and scaling of the coordinate system.

RTorus

RTorus is a 'synthetic' system that demonstrates important common properties of dynamical systems (Figure 56). Abraham and Shaw use this model as an example to explain several fundamental flow properties [Abra92]. This dynamical system models a coupled oscillation within three-space. Depending on the parameters of the model either an attracting cycle within the x-y-plane or an attracting torus around the z-axis appears.

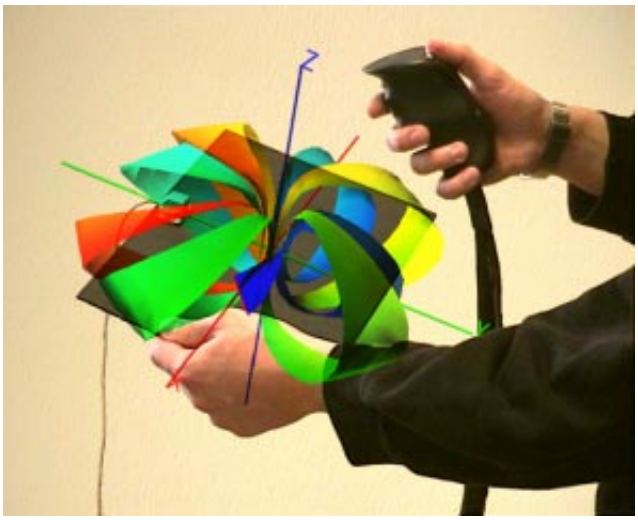


Figure 56: RTorus on the PIP (two-handed interaction)

Rössler

As a rather well-known dynamical system we also investigated the Rössler attractor in Studierstube (Figure 57). Rössler is also a three-dimensional dynamical system that exhibits a chaotic attractor if parameters are set properly. Taking this familiar dynamical system for analysis in Studierstube allowed us to easily compare visualization in AR to established techniques.

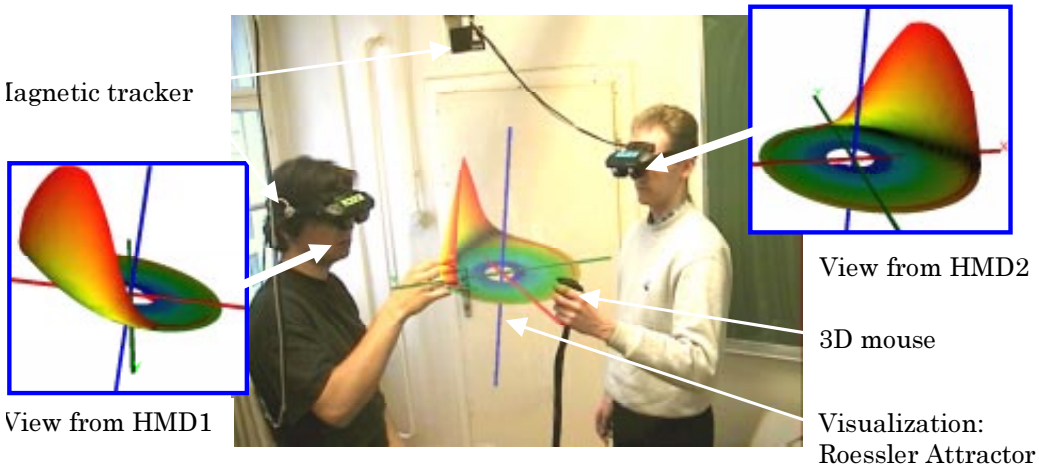
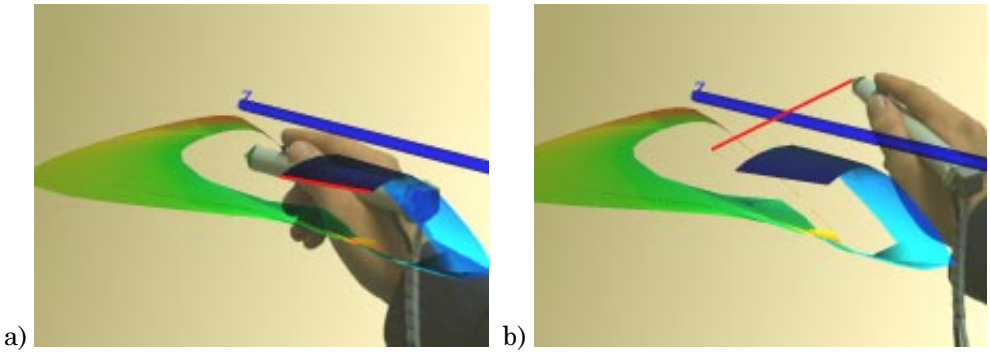


Figure 57: Roessler Attractor in Studierstube
Illustration of different viewpoints

Mixed-mode Oscillations

A model we investigated together with colleagues from our econometrics department is the 3D autocatalator [Milik1996, Petrov1992]. It is a simple 3D dynamical system which exhibits mixed-mode oscillations. These are phenomena often encountered in real world systems, for example in chemical systems. Depending on the parameters of this system either periodic or quasiperiodic (chaotic) solutions can be found. Direct immersion in the 3D phase space provided a useful tool for the investigation of its behavior. Structure and relations of visualization icons that are produced by the visualization system can be better investigated through 3D interaction in augmented reality.



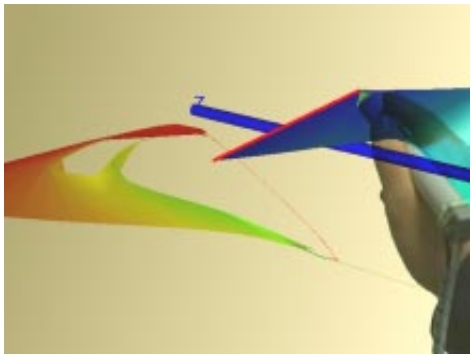
c) 

Figure 58: injecting a new streamsurface into MMO:
a) button down, old streamsurface still visible
b) drag inserts new seedline (red “rubberband”)
c) button up: new geometry (streamsurface) appears

We used streamsurfaces started by direct manipulation (Figure 58) as visualization icon and supplied the mathematician with direct access to the system parameters via dials on the PIP. The visualization expert used his PIP to adjust the integrator stepsize, accuracy and resolution of the streamsurface to optimize the resulting geometry.

Dynastic Cycle

This model deals with the visualization of the "Dynastic Cycle" , a three-dimensional dynamical system, that was modeled as an explanation for the rise and fall of dynasties in ancient China, given as alternating periods of anarchy and despotism [Feichtinger1996]. The three system variables X, Y , and Z express the amount of farmers, bandits, and soldiers, respectively. The model defines their interactions similarly to well-known food-chains (prey, predator, and super-predator). The evolution induced by the Dynastic Cycle is governed by slow-fast dynamics. Two of the system variables (X, Y) are fast variables that change rapidly in comparison to the last one (Z). The knowledge about this slow-fast characteristics simplifies the analysis and must be considered during visualization. Animating the length of the streamline in the virtual environment proved to be an efficient way of visualizing this behavior and was accomplished by simply inserting an animated float parameter - a standard feature of AVS - to the dataflow network. Since this system exhibits vastly different behaviors, we used sliders on the PIP to adjust the most important pa-

rameters. By using these sliders and the insertion of animated streamlines we were able to obtain the visualizations in figure 6.

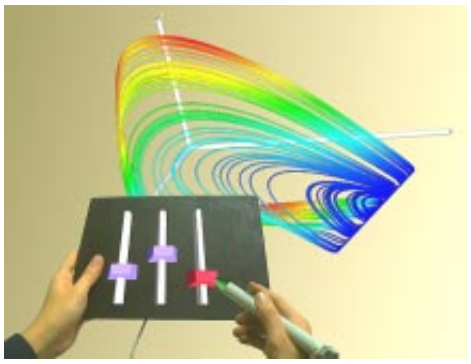
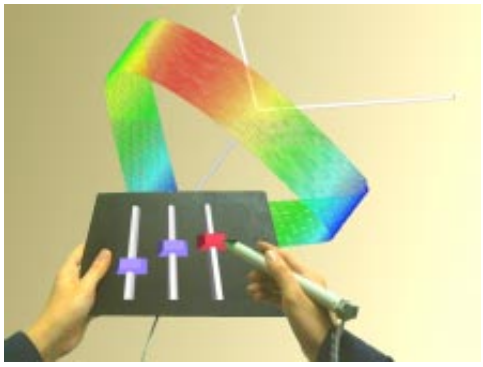
a)  b) 

Figure 59: Changing system parameters using the PIP (dynastic cycle: a) chaotic, b) periodic)

In addition to all interaction methods used in the mixed-mode oscillations example we allowed introduction of streamlines by all users. Streamline parameters were again accessible via the PIP of the visualization expert.

13.2 Studierstube as a generic user interface

During our work *Studierstube* quickly grew from an environment primarily planned for scientific visualization into something of an all-purpose environment. In the following section we present applications besides scientific visualizations which have been developed using the Studierstube API (Application Programmer Interface).

13.2.1 A sample Workspace session

This section gives a short overview of essential features of our Studierstube Workspace implementation. We want to show how the Workspace concept has been applied in our VE and how standard tasks can be performed. Note that these examples only represent a small aspect of possible situations and policies, and are presented here as demonstration of features discussed before and as proof of concept.

The user enters the Workspace by putting on the head- mounted display (HMD) and picking up pen and PIP. Normally the PIP is held along an edge using the non-dominant hand, similar to a painters palette. The pen is held like a real pen and allows interaction on the PIP in a familiar pen-and-clipboard manner as well as 3D interaction with 6DOF. For some tasks users prefer to hold the pen like a wand or an umbrella, which is made possible by the symetric arrangement of buttons along the pens axis.

The PIP is the main mechanism for abstract interaction in the Workspace, e.g. input of values and selection of abstract properties as opposed to direct interaction like 3D dragging and pointing. It is not only used as application input device displaying the input elements of running applications but also as primary control device for the Workspace. Directly accessible via short-cut corners (red and green triangles in figures 5 & 6) are two special sheets which always available, the system sheet and the application sheet.

The application sheet - selected via the green corner - functions as a simple shell and enables the user to browse the filesystem and start Studierstube applications or switch focus between running applications. This module has been implemented along the lines of the familiar “file open” dialog from desktop applications. Running applications are shown as icons and allow explicit focus changes (Figure 5). Implicit focus change occurs when clicking into a non-focused window.

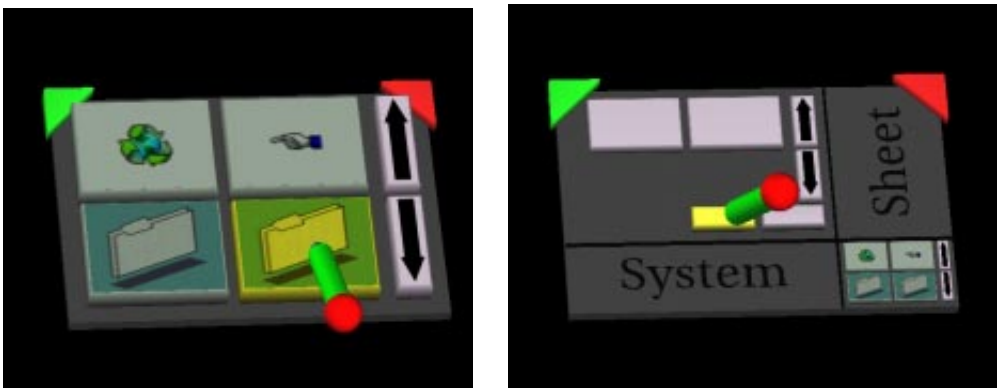


Figure 60: The application sheet (left) starts new applications and switches between running applications. The system sheet (right) can be used to control system and window state.

The system sheet (Figure 7, right) gives access to 3D window configuration methods. A 3D window can be switched to a maximized, 3D, 2D or minimized representation (described in section 8.6.2). Further options like display of title bar or opaque background are accessible too. A window can be resized and positioned by dragging its frame or its corners equivalent to 2D windows.

Focus changes are realized by a pen-click into the desired window, which automatically switches to the associated input context (sheet) on the users PIP.

13.2.2 Workspace Applications

The following applications demonstrate the variety of tasks that can be performed within the framework of our Workspace implementation. Large parts of the relevant interface appearance and behaviour have been implemented using the predefined interaction classes, in some cases even only by scripting inside the scene description files.

Calculator

The simplest application is this simulation of a desktop calculator (figure). It has been written as a test program for 2D interaction elements and demonstrates nicely how the PIP is used for 2D interaction with the “button” interaction elements. This application is frequently used to instruct new Studierstube users on the usage of the PIP. All visible geometry and most of the interaction behaviour has been defined in an Open Inventor scene description file using simple scripting in a text editor.

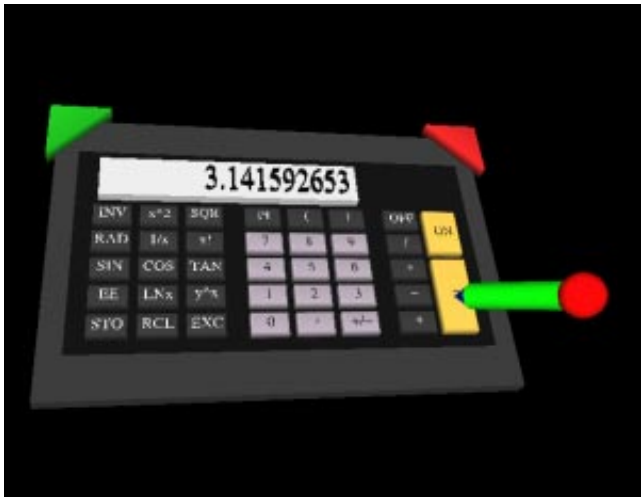


Figure 61:The calculator utility on the Personal Interaction Panel makes intensive use of 3D button widgets. The flat surface of the wooden panel gives good tactile feedback and provides natural support for 2D interaction in 3D

X-ray viewer

Almost as simple is this integration of the SEAM [Schaufler1998] interaction element in a medical simulation. The SEAM acts as a magic lens (see section 12.7.1), giving the user the ability to look under the skin of a patient. Positioning of the lens is done by dragging its frame over the patient, allowing a real-time cutaway view of skin and skeleton at the same time.

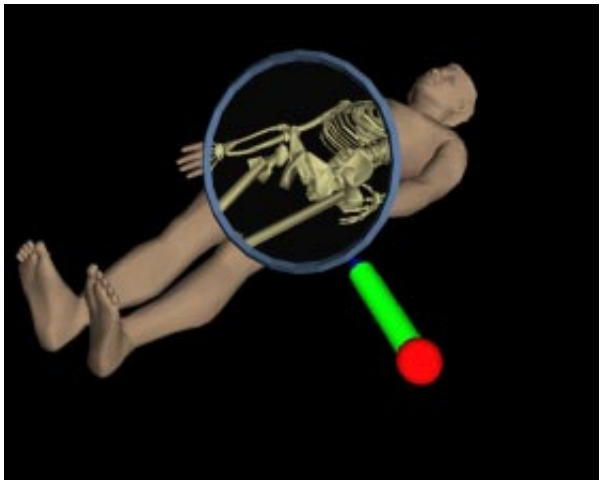


Figure 62: X-ray viewer - the pen becomes a lens-shaped tool to examine medical data

Since all the positioning is done by a “dragger” interaction element, the application consists only of an initialization of body and skeleton representation and the SEAM with its frame coupled to the dragger.

3D flow visualization

Here we implemented our geometry and texture based flow visualization technique [Fuhrmann1998b] (see also chapter 12) in Studierstube. Parametrization of the simulation and the visualization was done via the PIP. Additionally SEAMs were used as “magic lenses” and “magic boxes” to navigate inside a sparse representation of the 3D flow and select areas where the flow should be depicted with high detail.

Again all real-time interaction have been defined using the standard interaction elements and only the application specific parts - simulation, visualization and animation of the results - had to be implemented.

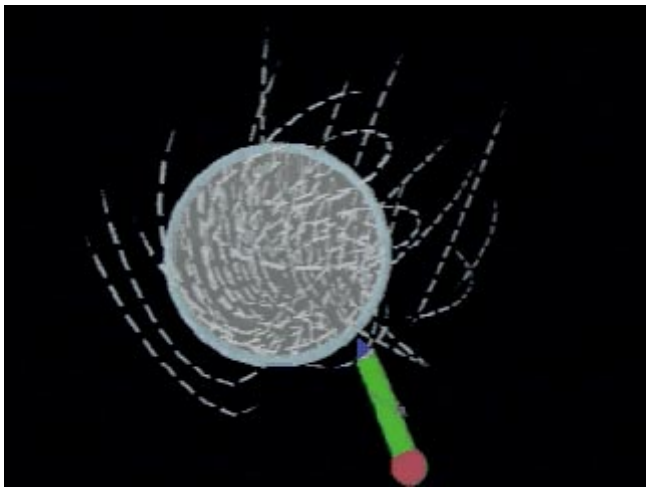


Figure 63: Flow visualization using magic lens for focussing

Landscaping

In our landscaping application [Schmalstieg1999] we have integrated a variety of interface elements and methods. It is a CAD-like application specifically designed for the development of landscaping solutions in suburban environments. Simple interactions like object placement (houses, trees) are integrated as well as specialized metaphors like a cable TV tool that provides the user with X-ray vision (Figure 9). The user can look under the surface of the landscape representing an island (using wireframe rendering) and uses the pen to lay wire and connect houses to a cable TV network.

Interesting here is the direct application of the PIP as 3D interaction device: Sweeping the PIP like a fishnet through the scene (Figure 10) selects all objects which were "caught" in the motion. Furthermore the PIP can be used as a camera, taking virtual snapshots of different states of the landscape and positioning them like notes somewhere in the Workspace. This snapshot tool allows the user to manage a collection of scenes that are viewed from different perspectives and in different stages of development.

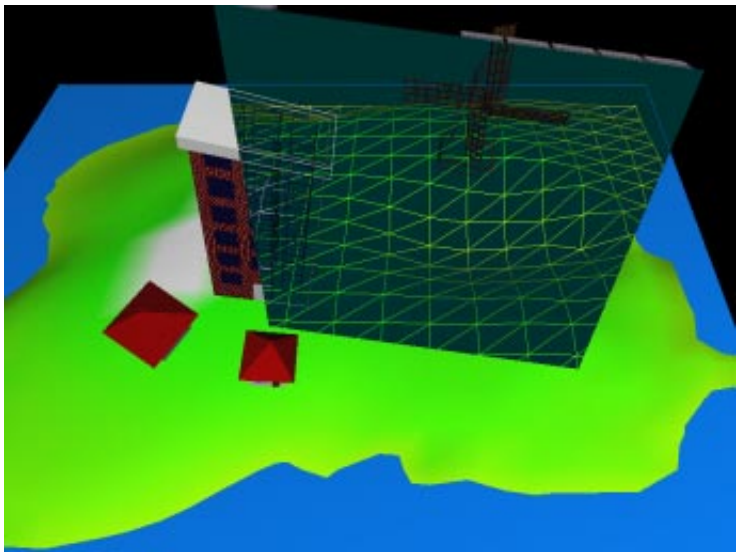


Figure 64: Using the PIP as "magic lens" to show hidden properties of the landscape

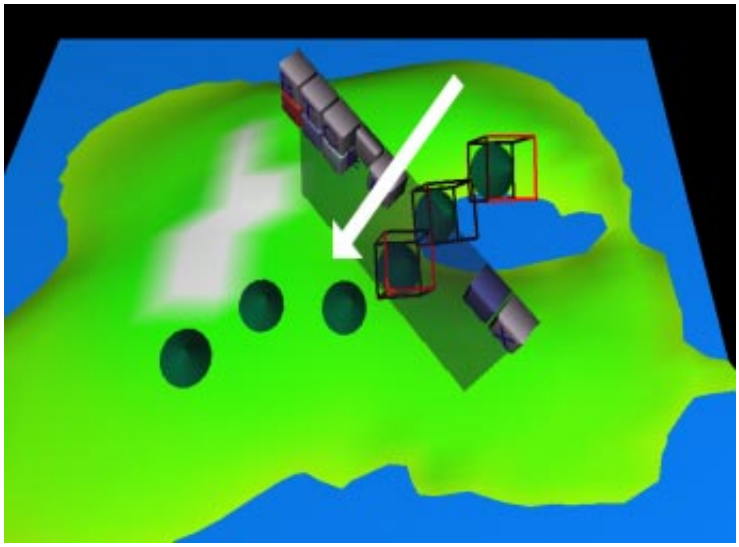


Figure 65: Using the PIP as "fishnet" to select objects (selected spheres with red frame)

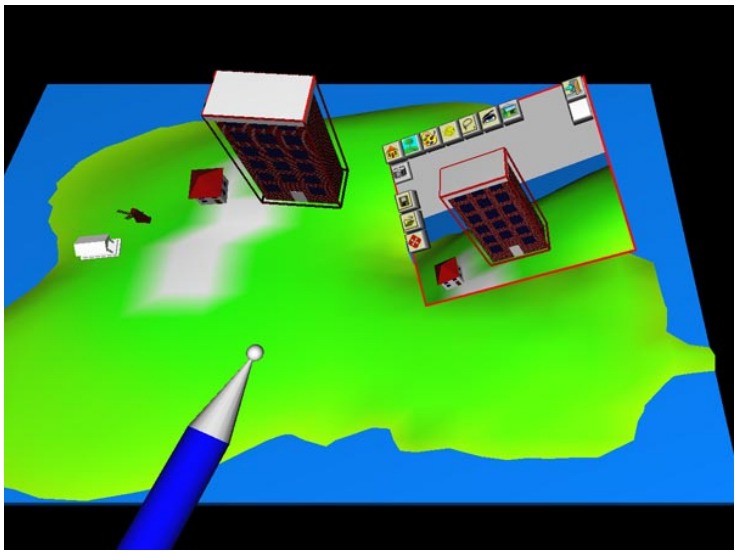


Figure 66: Taking and displaying 3D "snapshots" using the PIP

3D painting application

The painting application (Figure 67) demonstrates how multi-user application function inside Studierstube Workspace. Upon start-up, it generates a single 3D window and PIP sheets for every user inside

the Workspace. The sheets contain sliders for colour selection and brush size as well as buttons to select the brush type - spray or pen - and to clean the blackboard.

Every user may select a colour and brush according to his needs, which is displayed as “life size” icon on his PIP. Painting is done by dragging a pen through the window. Depending on the selected brush type, a stream of 3D “dots” - emulating a spray can - is generated. Equipped with this simple interface, users are free to generate whatever three- dimensional pointilistic art they may have in mind.

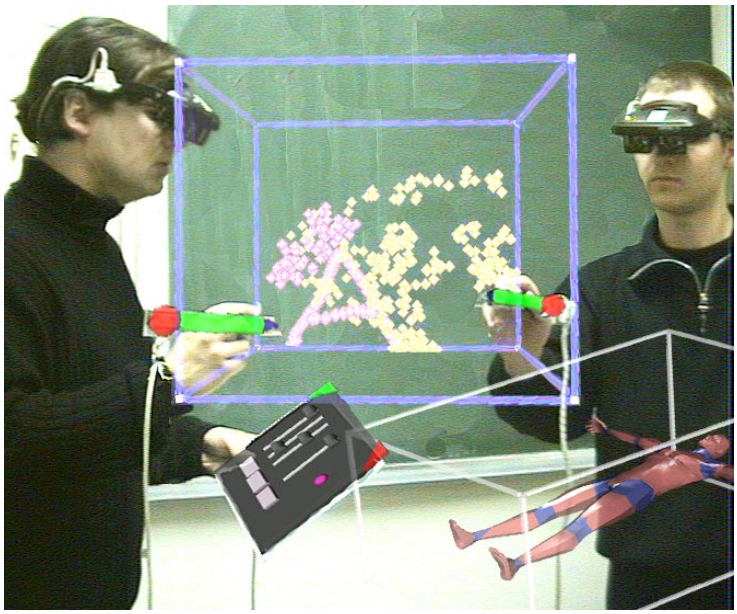


Figure 67: 3D collaborative "painting" (Note second application window
with medical visualization in front)

The possibilities of multi-user collaborations appeared to most users when they inadvertently entered each others paint volume and resulted in impressive, if abstract works as well as in “paint fights”. In this application the advantages of AR could clearly be observed: since users could see each other as well as their creations, collisions of painters or paintings could easily be avoided (or deliberately provoked!) and collaboration on single paintings was enhanced by gestures and discussions.

13.3 Scientific Collaborations

Studierstube has been part of the following collaborations with other research institutions:

13.3.1 University of Graz: Hybrid Magnetic-Optical Tracking

University of Graz
Axel Pinz, Thomas Auer
http://www.crcg.edu/projects/virtual_table.html

The commercial magnetic trackers we use in *Studierstube* have only limited range and exhibit significant distortions and noise at distances greater 1.5 meters. As an remedy a second workgroup at the university of Graz has constructed an augmented reality helmet that integrates the magnetic sensor, a stereo camera pair and a see-through display used for augmenting the room, similar to [Hoff1996]

The helmet in action is depicted in Figure 79. Using the helmet, a hybrid system combining both magnetic and optical tracking has been developed: The output from the magnetic tracker is used as an initial estimate for the optical system. After the result from the optical system has been obtained it is verified with the magnetic system by computing the offset between the magnetic and the optical system. If the offset is too large, the result from the optical system is not accepted and we have to rely on the magnetic system alone, otherwise, the data generated by the optical system are used. Further, if the optical result is accepted, we update the offset between the two systems, because the offset depends on the current location of the helmet, but has only small changes for small movements of the user. [State1996a].

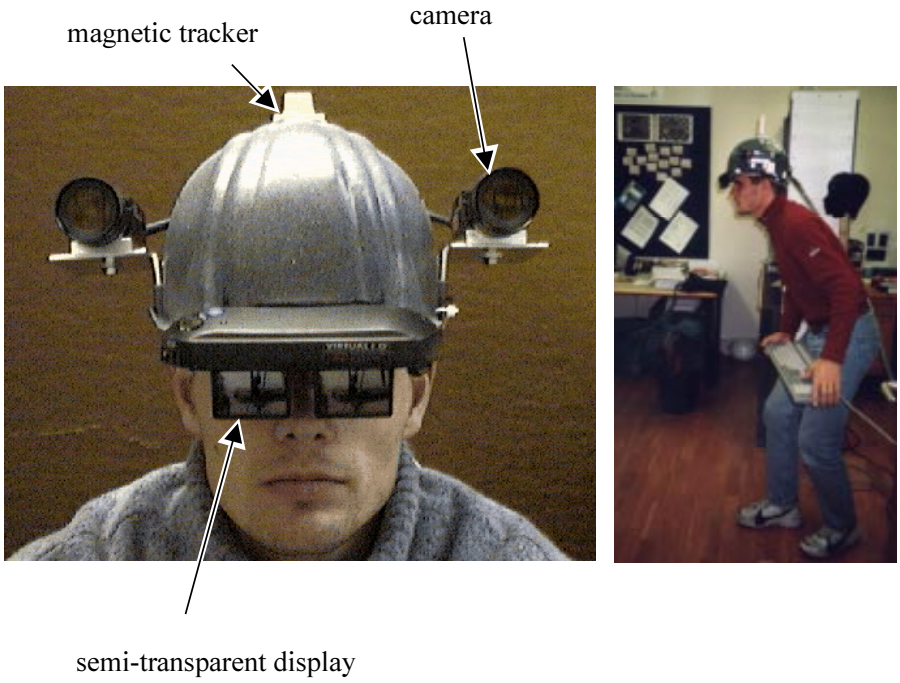


Figure 79: Helmet-Mounted display with attached cameras

13.3.2 CRCG Providence: Studierstube VT and Sketch interface

Fraunhofer CRCG Providence, RI,
L.Miguel Encarnação, Sean Chandler
http://www.crcg.edu/projects/virtual_table.html

The adaptation of the *Studierstube* concept to a completely different virtual environment has been performed by Dieter Schmalstieg in the course of his work at the Fraunhofer CRCG [Schmalstieg1999]. Here the display device is not a semi-transparent HMD but a virtual work-bench [Krüger1994] (Figure 80). The implementation of the PIP has been adapted for back-projection by using a transparent pad which allows augmentation by looking through it onto the display surface (

Figure 81).

Figure 80: Setup for *Studierstube VT* (Illustration)

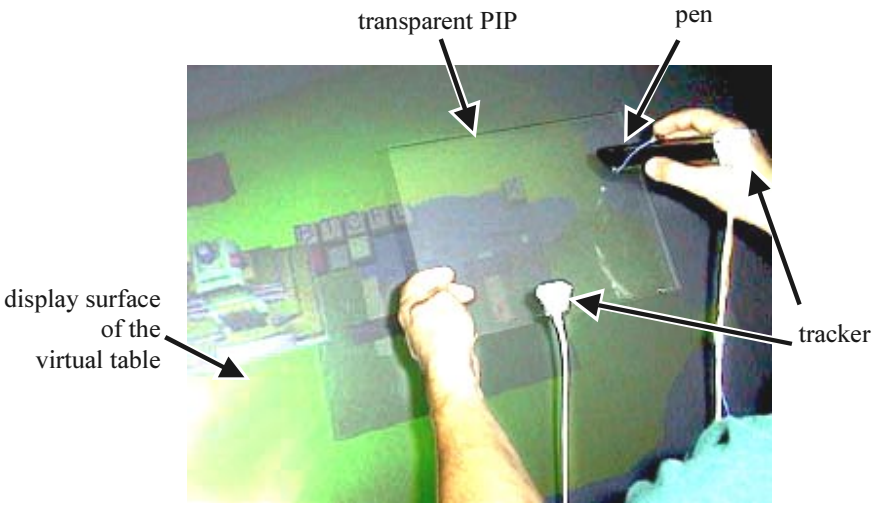


Figure 81: The transparent PIP in *Studierstube/VT*.

The SKETCH gesture interface has been adapted by the Fraunhofer CRCG to recognize gestures performed on the PIP (Figure 82).

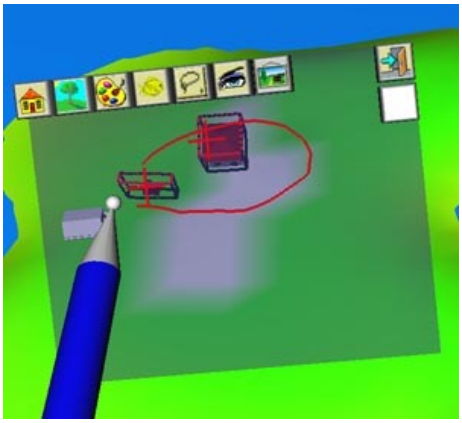


Figure 82: using gestures on the transparent PIP

Gesture-based interfaces offer an alternative to traditional keyboard, menu, and direct manipulation interfaces. The ability to specify objects, an operation, and additional parameters with a single intuitive gesture appeals to both novice and experienced users. Gesture-based interfaces for platforms such as the BARCO table are desirable, as they permit far more natural interaction than traditional mouse and keyboard interfaces. Our system includes gesture recognition techniques developed for SKETCH, a rapid virtual prototyping environment, developed at Brown University [Zeleznik1996]. In the early stages of a design, often what is required is a rapid conceptual model of the desired product.

SKETCH is designed to bridge the gap between hand sketches and computer-based modeling programs, combining some of the features of pencil-and-paper sketching and some of the features of CAD systems to provide a lightweight, gesture-based interface to ``approximate" 3D polyhedral modeling. SKETCH uses a gestural mode of input in which all operations are available directly in the 3D scene through a three-button mouse or three-button stylus.

13.3.3 Universität Tübingen: Nähstube

Universität Tübingen,
Department Graphisch-Interaktive Systeme
Wilhelm Schickard Institut für Informatik
Bernd Eberhardt

The *Nähstube* ("sewing room") is an ongoing project using *Studierstube* to interactively place and connect pieces of cloth over a tailor´s dummy. Complex 3D-placement operations like this are only difficult to achieve in desktop environments but are easily performed in an environment where 6 degrees-of-freedom input devices and 3D head-tracked viewpoints are implemented (Figure 83).

Figure 83: Placing simulated cloth over a virtual dressmakers dummy

Generating realistic images of cloth with different fabrics such as wool, cotton or silk is still a problem in garment design. The correct fall of a towel or a skirt can only be observed when the material is produced, the garment sewn and the result worn by a person. To reduce the design cycle we want to simulate this whole process with computer graphics.

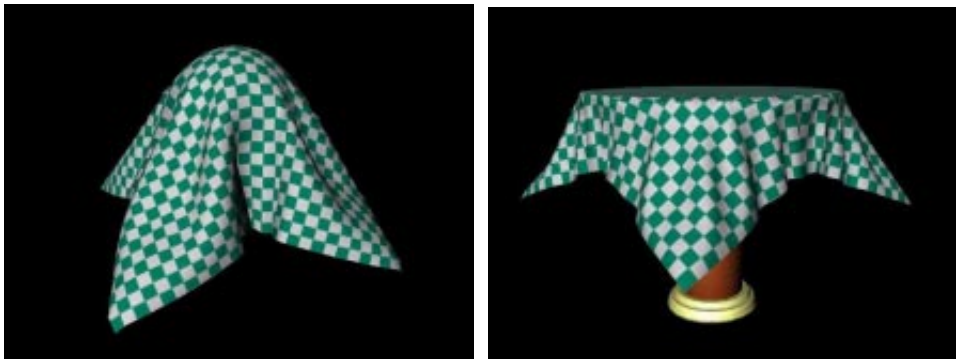


Figure 84: Draping simulated cloth over a sphere and a table

The simulation developed by Bernd Eberhardt uses particle systems to simulate the fall of the cloth and obtain realistic draping behaviors. This approach is more general than others and produces images much faster than other implementations (Figure 84).

13.3.4 Fraunhofer IGD Darmstadt: ARCADE

Fraunhofer IGD A2 Darmstadt,
Department for Industrial Applications
Wilhelm Schickard Institut für Informatik
André Stork, stork@igd.fhg.de
http://www.igd.fhg.de/www/igd-a2/Projekte/Arcade/arcade_e.html

ARCADE (Advanced Realism CAD Environment) - developed at Fraunhofer IGD - is a 3D modeling system, that places natural interaction techniques (for instance 3D input for object creation) at the user's disposal. During the whole construction process the objects are displayed realistically using insights from the science of perception. Moreover ARCADE provides visualization techniques which enable the user to recognize the relative position of the objects in the design space.

Traditional 3D-CAD-Systems are burdened by a discrepancy between the virtual three-dimensional design space and the two-dimensional input and output devices. Often 3D interactions have to be built from a sequence of two-dimensional actions. The presentation of the 3D objects on the 2D display and the applied visualization

techniques render only a few depth and shape hints. These discrepancies hamper the user's orientation, placement and navi-gation, so that the advantages of 3D cannot be used entirely.

ARCADE offers the following new approaches, features and capabilities:

- Integration of 3D input and output devices
- intuitive, direct manipulative 3D interaction techniques
- high-quality presentation of design objects during construction (Material, Texture, Lighting)
- support of a distributed cooperative design process
- interactively configurable graphical user interface
- History Graph with pattern creation and powerful undo, redo and copy features
- intelligent walls as visualization of the dynamic, virtual design space
- transparent shadows
- 3D grid with snapping und alignment
- interactive clipping-plane
- interactive walk-through

Since the graphical input and output of ARCADE is already based on Open Inventor, the integration into *Studierstube* was quite straightforward. Special interface techniques are still managed by ARCADE directly, whereas the standard *Studierstube* input elements enrich the user interface of ARCADE and transfer most of the 2D desktop interface into the virtual environment.

14 Evaluation

This thesis has presented design guidelines and implementation strategies for a collaborative user interface for scientific visualization in augmented reality - the Studierstube Workspace. Below we evaluate the results presented in chapter 13 in an informal user study conducted during our work on the visualization of dynamical systems.

We show how our shared virtual environment solves the problems stated in chapter 3 and which of the realized concepts we implemented proved to be most valuable.

14.1 Studierstube Workspace

Concept and implementation of our design proved to be viable under the requirements which we formulated in our problem statement in chapter 3:

- *Augmentation*: The integration of the *Personal Interaction Panel* and the tracked props and avatars we used for occlusion in chapter 10 were easily integrated into the environment. Precise calibration as described in chapter 9 enabled the registration of virtual and real geometry necessary for unhindered interaction.
  The special interaction techniques as described in section 0 worked as well on *PIP* as the 2D interaction elements (section 8.3.1).
- *Collaboration*: The direct visual contact between users and the concept of a shared augmented environment resolved most of the physical and logical collissions common to immersive or distributed virtual environments.
  The concept of *Multi User Aware Applications* (section 8.5) allowed a sopphisticated treatment of general and application specific collaboration techniques. The 3D Event System (section 8.6.1) actively supports all mechanisms necessary to implement collaboration on different levels of sophistication.
- *Multi-application*: Dynamically loading and unloading of Applications (section 8.6.5), the existence of a well-define Application Programmer interface and synchronized output into different 3D-windows (section 8.3.2) allows the concurrent use of

widely differing applications or multiple instances of the same application (section 13.2.2).

– *Sophisticated Visualization Techniques*: As described in chapter 11 the integration of interaction and focussing techniques like magic lenses and magic boxes as well as the implementation of a newly developed visualization icon for flow visualization – the dashtube – has been successfully applied in Studierstube. The client-server principle explained in section 11.2 which integrated Studierstube as an interaction frontend into a commercial dataflow network visualization system has already been used in different applications.

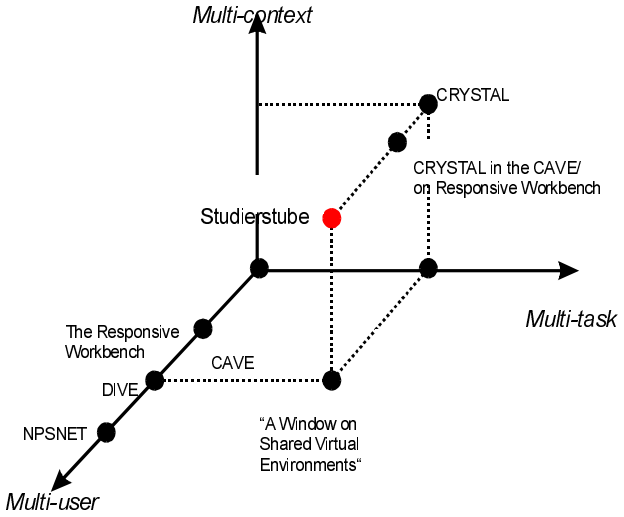


Figure 85: Taxonomy of virtual environments (adapted from: [Tsao1997])

In Figure 85 we contrast the scalability of several VR systems to Workspace in terms of multi-user, multi-tasking and multi-data context operation. The most important contribution of Workspace is that it is scaleable in all three categories.

Studierstube is to our knowledge the only virtual environment today which is able to support multiple concurrent applications (multi-task) communicating with multiple users using multiple individual I/O contexts. This *multiplicity of interaction paths* gives a user a maximum of

flexibilty to work alone or collaborate with other users on different tasks inside a shared environment.

We also introduced the notion of a multi-user aware application (or MUA), which is able to deal with multiple concurrent users at once without the need for monopolization of all system resources. The workspace system is designed to supply a maximum of information regarding users, contexts and environment to the MUAs if desired, but also supports applications designed only to handle one user at a time. This concept proved to be extremely helpful in some of our collaborative projects (chapter 13.3) where legacy code developed for a more conventional form of single-user interaction had to be included.

Studierstube Workspace has proved to stand up against the demands of a wide variety of applications (chapter 13). Many Workspace-specific tasks like focus changes or application loading have been implemented in a way which enables users to transfer skills acquired in desktop environments into augmented reality. For example the concept of a "3D window" – an enclosed section containing interaction space for a single application – has been readily accepted by all users familiar with 2D GUIs.

The application programmer interface has enabled programmers to quickly integrate their code into the Workspace and supplement it with both 6DOF interaction techniques and conventional 2D graphical user interface elements on the PIP. Since the decisions on a consistent policy regarding multi- user applications can not be done at this early stage, the implementation of MUAs still depends heavily on explicit intervention on the application programmers side. The resulting variations in applications policy regarding user interaction are going to aid us in resolving this aspect for future versions of the Workspace. On the other hand the missing global policy may lead to an unneccesary proliferation of interaction conventions for identical tasks, not unlike a similar phenomenon occuring under X-windows, where only popular toolkits like MOTIF introduce some consistency.

14.2 Collaborative Scientific Visualization

By connecting the AR system Studierstube and the visualization system DynSys3D we have initiated a synergistic effect: Researchers who investigate dynamical systems profit from the intuitive interaction techniques available for 3D phase space in Studierstube, and also from the collaborative setting. From an AR researcher's perspective, the behavior and demands of "real" users (designers of dynamical systems) supported the development of Studierstube as a practical tool, and also permitted us to verify that useful work can be done in such a setup. We found that immersion into the virtual space provided by Studierstube is enhanced by collaboration as it leads to increased acceptance of the presented virtual objects.

During our work of investigating the models created with DynSys3D and Studierstube, we have made some encouraging observations in the behavior of the involved users. Users were generally pleased with the option of seeing abstract mathematical concepts such as phase space representations, attractors etc. as concrete, stereoscopic visualizations. They reported an increased understanding of the structures compared to a screen based rendering as provided by standard AVS. In particular they liked to walk around a visualization, look at it from above and below and even stand in its middle. We also observed that social behavior such as pointing to indicate an interesting feature to a peer were frequently used to complement verbal discussion and users gestured as if the visualizations were real objects.

Our users stressed that they were interested in both global features of the dynamical system (such as its overall shape) and local features such as attractors and therefore needed to shift their view with respect to the presented visualization. Consequently, they constantly used all available options for changing the perceived size of the representation - a scale slider on the PIP, stepping back, and finally holding a smaller representation at arm's length.

During the development of a dynamical system, a particular role behavior emerged relatively independent of the application and user group. In the early stages of the dynamical system design readjustments of the AVS/DynSys3D network are often necessary. While some of the users remained immersed, the AVS expert frequently sat down at the workstation to modify the AVS network.

Moving back and forth between immersed view and the screen-based workplace was not perceived problematic because of the see-through capability of the HMD. In many cases the AVS workstation was even operated standing with the HMD on.

As the work progressed, all users tended to stay immersed and simple modifications were made from inside the VE using interface elements on the PIP. Users reported that they were pleased with the rapid feedback for visualization mapping (e. g. the creation of a new streamline takes 1-5 seconds) and the possibility to continue interactive exploration while waiting for such feedback.

We also observed casual educational settings with one teacher and multiple students. Frequently the "students" were researcher colleagues that happened to walk by and expressed interest in the ongoing visualization. These settings have a tendency to involve more than two people and are a very strong argument for a collaborative setting such as ours. The most severe limitations in these cases are usually hardware-bound (rendering capacity, number of HMDs and PIPs, floor space etc.)

While the low resolution of the HMDs was not found to be a significant problem, users disliked the small viewing angle of the HMD, registration errors and lag. They also criticized the absence of a positioning method equally precise to numerical specification of parameter values in AVS dialog boxes. Nevertheless, the comments we received were generally encouraging and sometimes even enthusiastic.

14.3 Unresolved problems

In this section we number special cases or problems which have been encountered during our work and hitherto not been satisfactory solved. We plan to concern our further research with these topics.

14.3.1 User Migration

An important aspect for a natural collaboration in the Workspace is user migration into and out of the environment. It should be possible for a user to completely leave the environment and for new users to

join it. Some applications (especially MUAs) may need to recognize these changes. Not only has the user context to be established or removed, the time when this happens may be crucial. The initial PIP state of a user which joins an application may be considerably different from the one at the start of this application. The necessary message structure is going to be integrated in future releases of the message manager (section 7.6).

14.3.2 Access Rights and Visibility Layers

User-dependent access rights for visibility and interaction make sense in applications where an inherently asymmetrical relationship exist, as discussed in the case study "educational demonstration".

At the moment mutual exclusion mechanisms are only included as means of lowest-level conflict resolvement. There exists no consistent, global system mechanism to mark a window or PIP sheet as private or to display data only for a selected subset of users.

Their implementation via different layers [Szalavári1998b] is already in the final implementation stage and is going to be included in our next release.

14.3.3 Scalability

Most of the described results have been obtained using a single-computer version of *Studierstube*. All graphic output has been rendered on a single system: a SGI Indigo2 Maximum Impact with Multi-Channel Option. To make our system really scalable – supporting an arbitrary number of users – we need a mechanism which allows us to distribute the *Studierstube* functionality over a cluster of computers.

Adding another user would then be possible on a modular basis. It would only involve connecting another workstation/HMD/tracker unit to the cluster.

We have already shown how this can be done on the lowest common level by distributing the Open Inventor database synchronized over a couple of computers [Hesina1999]. Figure 86 shows that only slight changes to the concept in Figure 16 had to be made to accomplish

this. Essentially the Open Inventor internal notification mechanism is used to notify the network layer from changes in the scenegraph and to distribute these to other nodes of the cluster.

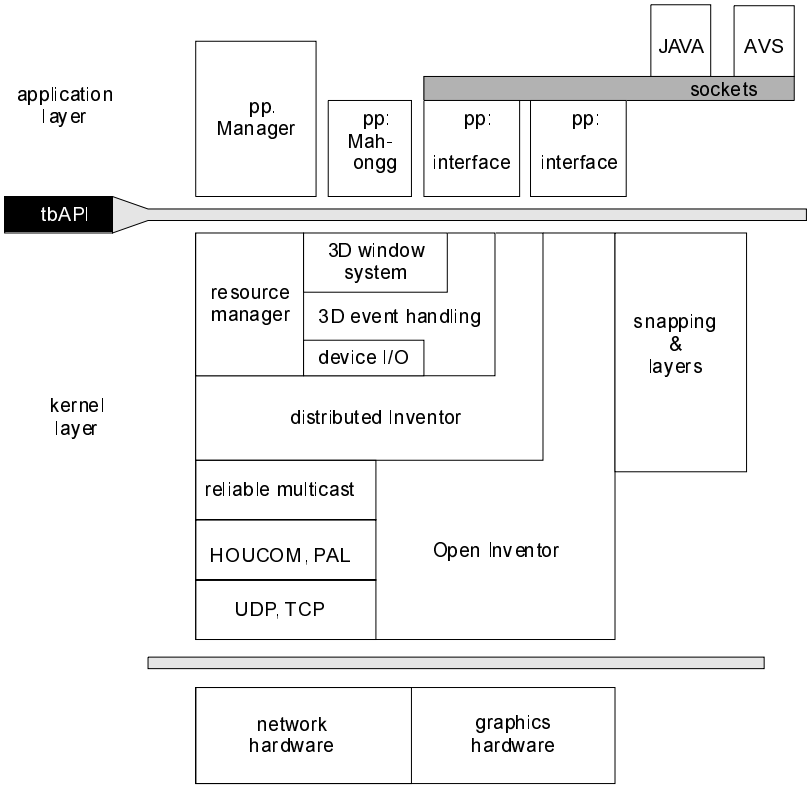


Figure 86: *Distributed Studierstube* architecture schematic

This solution of course distributes *all* changes to the scenegraph and may lead to considerable network loads. An efficient distribution of *Studierstube* would have to include a mechanism to distribute not only graphic data but also parts of the applications like JAVA applets to reduce the network load and the calculation load on the computer acting as master.

15 Conclusion

The subject of this thesis is a shared virtual environment for scientific visualization. We presented *Studierstube*, a multi-user environment employing *augmented reality* – the composition of the real surroundings with computer generated images – to allow the tight cooperation of users in a shared environment of completely virtual objects and real objects enhanced by virtual augmentations.

Our goal was a collaborative work environment for scientific visual visualization and other applications heavily utilizing complex three-dimensional graphics and interaction techniques. The existing implementation of the *Studierstube* concept has already proven our concept to be applicable in many different areas including visualization of dynamical systems, medical visualization, flow visualization, computer aided design and in the development and testing of novel user interface techniques.

While these results are encouraging, still much remains to be done. The problems discussed in section 14.3 only constitute a small selection of the topics for future research which occured to us during our work. Essentially they are of a purely technical nature and are without doubt going to be solved. But other problems remain, some of them of a technical nature too – tracking, prediction, frame rate – but most of them stemming from the same root:

Virtual reality – though already about 25 years old – is still in its infancy.

From the perspective of conventional graphical user interfaces virtual reality is somewhere between the XEROX Dynabook and the first Macintosh: Obviously a good concept, but still lacking the stable technical background and the solid foundation of inexpensive hardware necessary to put it to widespread use.

Another good reason for continuing our work.

"To infinity – and beyond!"
Buzz Lightyear

16 References

[Abraham1992] R. Abraham & C. Shaw: Dynamics - The Geometry of Behavior. Addison-Wesley, 1992.

[Agrawala1997] M. Agrawala, A. C. Beers, B. Fröhlich, I. McDowall P. Hanrahan, and M. Bolas. The Two-User Responsive Workbench: Support for Collaboration Through Individual Views of a Shared Space. Proceedings SIGGRAPH '97, pages 327-332, 1997.

[Angus1995] I. Angus and H. Sowizral. Embedding the 2D Interaction Metaphor in a Real 3D Virtual Environment. Proceedings SPIE, volume 2409, pages 282-293, 1995.

[AVS1992] AVS Developers Guide - Release 4. Advanced Visualisation Systems Inc, 1992.

[Azuma1994] Azuma, R. and Gary Bishop, G. Improving Static and Dynamic Registration in an Optical See-Through HMD. Proceedings of SIGGRAPH '94, pp. 197-204, ACM Press, July 1994.

[Azuma1997] Ronald Azuma. A Survey of Augmented Reality. Presence, 6(4):355-385, 1997

[Badler1993] Norman I. Badler, Michael J. Hollick and John Granieri. Real-time control of a virtual human using minimal sensors. Presence 2 (1), 1993, pp. 82-86.

[Bajura1992] M. Bajura, H. Fuchs, and R. Ohbuchi. Merging Virtual Objects with the Real World: Seeing Ultrasound Imaginery within the Patient. Proceedings of SIGGRAPH'92, (2):203-210, 1992.

[Bajura1995] Bajura, M., and Neumann, U. Dynamic Registration Correction in Augmented-Reality Systems. VRAIS'95, 1995.

[Berger1997] Berger, M. Resolving Occlusion In Augmented Reality: A Contour Based Approach Without 3D Reconstruction. Proceedings of Conference on Computer Vision and Pattern Recognition, IEEE, 1997, Poster Session 1.

[Bier1997] E. Bier, M. Stone, and K. Pier. Enhanced illustration using magic lens filters. IEEE Computer Graphics and Applications, 17(6), pages 62–70, November/December 1997.

[Billinghurst1997] M. Billinghurst, S. Weghorst, and T. Furness III. Shared Space: An Augmented Reality Approach for Computer Supported Collaborative Work. Virtual Reality Research Development and Applications, 3(1):25-36, 1997.

[Bloomenthal1990] J. Bloomenthal. Calculation of Reference Frames Along a Space Curve. Graphic Gems, pages 567-571. Academic Press, Cambridge, MA, 1990.

[Breen1996] David E. Breen, Ross T. Whitaker, Eric Rose and Mihran Tuceryan. Interactive Occlusion and Automatic Object Placement for Augmented Reality. Computer Graphics Forum (Proceedings of EUROGRAPHICS'96), 15(3):C11-C22, 1996.

[Brooks1990] F. Brooks Jr. et. al.: Project GROPE - Haptic Displays for Scientific Visualisation. Proceedings of SIGGRAPH 90, pp 177-185, 1990.

[Bryson1991] S. Bryson and C. Levitt: The Virtual Wind Tunnel. Proceedings of IEEE Visualization 91, 17-25, 1991.

[Bryson1993a] S. Bryson: The Distributed Virtual Windtunnel. SIGGRAPH 93 Course Notes 43, 3.1-3.10, 1993.

[Bryson1993b] Bryson, S. Measurement and calibration of static distortion of position data from 3D trackers. Virtual Reality for Visualisation (IEEE Visualisation), pages 179-189, 1993.

[Cabral1993] B. Cabral and L. C. Leedom. Imaging vector fields using line integral convolution. In Proceedings SIGGRAPH '93, pages 263–272, 1993.

[Carey1997] Rikk Carey, Gavin Bell: The Annotated Vrml 2.0 Reference Manual. Addison-Wesley, 1997.

[Carlsson1993] C. Carlsson, O. Hagsand: DIVE- A platform for multi- user virtual environments. Computers & Graphics, Vol. 17, No. 6, pp. 663-669, 1993.

[Conner1992] D. B. Conner, S.S. Snibbe, K. P. Herndon, D. C. Robbins, R. C. Zeleznik, and A. van Dam. Three-Dimensional Widgets. Proc.SIGGRAPH Symposium on Interactive 3D Graphics, 25(2):183-188, 1992.

[Crawfis1993] R. Crawfis and N. Max. Texture splats for 3D scalar and vector field visualization. In IEEE Visualization '93 Proceedings, pages 261–266. IEEE Computer Society, October 1993.

[Cruz-Neira1992] C. Cruz-Neira et al. The CAVE: Audio Visual Experience Automatic Virtual Environment. Communications of the ACM, 35(6):65,1992.

[Cruz-Neira1993a] C. Cruz-Neira, D. Sandin, and T. DeFanti. Surround-Screen Projection-Based Virtual Reality: The Design and Implementation of the CAVE. Proceedings of SIGGRAPH'93, pages 135-142, 1993.

[Cruz-Neira1993b] Cruz-Neira, C. et al. Scientists in Wonderland: A Report on Visualization Applications in the CAVE Virtual Reality Environment. IEEE 1993 Symposium on Research Frontiers in Virtual Reality. Visualization '93. San Jose, CA, October 1993. pp. 59-66.

[Roy1994] Roy, T., Cruz-Neira, C., and DeFanti, T.A. Cosmic Worm in the CAVE: Steering a High Performance Computing Application from a Virtual Environment. Special issue on Networks and Virtual Environments of Presence: Teleoperators and Virtual Environments, 1994.

[Dias1997] J. Dias, R. Galli, A. Almeida, C. Belo, J. Rebordao: mWorld: A Multiuser 3D Virtual Environment. IEEE Computer Graphics and Applications, Vol. 17, No. 2, pp. 55-65, March-April 1997.

[Dykstra1993] P. Dykstra: X11 in Virtual Environments: Combining Computer Interaction Methodologies. Proceedings of IEEE Symposium on Research Frontiers in Virtual Reality, pp. 118-119, 1993.

[Faugeras1993] Faugeras, O.: Three-dimensional computer vision: a geometric viewpoint. ISBN 0-262-06158-9, MIT Press, 1993.

[Faure1998] Francois Faure. Interactive solid animation using linearized displacement constraints. Proceedings of the Eurographics Workshop on Animation and Simulation, 1998.

[Feichtinger1996] G. Feichtinger, G. Fischel, E. Gröller, A. Prskawetz: Despotism and Anarchy in Ancient China: Visualizing the Dynastic Cycle. Jahrbuch für Wirtschaftswissenschaften 47/1, publisher Vandenhoeck & Ruprecht, Goettingen, Germany: 1-13, 1996.

[Feiner1990] S. Feiner and C. Beshers: Worlds Within Worlds: Metaphors for Exploring N-Dimensional Virtual Worlds, Proceedings of ACM UIST '90, pp. 76-83, 1990.

[Feiner1992] Feiner, S., MacIntyre, B., and Seligmann, D. Annotating the Real World with Knowledge-Based Graphics on a See-Through Head-Mounted Display. Proceedings of Graphics Interface'92, 78-85, 1992.

[Feiner1993a] S. Feiner, B. MacIntyre, M. Haupt, and Solomon. Windows on the World: 2D Windows for 3D Augmented Reality. Proceedings of UIST'93, pages 145- 155, 1993.

[Feiner1993b] S. Feiner, B. MacIntyre, D. Seligmann: Knowledge-Based Augmented Reality. Communications of the ACM, Vol. 36, No. 7, pp. 53-62, 1993.

[Fuhrmann1997] A. Fuhrmann, H. Löffelmann, D. Schmalstieg: Collaborative Augmented Reality: Exploring Dynamical Systems. Proc. of IEEE Visualization 1997, pp. 459- 462, November 1997.

[Fuhrmann1998a] Anton Fuhrmann, Helwig Löffelmann, Dieter Schmalstieg, and Michael Gervautz. Collaborative Visualization in Augmented Reality. IEEE Computer Graphics and Applications, 18(4):54-59, July/August 1998.

[Fuhrmann1998b] A. Fuhrmann and E. Gröller: Real-Time Techniques for 3D Flow Visualization. Proc. of IEEE Visualization 1998, pp. 305-312, November 1998.

[Fuhrmann1999] Fuhrmann, A., Hesina, G., Faure, F. and Gervautz, M: Occlusion in Collaborative Augmented Environments. Proceedings 5th Eurographics Workshop on Virtual Environments, ISBN 3-211-83347-1, pages 179-190, Springer-Verlag, Wien, 1999.

[Gerald1993] M. Gerald-Yamasaki: Cooperative visualization of computational fluid dynamics. Proceedings of Eurographics'93, pp. 497-508, 1993.

[Gettis1990] J. Gettis, P. Carlton, S. McGregor: The X Window System, version 11. Software Practice and Experience, 20(S2), October 1990.

[Gibbs1998] S. Gibbs, C. Arapis, C. Breiteneder: TELEPORT - Towards immersive copresence. ACM Multimedia Systems Journal, 1998.

[Hesina1999] G. Hesina, D. Schmalstieg, A. Fuhrmann, W. Purgathofer: Distributed Open Inventor: A Practical Approach to Distributed 3D Graphics To appear in: Proceedings of ACM Virtual Reality Software & Technology '99 (VRST'99), London, December 20-22, 1999.

[Hoff1996] W. A. Hoff, T. Lyon, and K. Nguyen, Computer Vision-Based Registration Techniques for Augmented Reality, Proc. of Intelligent Robots and Computer Vision XV, Vol. 2904, in Intelligent Systems & Advanced Manufacturing, SPIE, Boston, Massachusetts, Nov. 19-21, pp. 538-548, 1996.

[Interrante1997] V. Interrante and Ch. Grosch. Strategies for effectively visualizing 3D flow with volume LIC. In Proceedings of Visualization ’97, pages 421–424, 1997.

[Janin1993] Janin, A., Mizell, D. and Caudell, T. Calibration of head-mounted displays for augmented reality applications. Proceedings of the Virtual Reality Annual International Symposium (VRAIS '93), pages 246–255, September 1993.

[Jobard1997] B. Jobard and W. Lefer. Creating evenly spaced streamlines of arbitrary density. In Wilfrid Lefer and Michel Grave, editors, Visualization in Scientific Computing, pages 43–55. Springer-Wien-NewYork, 1997.

[Krüger1994] W. Krueger and B. Froehlich Visualization Blackboard: The Responsive Workbench (virtual work environment) IEEE Computer Graphics and Applications, 14(3), pp. 12-15, May 1994.

[Krüger1995] Krüger W., C. Bohn, B. Fröhlich, H. Schüth, W. Strauss, and G. Wesche. The Responsive Workbench: A Virtual Work Environment. IEEE Computer, 28(7):42-48, 1995.

[Kuhl1995] J. Kuhl, D. Evans, Y. Papelis, R. Romano, and G. Watson. The Iowa Driving Simulator: An Immersive Research Environment. IEEE Computer, 28(7): pp. 35-42, 1995.

[Kutulakos1998] Kutulakos, K. and Vallino, J. Calibration-Free Augmented Reality. IEEE Transactions on Visualization and CG, 4(1), pp. 1-20, January 1998.

[Lehner1997] V. Lehner, T. DeFanti: Distributed VR: Supporting Remote Collaboration in Vehicle Design. IEEE Computer Graphics and Applications, Vol. 17, No. 2, pp. 13-17, March-April 1997.

[Löffelmann1997a] H. Löffelmann, E. Gröller: DynSys3D: A workbench for developing advanced visualization techniques in the field of three-dimensional dynamical systems. WSCG 97, Plzen, Czech Republic, 301-310, 1997.

[Löffelmann1997b] H. Löffelmann, L. Mroz, E. Gröller: Hierarchical Streamarrows for the Visualization of Dynamical Systems. To appear in: Proc. of the 8th Eurographics Workshop on Vis. in Sci. Computing, April 1997

[Löffelmann1997c] H. Löffelmann, E. Gröller: Visualizing Poincaré Maps together with the underlying flow. ftp:// ftp.cg.tuwien.ac.at/pub/TR/97/TR-186-2-97-06Paper.ps.gz

[Löffelmann1997d] H. Löffelmann, L. Mroz, E. Gröller, W. Purgathofer. Stream Arrows: Enhancing the Use of Stream Surfaces for the Visualization of Dynamical Systems. Visual Computer, Springer, Vol. 13(8), pages. 359-369, 1997.

[Macedonia1994] M. R. Macedonia, M. J. Zyda, D. R. Pratt, P. T. Barham, and S. Zeswitz. NPSNET: A Network Software Architecture for Large-Scale Virtual Environment. Presence, 3(4):265-287, 1994.

[Marca1992] D. Marca, G. Bock: Groupware: Software for computer-supported cooperative work. IEEE Computer Society Press, 1992.

[Max94] N. Max, R. Crawfis, and Ch. Grant. Visualizing 3D velocity fields near contour surfaces. In IEEE Visualization ’94 Proceedings, pages 248–255. IEEE Computer Society, October 1994.

[McReynolds1998] Tom McReynolds, David Blythe, Brad Grantham, and Scott Nelson. Advanced Graphics Programming Techniques Using OpenGL. SIGGRAPH `98 Course notes

[Milik1996] Milik: Dynamics of Mixed-mode Oscillations. PhD thesis, Vienna U. of Technology, Austria, 1996.

[Obeysekare1996] U. Obeysekare et al.: Virtual Workbench - A Non-ImmersiveVirtual Environment for Visualizing and Interacting with 3D Objects for Scientific Visualization. Proceedings of Visualization 96, pp. 345-350, 1996.

[Oishi1995] Oishi T. and S. Tachi, S. Methods to Calibrate Projection Transformation Parameters for See-Through Head-Mounted Displays. Presence, 5(1), pp. 122-135, 1995.

[Pang1997] A. Pang, C. Wittenbrink: Collaborative Visualization with CSpray. IEEE Computer Graphics & Applications, 32-41, March-April 1997.

[Pausch1992] Pausch, R., Crea, T., and Conway, M. 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.

[Pausch1993] R. Pausch, T. Burnette, M. Conway, R. DeLine, R. Gossweiler: Alice: A Rapid Prototyping System For Virtual Reality. UIST'93 (1993)

[Petrov1992] V. Petrov, S. Scott, K. Showalter: Mixed-mode oscillations in chemical systems. Journal of Chemical Physics, Vol. 97, No. 9, pp. 6191-6198, 1992.

[Post1993] F. H. Post and T. van Walsum. Fluid flow visualization. In H. Hagen, H. Müller, and G. M. Nielson, editors, Focus on Scientific Visualization, pages 1–40. Springer, 1993.

[Press1988] Press, W., Flannery, B., Teukolsky, S., and Vetterling, W.. Numerical Recipes in C. Cambridge University Press, 1988.

[Raskar1998] Ramesh Raskar, Greg Welch, Matt Cutts, Adam Lake, Lev Stesin and Henry Fuchs. The Office of the Future: A Unified Approach to Image-Based Modeling and Spatially Immersive Displays. SIGGRAPH 98 Conference Proceedings, Annual Conference Series, pp. 179-188, Addison Wesley, July 1998.

[Robertson1989] G. Robertson, S. Card, and J. Mackinlay. The Cognitive Coprocessor Architecture for Interactive User Interfaces. Procedings of ACM CHI'89, pages 10-18, 1989.

[Schaufler1998] G. Schaufler and D. Schmalstieg. Sewing Worlds Together With SEAMS. Technical Report TR-186-2- 98-11, Institute of Computer Graphics 186-2, Technical University of Vienna, Vienna, Austria, August 1998.

[Schmalstieg1998] D. Schmalstieg, A. Fuhrmann, Z. Szalavari, M. Gervautz: Studierstube - An Environment for Collaboration in Augmented Reality Extended abstract appeared in proceedings of Collaborative Virtual Environments '96, Nottingham, UK, Sep. 19-20, 1996. Full paper in: Virtual Reality - Systems, Development and Applications, Vol. 3, No. 1, pp. 37-49, 1998.

[Schmalstieg1999] D. Schmalstieg, M.Encarnação, Zs. Szalavári: Using Transparent Props For Interaction With The Virtual Table. To appear in: Proceedings of SIGGRAPH Symposium on Interactive 3D Graphics '99, Atlanta, GI, April 26-28, 1999.

[Schroeder1996] W. Schroeder, K. Martin, B. Lorensen: The visualization toolkit: an object-oriented approach to 3D graphics. Prentice Hall, Inc., ISBN 0-13-199837-4, 1996.

[Shaw1993] A. Shaw, M. Green, J. Liang, and Y. Sun: Decoupled simulation in virtual reality with the MR toolkit. ACM Transactions on Information Systems, Vol. 11, No. 3, pp. 287-317, 1993.

[Smith1983] D. Smith, C. Irby, R. Kimbrall, W. Verplank, E. Harslem: Designing the Star user interface. BYTE, pp. 254-258, April 1983.

[Sowizral1998] Henry A. Sowizral, Kevin Rushforth, Michael Deering: The Java 3D Specification. Addison-Wesley, 1998.

[Stalling1995] D. Stalling and H.Ch. Hege. Fast and resolution independent line integral convolution. In Robert Cook, editor, Computer Graphics (SIGGRAPH 1995 Proceedings), pages 249–256, August 1995.

[State1996a] Andrei State, Gentaro Hirota, David T. Chen, William F. Garrett, and Mark A. Livingston. Superior Augmented-Reality Registration by Integrating Landmark Tracking and Magnetic Tracking. Proceedings of SIGGRAPH 96, Annual Conference Series 1996, ACM SIGGRAPH, pp. 429-438.

[State1996b] Andrei State, Mark A. Livingston, Gentaro Hirota, William F. Garrett, Mary C. Whitton, Henry Fuchs, and Etta D. Pisano (MD). Technologies for Augmented-Reality Systems: realizing Ultrasound-Guided Needle Biopsies. Proceedings of SIGGRAPH 96, Annual Conference Series 1996, ACM SIGGRAPH, pp. 439-446.

[Strauss1992] P. Strauss and R. Carey. An Object Oriented 3D Graphics Toolkit. Proceedings of SIGGRAPH'92, (2):341-347, 1992.

[Szalavári1997] Z. Szalavári., M. Gervautz: The Personal Interaction Panel - A Two-handed Interface for Augmented Reality. Proc. EUROGRAPHICS 97, Budapest, Hungary, pp. 335-346, 1997.

[Szalavári1998a] Szalavari, Z., Schmalstieg, D., Fuhrmann, A., Gervautz, M. Studierstube - An Environment for Collaboration in Augmented Reality, Virtual Reality: Research, Development & Applications, 1998

[Szalavári1998b] Z. Szalavári., E. Eckstein and M. Gervautz: Collaborative Gaming in Augmented Reality. Proc. of VRST'98, Taipei, Taiwan, pp.195-204, November 2-5, 1998.

[Taylor1993] R. Taylor et. al.: The Nanomanipulator: A Virtual Reality Interface for a Scanning Tunneling Microscope. Proceedings of SIGGRAPH 93, pp. 127-134, 1993.

[Tsai1986] Tsai, R.. An efficient and accurate camera calibration technique for 3D machine vision. Proceedings CVPR '86, pages 364-374, IEEE, June 1986.

[Tsao1997] J. Tsao and Ch. J. Lumsden. CRYSTAL: Building Multicontext Virtual Environments. Presence, 6(1):57-72, 1997.

[Tuceryan1995] Tuceryan, M., Greer, D., Whitaker, R., Breen, D., Crampton, C., Rose, E., and Ahlers, K., Calibration Requirements and Procedures for Augmented Reality, IEEE Trans. on Visualization and Computer Graphics, September 1995.

[Turk1996] G. Turk and D. Banks. Image-guided streamline placement. In Proceedings SIGGRAPH 1996, pages 453–459, 1996.

[Viega1996] J. Viega, M.J. Conway, G. Williams, and R. Pausch. 3D magic lenses. In ACM UIST'96 Proceedings, pages 51–58. ACM, 1996.

[Waldrop 1995] M. Waldrop, Marianne S., Pratt, Shirley M., Pratt, David R., McGhee, Robert B., Falby, John S. and Zyda, Michael J. Real-time Upper Body Articulation of Humans in a Networked Interactive Virtual Environment. Proceedings of the First ACM Workshop on Simulation and Interaction in Virtual Environments, University of Iowa, 13 - 15 July 1995, pp. 210-214.

[Ward1992] Mark Ward, Ronald Azuma, Robert Bennett, Stefan Gottschalk, Henry Fuchs. A Demonstrated Optical Tracker with Scalable Work Area for Head-Mounted Display Systems. Proceedings of 1992 Symposium on Interactive 3D Graphics (Cambridge, Mass., March 29 - April 1 1992), 43-52

[Waters1997] R. Waters, D. Anderson, J. Barrus, D. Brogan, M. Casey, S. McKeown, T. Nitta, I. Sterns, and W. Yerazunis. Diamond Park and Spline: Social Virtual Reality with 3D Animation, Spoken Interaction and Runtime Extendability. Presence, 6(4):461-481, 1997.

[Wegenkittl1997a] R. Wegenkittl and E. Gröller. Fast oriented line integral convolution for vector field visualization via the internet. In IEEE Visualization '97 Proceedings, pages 309–316. IEEE Computer Society, October 1997.

[Wegenkittl1997b] R. Wegenkittl, E. Gröller, W. Purgathofer, Animating Flowfields: Rendering of Oriented Line Integral Convolution, Computer Animation '97, pages 15-21, IEEE Computer Society, June 1997.

[Whitaker1995] Whitaker, R., Crampton, C., Breen, D., Tuceryan, M., and Rose, E. Object Calibration for Augmented Reality. Proc. EUROGRAPHICS'95, pp. 15-27, 1995.

[Wijk1993] J. J. van Wijk. Flow visualization with surface particles. IEEE Computer Graphics & Applications, 13(4):18–24, July 1993.

[Williams1983] L. Williams. Pyramidal parametrics, Computer Graphics (SIGGRAPH 1983 Proceedings), 17(3), pages 1-11, July 1983.

[Wloka1995] Matthias M. Wloka and Brian G. Anderson. Resolving Occlusion in Augmented Reality. Proceedings of Symposium on Interactive 3D Graphics, ACM SIGGRAPH, 1995, pp. 5-12.

[Woo1997] Mason Woo (Contributor), Jackie Neider, Tom Davis, OpenGL Architecture Review board, "OpenGL Programming Guide : The Official Guide to Learning OpenGL, Version 1.1", 2nd edition (January 1997), Addison-Wesley Pub Co; ISBN:201461382

[Wood1997] J. Wood, H. Wright and K. Brodlie: Collaborative Visualization: Proc. of IEEE Visualization, 253-259, 1997.

[Zeleznik1996] Zeleznik, R.C., Herndon, K., Hughes, J.: Sketch: An Interface for Sketching 3D Scenes, Proceedings of SIGGRAPH'96, 163-170.