A Toolset for Visualization, Interaction, and Rendering in Virtual Environments

Dissertation
der Fakultät für Informatik
der Eberhard-Karls-Universität Tübingen
zur Erlangung des Grades eines
Doktors der Naturwissenschaften
(Dr. rer. nat.)

vorgelegt von
Dipl.-Inform. Stanislav L. Stoev
aus Tübingen

Tübingen
2001

Tag der mündlichen Qualifikation:	*19.12.2001*
Dekan:	Prof. Dr. A. Zell
Erster Berichterstatter:	Prof. Dr.-Ing. Dr.-Ing. E.h. W. Straßer
Zweiter Berichterstatter:	Prof. Dr. W. Rosenstiel

To my family

Acknowledgements

The presented work was conducted during my occupation as Research Assistant at the Interactive Graphics Systems Group of the Wilhelm-Schickard-Institute for Computer Science (WSI/GRIS) at the University of Tübingen.

First and foremost, I would like to thank Prof. Wolfgang Straßer for his supervision and support throughout my work since the very beginning to the final correction of the present work. The referee Prof. Rosenstiel I would like to thank for rendering his expert opinion and for many valuable hints with respect to my work.

I would furthermore like to express my gratitude to those who supported me and were of great help throughout the cooperation and surveys. I would especially like to thank Dieter Schmalstieg (Dr. Dide, Vienna University of Technology), Tobias Hüttner (Egi.sys AG), Konrad Burgbacher and Wolfgang Dietz (University of Stuttgart), and Ingmar Peter (WSI/GRIS) for the fruitful cooperation within the diverse common projects.

Further thank is due to the colleagues and students of the WSI/GRIS for the good cooperation and helpful initiatives, especially to the students Matthias Feuerer and Michael Ruckaberle, who with their work delivered valuable contributions to the implementation of the presented system.

Special thanks need to be expressed to my family, to Veronika Dossinger (Vroni) for the overall support during the last four years, my father for the appreciation and for correcting and improving the English writing of this thesis, to my mother for the inspiration and many valuable hints, and to my sister, who tried to shorten my spare time in many various ways.

Tübingen, 22.11.2001

Stanislav L. Stoev

Abstract

Virtual reality (VR) applications play a significant role in computer graphics nowadays. They act as an interdisciplinary bridge between the visualization of data and their interactive exploration on the one hand, and various other research areas like medicine, tele-robotics, computer aided design (CAD), visualization of simulation data, etc. on the other hand. This work addresses some of the most pressing issues considering the assembly of a virtual reality system: the data preprocessing and preparation, the data visualization, the interaction with the data, and their display.

Elaborating on each of these components, the work presents problem-tailored algorithms, as well as general strategies for tackling the introduced problems. In particular, a new algorithm for segmenting image and volume data is discussed, addressing foremost medical applications in general. The proposed method offers a solution to the up to now unsolved problem of segmenting objects of interest out of given input data. Moreover, a detailed discussion on visualization issues of data containing a time dimension is provided, demonstrating a set of visualization and interaction props for the analysis of historical events in time. The described set of tools has proved to be a valuable companion for historians, helping them to simultaneously visualize various parameters of interest and to study historical events.

In addition, the navigation in general is analyzed, as well as the remote object manipulation in virtual environments, presenting a new interaction metaphor: the *through-the-lens* metaphor. Based on this new concept, a set of navigation tools is introduced, which exploit features of existing techniques, attempting to overcome their limitations and to enable more powerful interaction. The through-the-lens concept has been also applied to develop a new remote manipulation technique for adjusting various parameters of objects out of the user's reach, while still at their original location.

Finally, in order to enable fast rendering of the visualized data, two algorithms for accelerating the rendering performance of a virtual reality system are proposed. Each of them exploits particular characteristics of the representation: (1) an image-based rendering approach based on the features of the introduced through-the-lens tools is presented, (2) the stereo rendering of large scenes is addressed in general.

To summarize, the contributions of this work facilitate the creation of virtual reality applications, as well as the interaction with the visualized data in particular. The presented interaction props and rendering algorithms provide a set of easy to use, while valuable techniques based on new and extending existing approaches.

Kurzfassung

Heutzutage spielen Anwendungen aus dem Gebiet der virtuellen Realität (VR) eine wichtige Rolle in der Computergraphik. Sie ermöglichen die Datenvisualisierung und die interaktive Datenerkundung in verschiedenen Forschungsgebieten wie Medizin, Telerobotik, CAD-Systeme, Visualisierung von Simulationsdaten usw. Diese Arbeit beschäftigt sich mit einigen der interessantesten Kernpunkte der Erstellung eines VR-Systems: der Vorbereitung und Vorverarbeitung der Daten, deren Visualisierung, der Interaktion mit den Daten und deren Ausgabe.

Eingehend auf jede einzelne Komponente, stellt diese Arbeit problem-zugeschnittene Algorithmen, sowie allgemeine Strategien zur Lösung der entstehenden Probleme vor. Insbesondere wird ein neuer Algorithmus zur Segmentierung von Bild- und Volumendaten eingeführt, der vor allem für medizinische Anwendungen relevant ist. Das präsentierte Verfahren bietet eine Lösung des bisher ungelösten Problems der Segmentierung von Objekten aus vorliegenden Eingabedaten. Weiterhin wird detailliert die Visualisierung von Daten erläutert, die einen Zeitparameter enthalten. Dabei wird eine Reihe von Visualisierungs- und Interaktionswerkzeugen zur Analyse historischer Daten demonstriert. Die vorgestellten Techniken haben sich als wertvolle Werkzeuge für Historiker erwiesen, die damit eine Vielzahl verschiedener Parameter gleichzeitig graphisch darstellen und historische Ereignisse studieren können.

Weiterhin wird der Frage nach der Navigation und der Manipulation entfernter Objekte nachgegangen und eine neue Interaktionsmetapher vorgestellt: die Durch-Die-Linse-Metapher (*through-the-lens*). Basierend auf diesem Konzept werden verschiedene Navigationswerkzeuge präsentiert, die die Eigenschaften bekannter Techniken instrumentalisieren, jedoch deren Einschränkungen umgehen und eine leistungsfähige Interaktion ermöglichen. Das Durch-Die-Linse-Konzept wird ebenfalls verwendet, um eine neue Technik zur Manipulation entfernter Objekte zu entwickeln. Diese dient dazu, verschiedene Parameter von Objekten außerhalb der Reichweite des Benutzers, jedoch in ihrer natürlichen Umgebung, einzustellen.

Schließlich werden zwei Algorithmen vorgestellt, die eine schnelle Darstellung der zu visualisierenden Daten in einer virtuellen Umgebung ermöglichen. Jeder dieser Algorithmen nutzt typische Eigenschaften der Repräsentation. Erstens wird ein bild-basierter Ansatz vorgestellt, der auf Merkmalen des Durch-Die-Linse-Konzeptes beruht. Zweitens wird auf die Stereo-Ausgabe von großen Szenen im Allgemeinen eingegangen.

Der Beitrag dieser Arbeit liegt darin, die Interaktion mit Daten, sowie deren Visualisierung in einer virtuellen Umgebung zu unterstützen. Die vorgestellten Interaktionswerkzeuge und Darstellungsalgorithmen zeigen, dass sie einfach einzusetzen und trotzdem mächtige Techniken sind, die auf neuen Konzepten basieren und existierende Methoden erweitern.

Contents

Chapter 1

Introduction and Motivation

Virtual reality (*VR*) entails the use of advanced technologies, including computer graphics, multidimensional input, large screen displays, and head mounted displays, to produce a simulated, thus, *virtual environment* (*VE*) that users perceive as comparable to real world objects and events. The major aim of a VR-application is to let the user feel the artificial world as real as possible. In other words, the user interacts with displayed images, moving and manipulating virtual objects, and performing other actions in a way that creates a feeling of actual presence in the simulated environment.

The virtual reality area has been subject of extensive research over the past years. Many researchers have addressed various issues in this field reaching from hardware components through software architectures and models to human factors and human computer interaction.

This work addresses the way of creating a virtual reality application. In particular, it deals with issues concerning the single steps towards creating real feeling, powerful, versatile, and efficient virtual reality applications. In order to accomplish this task, we first have to identify what we intend to present. In general, this is some kind of data, which has to be prepared in a way that the user can interactively work with it. Thus, the first step towards creating an interactive virtual environment is the data acquisition. Once the data have been acquired, it has to be processed in order to enable further visualization and interaction. The next step is the design of the application's architecture. In our context, this term is narrowed to include the human computer interface, the interaction design, the data visualization, and the rendering. We endeavor to abstract from the underlying realization and source code. Where this is not possible, short details are given.

Finally, the last main component is the hardware. This is a factor defining to a great degree the possible interaction type, the user interface, and even the application area. The techniques presented throughout this work are developed on a particular hardware setup. Nevertheless, they are not limited to our setup, but are in general applicable to various immersive scenarios and different input and output devices. Whenever a hardware component is strictly required, this is pointed out.

1.1 Overview and Historical Notes

The first immersive virtual environment has been presented more than 30 years ago [Sutherland, 1968]. Ivan Sutherland described a system involving a tracked head-mounted display and "real time three dimensional computer graphics" based on two CRTs. Since these first efforts towards creating the illusion of being surrounded by a virtual world, many VR-setups and applications have been presented in the literature.

The main features defining the difference between a traditional desktop application and a virtual environment are the head tracking, the stereoscopic vision, the three-dimensional interaction, and the large projection displays used. Thus, there have been four main research directions defining the progress and the quality of a virtual reality setup over the past thirty years considering the involved hardware: the tracking technology, the computer graphic hardware, the display technology, and haptic feedback technology.

The development in the computer graphics area was mainly driven by the game industry and thus, led to more reliable, faster, and cheaper graphics hardware. This fact has revolutionized not only the gaming industry, but also the virtual reality. In the early days, millions of dollars have been needed to set up a VR-lab. This cost was mainly caused by the graphics hardware, which has to be fast enough to guarantee interactive frame rates for displaying complex, thus realistically looking scenes on multi-screen stereo projections or head mounted displays. Since these requirements are met even by consumer graphics hardware nowadays, cheaper graphics boards can be used to build a modern VR-setup.

The next component, the display technology, has also dramatically improved since the introduction of the first VR-systems [Cruz-Neira et al., 1993, Agrawala et al., 1997]. Not only similar systems can be built at fraction of the costs, but also new scenarios (like powerwall, virtual table, etc.) have been implemented enabling application development and research in the VR-area [Schmalstieg et al., 1999].

On the other hand, the tracking hardware, applied for carrying out the interaction in a VE, is still "special purpose" hardware, which stability and precision significantly improved over the past years. The precise tracking devices are the key to head/hand tracking and implementation of virtual tools. The latter in turn predetermine the convenience of the interaction in a virtual environment. Even more, they make up a *real*-feeling virtual environment. In the first VR-systems, it was sufficient to track the user's head in order to generate and display the appropriate images. Nowadays, we can observe the utilization of various input devices for position tracking, data gloves, and even devices offering haptic feedback [Brooks, Jr. et al., 1990, Hirose et al., 2001, Gibson, 1998, Darken et al., 1997, Iwate et al., 2001].

Looking at the software, we also encounter significant progress during the past years. Many of the VR-applications are meanwhile well beyond the experimental stage. Some are even daily utilized in various medical applications [Rothbaum et al., 2000, Rothbaum and Hodges, 1999, Hodges et al., 1999], tele-robotics [Heguy et al., 2001], scientific simulations and visualization [Jaswa, 1997, Rau et al., 1998, Weiskopf, 2000], and other areas. This progress is mainly based on two development directions: the research on rendering techniques in the classical computer graphic and on user interfaces and interaction techniques in virtual environments.

To the first group count algorithms in the areas of model simplification, multi resolution representation, lighting and shading, hidden surface removal, geometry database management, texture mapping, image based rendering, etc. These approaches help the developers of VR-applications to focus on the VR-components: the interaction with, and the behavior of the applications. They enable more realistic presentation and rendering of complex scenes at interactive frame rates. The latter is a crucial requirement for every VE. A system, which does not run fast enough, (ca. 25 fps) may cause motion sickness, fatigue, and perceptible ill effects.

On the other hand, the user interfaces and the interaction techniques, still provide an enormous challenge for the designers of VR-systems. Since one of the primary goals of a *virtual* environment is to create for the user the illusion of being surrounded by a *real* world, the degree of realism and detail richness has to be increased, while enabling the participant to interact with the synthetic world in a natural way. The developments in this area can be divided in two groups: the adaptation of techniques, which have and such, which do not have a counterpart in the real world. Even though, many of the natural navigation and interaction abilities of the human are well understood, it is hard, often impossible to use this knowledge to design tools and techniques for the virtual world scenario. Furthermore, the aim of many virtual worlds is to provide powerful exploration capabilities, which might not have a counterpart in the real life, while offering valuable interaction aids tailored to the particular application or task.

Independent of the interaction, however, the goal of a virtual environment is to provide visualization better than common desktop computers provide. This can only be achieved when sound *visualization* and *interaction* techniques are applied, exploiting the typical features of a VR-system. Moreover, the visualization heavily depends on the underlying data, how it is prepared, displayed, and what type of interaction is supported.

1.2 Problem Statement, Goals, and Contributions of This Work

Putting it together, the main components of a virtual reality application can be divided in two main categories: hardware and software components. The hardware has to be present in order to construct a virtual environment. This work is based on a particular hardware, discussed in detail in Chapter 2, on which we demonstrate the proposed concepts and ideas. However, our hardware setup does not restrict the application of the presented tools to these particular hardware components. In the remainder of this work, we concentrate on the software issues. We elaborate on concrete applications from particular science areas, as well as interaction concepts and rendering techniques in general. The various topic categories, which we discuss here, are shown in Figure 1.1.

1.2.1 Scientific Visualization

First, we address the data processing and visualization problem in two particular cases: (a) the visualization of 4D data consisting of space and time dimensions and (b) the extraction of object boundaries out of 2D images and 3D volume data. The second problem, the segmentation of 2D and 3D data, is up to now an unsolved problem in the image processing and computer vision area. The target of the segmentation is to extract features of different objects contained in an input dataset. In the past years, the watershed transformation has been widely used for performing this task. It is generally performed on the gradient image of the original data. The watershed transformation extracts basins out of the data. The dams of these basins lie more often than not on boundaries between objects.

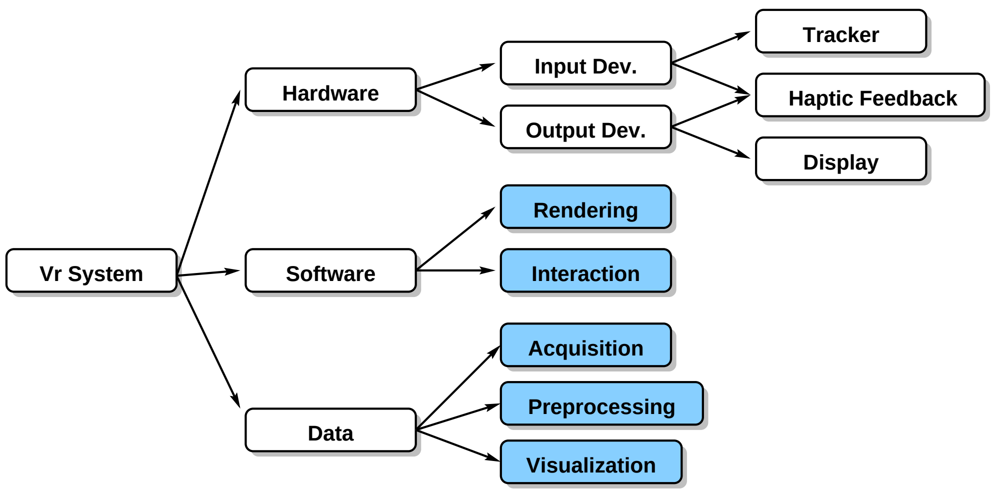


Figure 1.1: Components of a virtual reality application. The marked components are addressed throughout this work.

Unfortunately, the watershed transformation has the problem of over-segmenting the input data. Thus, a basin merging criteria have to be applied in order to extract meaningful areas out of the basins containing data. The main difficulty of this step is that often the diverse objects contained in the input define locally different criteria for basin merging. However, the conditions for the merging are defined globally and fail to cover the features of all single objects in the input data. Furthermore, the watershed transformation is in its nature a global transformation. That is, the watershed transformation is applied to the entire input data and produces a segmentation of it. Therefore, our goal in this computer graphics area was to examine and develop new algorithms, making use of local object characteristics and apply them locally in order to segment a single object rather than the entire input data. This in turn would allow precise segmentation of particular structures out of the given dataset.

The resulting algorithm, presented in this work, supports fast and accurate extraction of objects of interest with a little user input, exploiting only local features of the marked objects. Therefore, we propose a set of criteria for merging adjacent basins, implementing a form of a controlled region growth algorithm. The local application and the proposed merging criteria not only significantly improve the segmentation results, but also speed up the application of the watershed transformation, especially when a volume dataset is considered. In this way, we developed an approach overcoming the most significant drawbacks of the original watershed transformation, while implementing the basic idea and the advantages of the latter.

The second visualization challenge addressed in this work is the visualization of historical data. Up to now, the researchers working in this area still utilize 2D-maps in combination with data sheets that contain various information about particular points on the map. These requisites, however, make it extremely difficult to analyze events in time and to answer questions like: "Why are some decisions made in the way they were made by the people participating in the studied wars, migrations, or battles". The question here was: How can we employ computer graphics in order to facilitate the visualization and interactive exploration of historical data?

The main difficulty concerning the visualization of 4D data we are dealing with, is the appropriate display and the interaction with it. Many people are meanwhile familiar with the interaction with 3D data. The display of 3D data including a time dimension, however, is not straightforward and can be carried out in various different ways. Another question arising at this point is how could we display as much data as possible, in order to allow the user to concentrate on the visualization and not on the data sheets. On the other hand, the display of too many data parameters may lead to cognitive overload and even worsen the visualization.

The goal of this work was mainly geared towards the generation of powerful visualization tools that facilitate the interactive exploration of historical data. We propose new interaction concepts for the vivid exploration of such data in a virtual environment improving upon the traditional 2D visualization combined with study of the accompanying data sheets. Moreover, the presented tools are not limited to the application to historical data. They can be applied to various other visualizations of 3D data containing a time dimension as well. Finally, the work in this area consisted of two main components: the development of a portable traditional desktop application and of a versatile virtual reality application. The latter aims to exploit the various interaction features of a VR-system and to offer vivid gain of insight into the data.

1.2.2 Interaction in Virtual Environments

Additionally to the employment of VR-techniques for the visualization of historical data, we also address the interaction in VEs in general. Although, much effort is made towards providing suitable navigation in virtual reality applications, many of the resulting tools still cause loss of orientation and inadequate navigation. Furthermore, considering the remote object manipulation, none of the published approaches is suitable for manipulation *at* the remote location, while enabling interaction with the object of interest as if it is within the reach of the user's hands. Lastly, none of the various special purpose interaction techniques and tools published in the literature addresses the interactive exploration of 3D data containing a time dimension. These facts introduce the next goal of this work: to investigate and develop new navigation and interaction techniques making use of available concepts, while overcoming the drawbacks of the existing tools.

The result of our research in this direction is a new paradigm for the simultaneous exploration of two superimposed virtual worlds: a *primary* world surrounding the user and a *secondary* world explored with a lens-like tool. Due to the applied lens, connecting both virtual worlds with each other, we call this idea the *through-the-lens* paradigm (as shown in Section 4.2, p. 51). Based on this concept, we propose a set of techniques for convenient navigation, exploration, visualization, and remote object manipulation in VR-applications.

Considering the navigation in virtual environments, we show how the *through-the-lens* concept was applied in order to circumvent the typical limitations of traditional navigation aids like orientation loss and fatigue, while still supporting powerful, intuitive, and precise work in virtual environments. We discuss three different navigation concepts for interactive traveling within the secondary world. Each of them is applied for navigation with a particular target: from coarse level exploration to high precision viewpoint positioning.

The second technique based on the through-the-lens concept, the remote object manipulation, enables the accurate interaction with distant objects in their natural environment. This is a feature that is not supported by any of the known remote interaction techniques, which, however, greatly improves the completion of the aimed task. Even though this is a typical technique that does not have a counterpart in the real world, it proved to be extremely useful, even indispensable in virtual environments.

Finally, with the through-the-lens tools for exploration of historical data we have shown how this concept can be successfully combined with the navigation through the time dimension. The proposed system for visualization of historical data offers a valuable companion for the analysis and interactive study of historical events. Summarizing, we can state that the through-the-lens tools significantly enhance the interaction and enable a versatile visualization of the explored data.

1.2.3 Rendering Algorithms

Although the computer graphics hardware is being rapidly improving and is capable to render even very large scenes, the amount of data is growing even faster and this is still a potential bottleneck of a graphical application system. This in turn may affect and negatively influence the interactivity of a virtual environment. Especially when displaying a scene in stereo mode, the data has to be rendered twice in order to generate one image for each eye, even though there is a lot of redundant information in both images. On the other hand, the through-the-lens tools addressed above, may also cause a deterioration of the frame rate, since the complete scene behind the through-the-lens window passes through the rendering pipeline additionally to the primary synthetic world surrounding the user. The goal of this work in this context was to detect potential bottlenecks and avoid them exploiting the characteristics of these two problems.

Considering the stereo rendering, we propose a method for partitioning the scene in two parts: (a) the near foreground, which has to be rendered separately for each eye and (b) the far background, which appears to be nearly the same for both eyes. In order to prove, that this scenario does not affect the stereo-based depth perception and to compare the traditional stereo rendering with the proposed mixed (stereo/mono) rendering, we performed a set of usability studies. The latter have shown that this approach is applicable and that the human depth perception can be fooled without causing distortion in the stereovision.

In addition, in order to speed up the rendering of the through-the-lens tools, we propose an image-based approach for displaying the world behind the through-the-lens window. Thereby, we exploit the fact that only small part of the secondary scene (behind the through-the-lens window) can be viewed through a window with limited size. Keeping in mind these restrictions, we have developed an image-based rendering technique, which enables fast rendering and adaptive acquisition of the required data, while still supporting the features of the geometry-based rendering.

To summarize, throughout this work, we address typical problems in each of the components marked in Figure 1.1. For each of them, we analyze the reasons why the problems appear and point out possible ways for overcoming the limitations of existing algorithms. We also present new approaches in order to solve the addressed problems.

1.3 Summary of This Work

The remainder of this work is organized as follows. In Chapter 2, we describe the soft- and hardware, on which the developed tools and techniques are based. We present in detail the fundamental interaction tools based on the available hardware, as well as the underlying software system. We also shortly point out the main differences between a traditional desktop computer and a virtual reality application with the associated hardware environment. Finally, we address some of the problems with the input devices in a virtual reality setup.

Chapter 3 (p. 19) consists of two main parts that are independent of the virtual environment. These two parts elaborate on the visualization and the data preprocessing. In the first, we present the new watershed transformation for extraction of objects of interest out of image and volume data. We give an outline of the proposed algorithm, as well as results of its application in terms of the required time and the achieved segmentation. In the second part of this chapter, we address the preprocessing, digitalization, and display of historical data. Furthermore, we also discuss some additional props for orientation in, and navigation through the space and the time dimensions of the underlying data.

In Chapter 4 (p. 47), we present the through-the-lens concept. First, we introduce a taxonomy for the states of the two aligned virtual worlds and the way they are visualized in each other. Thereafter, we present the set of techniques based on this metaphor, which can be applied for navigation, interaction, remote object manipulation, and visualization of historical data in a virtual environment. For each of these tools, we discuss its implementation and application.

Finally, in Chapter 5 (p. 83), we give a detailed overview of two approaches for accelerating the rendering in virtual environments. Each of them exploits different features of (a) the stereo rendering in general and (b) of the through-the-lens tools described in Chapter 4. For each algorithm, we compare the performance on at least two different datasets and show the improvements achieved with the described methods. We conclude this work with Chapter 6 (p. 111), which gives a brief summary and some concluding remarks.

Chapter 2

System Setup

In this chapter, we describe in detail our tabletop-projection output device, as well as the tracker-based input device. First, we discuss related work on input and output devices including stereo projection techniques and two-handed interaction. Then, we present the hardware used to realize the interaction and display in our system. Finally, we also address calibration issues and discuss the software environment, introducing design requirements to the developed system. Even though our implementation exploits the hardware and software presented in this chapter, all techniques described later on can be easily ported and integrated into other scenarios and setups.

2.1 Output Devices

Like any traditional desktop computer, a virtual reality system consists in general of an input device, an output device, and a "main computer". The output device defines the type of the VR system and predetermines its application area. Considering the output, a VR-system may be *immersive* or *augmented* (also known as "see-through"). In the first scenario, the participant views a completely virtual world. She/he is essentially isolated from the outside world and fully enveloped within the computer-generated environment. The typical devices applied in this case are multiple wall-projections (so called CAVE-like systems [Cruz-Neira et al., 1993]), single wall projections, one or two sided virtual workbenches [Agrawala et al., 1997], tabletop displays, as well as head mounted displays (HMD) [Sutherland, 1968, Feiner et al., 1993]. The second group, the see-through systems, is typically based on semi-transparent HMDs, allowing to view the surrounding environment and to simultaneously *superimpose* information via semi-transparent HMD [Bajura et al., 1992, Pausch, 1991]. Depending on the target application area one of these types has to be chosen. In both cases, different input devices and thus interaction concepts are utilized.

Although CAVE-like devices are impressive and provide remarkable immersion into a virtual environment, the evolution trend of VR technology is clearly oriented towards smaller, cheaper, and more flexible systems. The latter offer great resolution bringing 3D immersive technology into the workplace at a fraction of the cost of customary VR technology. One direction of this technology development makes up table-like devices (virtual tables). Among the first tabletop devices, the most important is the virtual workbench presented in 1997 at Siggraph [Agrawala et al., 1997]. Since then, many similarly designed projection setups have been developed. Such systems are especially designed for applications that are traditionally performed with or on workbenches, tables, or presentation surfaces. Through the scene presentation (synthetic or real-world scenes) on a horizontal or vertically-tilted output surface, the usual dialog concept for man-machine communication is put into an application-oriented form that is tailored to the particular working situation. This facilitates the intuitive work within this "common", though virtual environment.

Throughout this work, we address the immersive scenario. Even though all techniques and tools we present in the subsequent chapters are developed on a virtual table output device (described in Section 2.3), all of them are portable and applicable in any other immersive scenario.

2.2 Input Devices

There are various ways for realizing input devices in a VR-system. The extremely diverse employment configurations and scenarios require the development of appropriate technologies for interaction with them. Since an immersive virtual reality setup aims to *surround* the participants, who can move freely in the virtual environment, traditional input devices like keyboard and mouse are unsuitable for interaction in this scenario. The most common way to integrate input is to use a tracker, which gives information about the position and orientation of tracked points of interest. Such points of interest are typically the viewer's head and hands.

Conceptually, there are five main tracking approaches: electro-magnetic, mechanical, optical, acoustic (ultrasonic), and gyroscopic tracking, which differ mainly in the accuracy, resolution, tracking speed, latency, working volume, and liability. The most frequently used approach is the electro-magnetic tracking, since such tracker systems are robust, occlusion-independent, precise, and readily extendable. Unfortunately, the price we have to pay for these features is the sensibility to noise and the cables still used to connect the sensors to the tracker. The first problem can be circumvented in many different ways. In Section 2.3.3, we elaborate on the way this *undistortion step* is performed in our system. The second problem, the wired sensors, is not addressed in this work, since we do not elaborate on any hardware issues.

2.3 Hardware Setup

The hardware used in our setup consists of an electro-magnetic 6DOF tracking system (*Ascension, Flock of Birds*) used as an input device and a tabletop display (*Barco, Baron* or *Virtual Table*) as an output device. The screen has an approximate size of $140 \times 105cm$. It is tilted about its x-axis, on which the images are back-projected in stereo mode. The virtual table can be arbitrary rotated about the tilt axis as shown in the Figure 2.1. The rotation angle varies between $0°$ and approximately $80°$ from the horizontal position. Thus, it enables the realization of traditional workbenches, as well as (almost vertical) power-walls. The latest version of the table allows even a $90°$ tilt and provides an improved display on a high-resolution screen supporting a wide viewing angle with built-in optical correction and distortion reduction. These outstanding features, combined with a modular design and the compactness of the unit make it the product of choice for our 3D virtual reality setup.

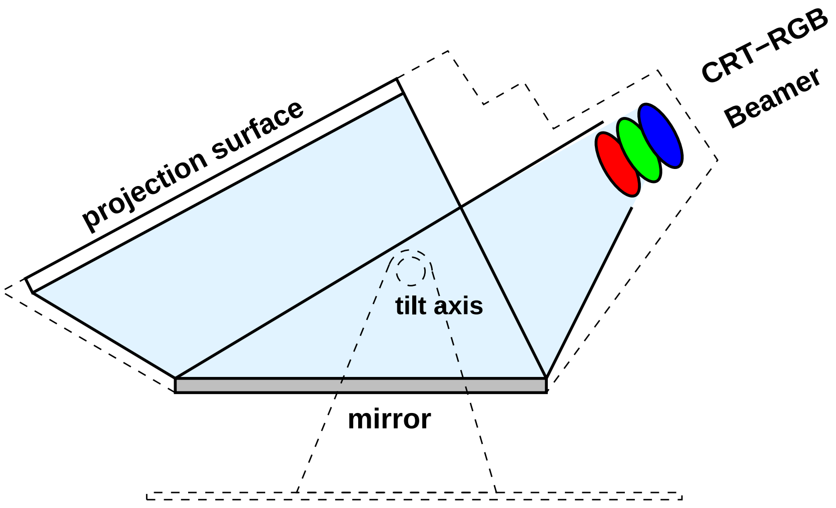


Figure 2.1: The projection technology of the virtual table with projection surface tilted $30°$ about the x-axis.

2.3.1 Stereoscopic Display

One of the main features distinguishing the common desktop output from the output of a virtual reality application is the *stereo* display of the virtual world[1]. As pointed out by Rosenberg [Rosenberg, 1993], the use of stereo projections over monocular projections is strongly recommended in order to improve performance in depth perception tasks. Intensive research has been carried out in the area of stereo projection and depth perception in the past years [Yeh and Silverstein, 1990, Wann et al., 1995, Yeh and Silverstein, 1992]. It has been shown, that the most important cues for a subject's depth judgment are the dynamic flow (which is achieved through head-tracked displays), the perspective, the stereo display, and the scene lights and objects' shadows. Although much depth information can be inferred from the introduced monocular depth cues alone, the stereoscopic depth cues provide additional information. It enhances the speed and accuracy of tasks requiring depth perception [Rosenberg, 1993], as this is typically the case in immersive virtual environments.

There are various ways to realize stereo projection [Hodges, 1992, Lipton, 1997]. In our scenario, we have chosen the alternating projection of pictures generated for the left and right eyes combined with an LCD shutter. In other words, each frame shown to the participant is generated with two slightly shifted and among the point of focus rotated virtual cameras. Afterwards, the LCD shutter switches the appropriate shutter on and off depending on which picture (for the left or the right eye) is being currently displayed. This process is typically performed approximately (or more than) 60 times per second, creating the illusion of perceiving one stereoscopic image.

2.3.2 Tracker, Pad, and Pen

As introduced above, another important depth perception aid is the dynamic flow achieved through head tracking. We applied an electro-magnetic tracker for carrying

[1]A detailed discussion on stereo vision and implementation of stereo displays is given in Section 5.3.2, p. 96.

out the interaction in our system. The data captured by the tracker is processed by a *tracker-server* [Reitmayr and Schmalstieg, 2001], which is responsible for the transformation of the received sensor position and orientation into the world coordinate system. The tracking unit is attached to and communicates directly with the tracker-server, which creates small data packets and sends them via multicast over the local area network (LAN), as shown in Figure 2.2. Each packet contains a time stamp, the

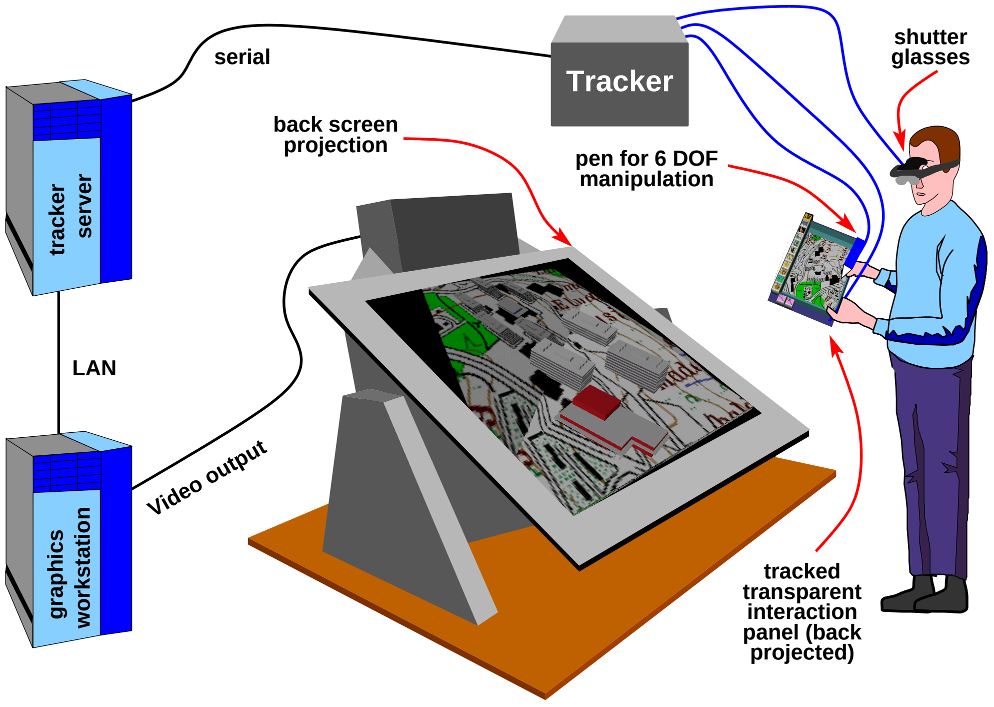


Figure 2.2: Hardware setup of the virtual environment. The images on the hand-held panel are back-projected via the table display and are aligned with the (tracked) interaction pad.

ID of the tracked station, its position, and orientation. Our tracker system tracks three stations with (at least) 50Hz frequency. Measurements showed that no tracker events get lost when sent trough a 100 Mbit LAN. Hence, each tracker client receives these packets with the same frequency.

The three receivers, tracked by our Flock-of-Birds unit, provide information about their position and orientation in the space. The first is used to track the viewer's head (see Figure 2.2). The virtual camera is attached to this receiver. The second and the third receivers are attached to the devices held in both hands in order to enable two-handed interaction. The first contributions to two-handed interaction we know of are the work of Buxton and Myers [Buxton and Myers, 1986] and Guiard [Guiard, 1988]. Buxton and Myers reported on significant performance increase when bimanual navigation/selection is applied compared to accomplishing the task unimanually. Guiard's work showed how a prop in the non-dominant hand is used to define a (coarse oriented) coordinate system, a kind of a reference frame, while the dominant hand is used for fine positioning relative to that coordinate system. Kabbash [Kabbash et al., 1994] evaluated two-handed interaction techniques and compared them with unimanual interaction. The conclusion of his work is that bimanual interaction can improve overall performance, especially when asymmetric partition of labor is possible. Further work in this area, as well as systems applying bimanual interaction are described in [Brooks, Jr., 1988, Turner et al., 1996, Cutler et al., 1997].

Exploiting the above observations on the asymmetric use and coordination of human hands, several research groups developed two-handed interaction techniques for different virtual environments. The resulting tools are known under various names like:

- *pen and palette* [Sachs et al., 1991],
- *pen and tablet* [Angus and Sowizral, 1995],
- *physical clipboard* [Stoakley et al., 1995],
- *3D-Palette* [Billinghurst et al., 1997], the
- *personal interaction panel (Pip)* [Szalavári and Gervautz, 1997],
- *virtual palette and remote control panel* [Coquillart and Wesche, 1999].

The idea is quite simple: the user is provided with a tracked pad used as a frame of reference for the interaction with computer generated widgets. In addition, a tool for manipulating these interface elements is provided, e.g. a virtual pen. The virtual tool is a visual duplicate of a six-degrees-of-freedom input device, providing tactile feedback, similar to the Virtual Tricorder introduced by Wloka and Greenfield [Wloka and Greenfield, 1995]. This two-handed interaction concept turned out to be very intuitive to use and suitable for utilization in various kinds of virtual environments. Therefore, we based the interaction in our system on the *personal interaction panel* [Szalavári and Gervautz, 1997].

In this scenario, one receiver is attached to physical pen, which the user holds in his/her dominant hand. The virtual counterpart of the pen is used to manipulate the virtual 3D buttons, sliders, and other interaction elements projected on the interaction pad. The pad is tracked with the third receiver and is a transparent panel, on which the interaction elements are back-projected [Schmalstieg et al., 1999]. Since we always know the exact position and orientation of the pad, the virtual interaction elements are displayed in such a way on the table, that the user sees them on the pad's surface (through the transparent material). Furthermore, the pad can be used as a tool itself. It allows to superimpose information through it like this is done with the magic lens [Bier et al., 1993]. In addition, it can also display images on its surface as seen from a different viewport, through which the scene is projected. In this case, not only the viewport (with the pad on which surface it is mapped), but also the scene seen through it can be freely positioned in the space, applying the proposed *through-the-lens* technique (as will be discussed later on).

The basic aim of the pad is to provide passive tactile feedback when touched with the pen. Even though the physical pen cannot collide with the virtual tools mapped on the pad, the surface of the pad defines a physical reference frame facilitating the interaction.

The pen is a transparent device, with two buttons. It is held in the dominant hand and used as a real pen, as opposed to using a wand. The two buttons on the pen can be thought of as mouse buttons, which have a state (pressed or not) and generate events on state changes. Additionally, a sensor responsible for tracking the pen is attached to its tail. This allows realizing intuitive and powerful *two-handed interaction*, which has proven to be very valuable concept in modern VR-applications. On this interaction concept we have based all the tools and techniques described in the remainder of this work. Whenever we talk about two-handed interaction, we refer to the concept presented above.

2.3.3 Calibration

Finally, to complete the hardware description, we have to address the distortion problem. In every tracker system, there are various distortion sources. Since we have used an electro-magnetic tracker in our setup, we discuss only the electro-magnetic distortion sources. Although the Ascension (Flock of Birds) tracker is DC pulsed, as opposed to the Polhemus devices which are AC pulsed, metal (conduction and ferrous) objects and interleaved electro-magnetic fields magnify the distortion. Due to the DC nature of the tracker, the sensitivity to metal is claimed to be in the order 5 to 10 times less than AC tracker systems. However, some tracker distortion sources still have significant impact on the tracking quality.

When we consider distortions, we distinguish between static and dynamic distortions. Static distortions are fixed with respect to their continuity and their position. There are two ways to compensate the error caused by static sources: using passive shielding and applying active correction (or filtering). Dynamic distortion sources are very difficult to handle, due to their changing position, size, and interleaved electro-magnetic fields. Fortunately, dynamic distortion sources are small and thus have a rather little impact on the tracking. Therefore, in the following we concentrate on the static distortion.

The main distortion sources in our setup were a large current transformer one level bellow our VR setup, and the beamer used to generate the output images (see Figure 2.1). Here only the position distortion is addressed, since it turned out that

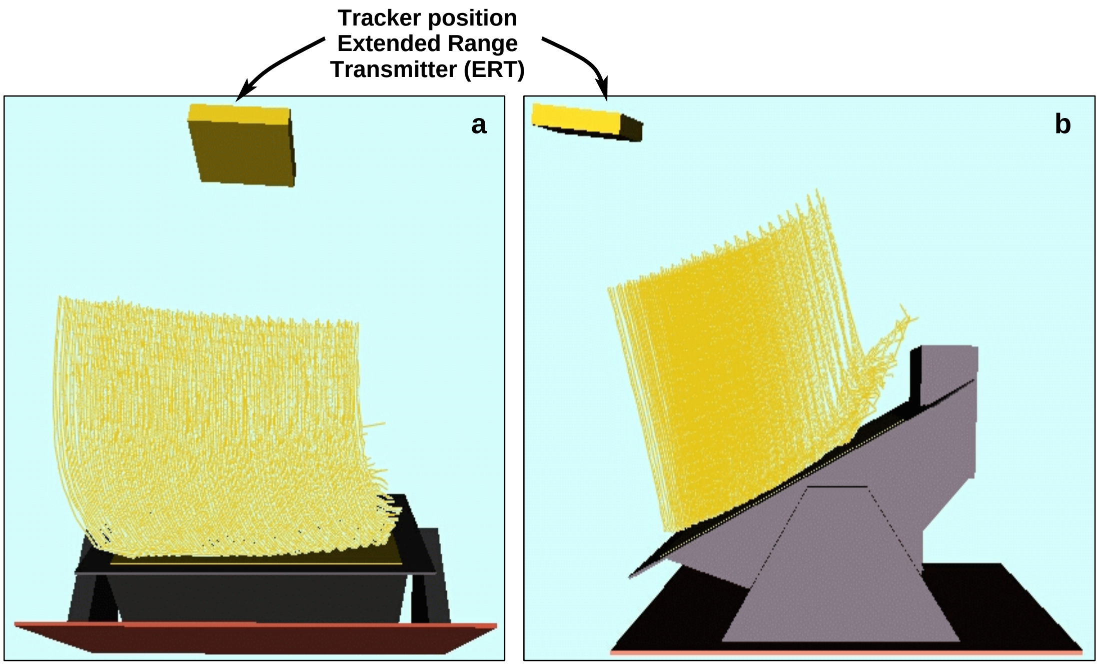


Figure 2.3: The space above the Virtual Table's surface was sampled on a regular $5 \times 5 \times 5cm$ grid. The points on the grid, however, were received by the system as shown in these two images. Image (a) shows the curved vertical lines caused by the transformer field. Image (b) shows the distortion near the beamer.

the orientation is not crucially affected by the distortion sources. In order to measure and correct the distortion of the position, we have sampled the space above the surface of the Virtual Table $(140 \times 105 \times 120cm)$ on a regular $5 \times 5 \times 5cm$ grid. For each of these 15950 points we have recorded the values measured by the tracker.

Figure 2.3 shows these measured values. The box above the virtual table indicated the position of the Extended Range Transmitter (ERT), which generates the required electro-magnetic field and was therefore mounted using only non-magnetic materials like wood and plastic. We can clearly see in Figure 2.3 the two distortion sources: (a) the transformer causing distortion in form of curved vertical lines and (b) the beamer mounted on the upper side of the table.

In order to compensate the distortion sources, we have to compute between which sampled and measured points lies the currently measured position and apply an interpolation scheme in order to determine its final position. For this, all sampled points corresponding to measured points within a sphere with particular radius are taken into account and weighted with the distance to the measured points.

Unfortunately, since this step is performed for each frame, this computation soon becomes the bottleneck of the tracking. Even though the tracker packets are received with 50 Hz frequency, not all of them can be processed. In practice, we have achieved processing frequencies of 30 Hz, which means that approximately each second tracker event is taken into account. This is sufficient for reasonable interaction with scenes with limited complexity. However, for large scenes that can be rendered on the same machine at reasonable frame rates (without tracking), the undistortion step can cause significant deterioration of the overall performance when tracking is activated. This is due to the fact, that the rendering time has to be added to the computation time for the undistortion. For instance, if the geometry can be rendered 20 times per second (20 Hz), the overall time for rendering one frame is $33ms$ for the undistortion and $50ms$ for the rendering, which results in $83ms$ per frame or 12 Hz. In Chapter 5, we discuss in detail two techniques for accelerating the rendering speed.

2.4 Software Environment

In this section, we do not address software design issues. Instead, we rather introduce the development environment in order to allow detailed description of the techniques discussed in the following chapters. The first basic criterion for the development was the object-oriented design. Object-oriented programming is generally focused on the development of classes as basic units. These units are derived from the types of data relevant to the implementation. While each class describes a certain general category, an instance of a class is a data structure representing one specific representative of this type. Through inheritance, object-oriented software development provides a mean to organize related classes in order to take advantage of their commonalities and to keep the class structure understandable and manageable. Furthermore, through polymorphism and dynamic binding, it allows building structures made of objects of different, yet related types, and to ensure that every operation will automatically adapt to the type of its target object. In this way, object-oriented programming provides a set of powerful concepts to address some of the most pressing issues of software development. Its most promising contributions refer to the aspects reliability, efficiency, reusability, portability, and extendibility of software quality [Meyer, 1995]. These observations strongly favor the use of the object-oriented, standardized, and widely accepted programming language C++ [Stroustrup, 1997], available on all Unix and Windows platforms.

In addition to the decision to extensively follow standards in order to support portability, we had to choose an object oriented graphics library. Since the graphics output is still a crucial and time critical task, another decision criterion was the rendering performance. Due to these conditions, the library of our choice was Open Inventor [Wernecke, 1999], which is object oriented, portable, and readily extendable library, built on top of OpenGL [Woo et al., 1998]. OpenGL is a software interface to graphics hardware supporting various hardware features and offering great performance. Although it is available on various operating systems, its design is kept simple and for performance reasons not object oriented. Open Inventor provides the bridge between object oriented design and fast, while portable graphics interface, which justified our choice.

Since Open Inventor is a graphics library, in its native form it only limitedly supports interaction and interface elements. However, it offers all mechanisms required for implementing interface aids. All these features are bundled in the Studierstube framework [Schmalstieg et al., 1996], which is an object-oriented library extending the standard Open Inventor functionality, allowing for transparent processing of 3D events. The latter are propagated through the Open Inventor scene-graph and can be used to define 3D interface elements like buttons, sliders, etc. and to realize various interaction concepts with them. Figure 2.4 shows the layers between the (VR) applications and the hardware discussed in the previous section. The manager classes

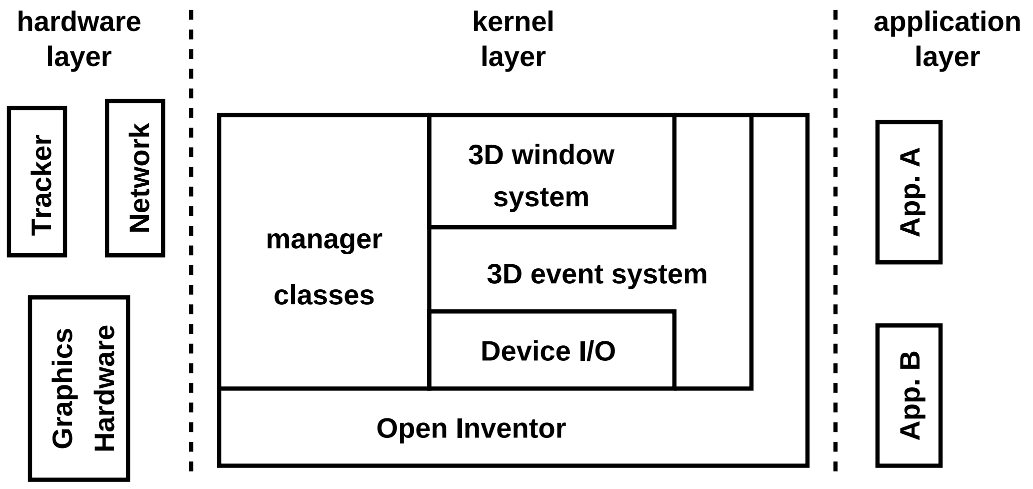


Figure 2.4: The layers of the software architecture.

are responsible for receiving the 3D events from the input devices, which are sent through the LAN. The windowing system, the event system, and the Device I/O are not directly visible for the application programmer, who rather utilizes traditional Open Inventor classes as well as the additional Studierstube classes using one common interface: Open Inventor. The Studierstube library also supports concepts like the window-concept, in which each application can run in a designated window. This is similar to the traditional 2D window concept. Unlike the 2D window, however, a window is in this case a three-dimensional area.

Moreover, the Studierstube provides many additional useful classes. For instance, it implements the *SEAM*-paradigm [Schmalstieg and Schaufler, 1999]. A *SEAM* is defined by Schmalstieg and Schaufler as a "door into another world" and stands for a *Spatially Extended Anchor Mechanism.* It makes it possible to connect virtual worlds using a kind of a wormhole. The participant can view or enter a world (called *secondary* world) seen through a window fixed in the space of the world surrounding him/her (called *primary* world). The Studierstube implements this and other concepts within its object-oriented framework. We discuss further architectural and implementation details in Chapter 4.

To summarize, the Studierstube library is used in our system as a layer between the user interface and the application. With it, we are able to readily create, (re-) configure, and maintain new and existing virtual reality applications. Through its independence from the underlying hardware, it provides a convenient and flexible library for implementing immersive or augmented, collaborative, and distributed applications.

Chapter 3

Visualization Approaches

3.1 Introduction

The visualization of data is a problem requiring preprocessing algorithms and display techniques tailored to the particular input data. With this in mind, the visualization task can be considered as covering following subtasks: data acquisition, data preprocessing, and data display, followed by interactive data exploration (see Figure 1.1, p. 4).

The first step, the original data acquisition, is not an issue in this work. We assume that the data exist in some form. The main emphasis now is the data preprocessing and display. To the preprocessing count the application of various filters, simplification algorithms, even transformation of the data from one type to another (e.g. 2D to 3D, nD to 3D and 4D). After completing this step, the display of the data has to be considered. This in turn includes visualization (e.g. streamlines), rendering (e.g. image-based/polygon representation), and display algorithms (2D, 3D, stereo, etc.).

In this chapter, we discuss in detail the preprocessing steps performed with two different data types towards assembling a VR-application. First, we present a segmentation approach applied for extracting objects of interest out of 2D image and 3D volume data. Since the interactive direct volume visualization is not possible on many machines nowadays, surface models have to be generated out of the input data. We propose a method for semi-automatic extraction of data structures based on a local watershed transformation [Stoev and Straßer, 2000]. The result of this approach is a surface model, which can be readily displayed with customer hardware. Section 3.2 is completely devoted to this problem and provides a discussion on each of the performed steps. Once the approach is applied and the data extracted, the resulting surface model can be used for displaying three-dimensional geometry in a conventional manner or in a VR-application.

The emphasis of the second part of this chapter (Section 3.3, p. 38) is on the preprocessing and visualization of historical data [Stoev and Straßer, 2001]. We present a pipeline reaching from the raw historical delineations to the display of a triangulated model. Furthermore, we discuss techniques that independently of the interaction with the data improve the data display and help to understand the visualized processes. This model is embedded in a VR-application for historical data exploration. Due to the four-dimensional (3D space and a time dimension) nature of the data, in Section 4.5 (p. 71) we also discuss in detail the interaction techniques used for data exploration.

3.2 Segmentation with the Watershed Transformation

In this section, we present a new technique for extracting regions of interest (ROI) applying a local watershed transformation. The proposed strategy for computing catchment basins in a given region of interest is based on a rain-falling simulation. Unlike the standard watershed algorithms, which flood the complete (gradient magnitude of an) image, the proposed approach allows us to perform this task locally. Thus, a controlled region growth is performed, saving time and reducing the memory requirement especially when applied to large image or volume data.

A second problem arising from the standard watershed transformation is the over-segmented result and the lack of sound criteria for merging the computed basins. For overcoming this drawback, we present a basin-merging strategy introducing four criteria for merging adjacent basins. The threshold values applied in this strategy are derived from the user input and match the attributes of the selected object rather than the attributes of all objects in the image. In doing so, the user is not required to adjust abstract numbers, but to simply select a coarse region of interest. A drawback of this strategy, however, is that the result of the segmentation often cannot be exactly reproduced.

The proposed algorithm is not limited to the 2D case. As we shall show, it is suitable for processing volume data as well. Concluding this section, we present the results of applying the proposed approach on several example images and volume datasets.

3.2.1 Introduction and Related Work

The extraction of meaningful regions from image and volume data continues to be an important and unsolved topic in the image analysis and image processing area. In the past years, the watershed transformation [Meyer, 1994, Najman and Schmitt, 1994] has proven to be a very useful and powerful tool for morphological image segmentation. Since its introduction in [Band, 1986], it is becoming increasingly popular in different science areas like biomedical, medical image processing [Higgins and Ojard, 1993, Sijbers et al., 1997], computer vision and segmentation [Wegner et al., 1995], even mesh segmentation [Mangan and Whitaker, 1999], etc.

The idea of the watershed transformation is quite simple. A (gradient magnitude of a) gray-scale image or volume is considered as a topographic relief [Vincent and Soille, 1991]. Each pixel in this digital image is assigned during the transformation to the catchment basin of a regional minimum. The *catchment basin* of a regional minimum is defined as the area, in which the pixels hit by a raindrop will cause the raindrop to flow to the regional minimum. In this way, influence zones for each of the (eventually pre-determined [Meyer, 1994]) regional minima are defined. The watershed lines are now the lines separating influence zones from each other.

Beucher and Lantuéjoul [Beucher and Lantuéjoul, 1979] were the first who proposed an immersion based watershed algorithm. In [Beucher and Meyer, 1993] and [Meyer, 1994] couple of techniques and algorithms related to the problem of watershed computing are described. In his work, Meyer [Meyer, 1994] defines the watershed transformation in the continuous and in the digital space in terms of a distance function, called topographic distance. One of the classical algorithms for computing the watershed transformation for a gray-scale image is also found in this work. The author predetermines the regional minima (single pixels or plateaus) and starts the flooding process at these minima [Meyer, 1994].

Another approach for computing the catchment basins is described in [Vincent and Soille, 1991]. The authors simulate flooding with water, coming up out of the ground and defining the catchment basins without predetermining the regional minima. This approach processes all image pixels in a sorted order, such that pixels with lower altitude are processed first. The preprocessing step here consists of sorting all (pointers to) pixels in an array. Utilizing a First-In-First-Out (FIFO) structure, the pixels at altitude *h+1* are processed after those at altitude *h*. This divides the problem into *m* sub-problems, where *m* is the number of all present pixel altitudes. Due to the processing of pixels at altitude *h* per iteration, the problem is reduced to computing the geodesic *skeleton of influence zones* (SKIZ). After sorting the pixels depending on their altitude, in order to guarantee fast access to pixels at given *h*, the SKIZ for each *h* is computed. Hence, the plateaus at the current altitude are flooded. Whenever two floods originating from different catchment basins reach each other, a dam is built to prevent the basins from merging. The presented approach is applied in [Vincent and Soille, 1991] to several data structures, including graphs and grids with arbitrary connectivity.

Moga et al. [Moga et al., 1995] report another approach for computing the watershed transformation based on rain-falling simulation within a gray-scale image. Their work describes a parallel algorithm, which first transforms the original image into a *lower complete* image I_l. In I_l the pixels belonging to a non-minimum plateau are labeled with the geodesic distance to the plateau's nearest outdoor. In doing so, a second ordering relation for the pixels in a non-minimum plateau is introduced in the resulting image. Afterwards a raindrop starts at each pixel and its path toward the line with the steepest descent (due to gravity) is followed until a regional minimum is reached. The set of all pixels attracted on the way to a particular regional minimum defines the catchment basin for this minimum. The process of following a raindrop's motion is sequentially performed for all pixels, which results in a set of catchment basins. Adjacent catchment basins are separated by watershed lines. Thus, raindrops falling on both sides of a watershed line flow into different catchment basins.

In order to achieve a meaningful segmentation, the watershed transformation is in general performed on gradient images. The gradient transformed images are computed in a preprocessed step. In these images, the gradient maximums are more often than not watershed lines surrounding homogenous regions.

One of the main disadvantages of the watershed transformation as described in the literature is that it is a global transformation. This means that it requires the processing of the entire input data. However, the goal of the segmentation is often the extraction of only a single region of interest (ROI) out of a given image or volume data. Especially when volume data is considered, the process of computing the watershed transformation is very time and memory consuming.

On the other hand, the extraction of a sole catchment basin does not provide sufficient results. Since the global watershed transformation produces in general heavily over-segmented results, a sole catchment basin is often meaningless for the segmentation of a region of interest. Nevertheless, the correct contours are most of the time present in the watershed-transformed image. Thus, an additional task after applying the global watershed transformation has to be performed: the computed catchment basins have to be merged appropriately. Various approaches for accomplishing this task are described in the literature and are based on gradient-watersheds on graphs [Vincent and Soille, 1991], basin dynamics [Najman and Schmitt, 1996], markers [Meyer and Beucher, 1990], inclusionary and exclusionary cues [Higgins and Ojard, 1993], image component labeling [Moga and Gabbouj, 1997], and multi scale gradient analysis [Jackway, 1995]. In practice, however, it is very difficult to define sound criteria for all objects in the image. Therefore, the definition of *local* merging criteria, matching the attributes of a given ROI, yields a sound solution of this problem.

Here we present a new approach for *locally* applying the watershed transformation. We first describe how a single basin can be extracted without computing the watershed-transformed for the entire input data and without preprocessing the input data. Afterwards, this method is generalized to the case, where a user makes a coarse region selection, which is flooded and the basins within the selected ROI are merged. These selected basins are now utilized to derive threshold values for a set of merging criteria, which is discussed in Section 3.2.4 (p. 28). Thus, a sort of controlled basin-level region growth is performed based on both the regular input data and its gradient magnitude transformed. This makes it possible to precisely define the merging criteria for the object of interest, instead of defining merging criteria for the entire input data.

3.2.2 Notations

For clarity, we introduce the single steps for the 2D case considering pixels located on a regular rectangular grid without loss of generality. Before describing in detail our approach, we make some definitions used throughout the remainder of this section. We define D_f to be the domain of the gray-scale image or volume dataset, where f denotes the image (volume) function. $N_G(p)$ stands for the set of neighbor pixels of a pixel p on the underlying grid G. Furthermore, we define the following terms:

- A pixel $p \in D_f$ is called an *isolated minimum* if $f(p) < f(q),\ \forall q \in N_G(p)$;
- A pixel $p \in D_f$ is defined as being *on a plateau* P with altitude h (or $p \in P_h$), if $\exists q \in N_G(p)$ with $h = f(p) = f(q)$;
- A pixel $p \in D_f$ is called an *outdoor* of a plateau P, if p is on the plateau P and $\exists q \in N_G(p)$ such that $f(p) > f(q)$;
- A pixel $p \in D_f$ is called an *inner pixel* of a plateau P, if $\forall q \in N_G(p), f(q) = f(p)$;
- A pixel $p \in D_f$ is called a *border pixel* ($p \in B(P)$) of a plateau P, if $p \in P$ and p is not an *inner pixel*;
- A plateau P is called a *minimum plateau* (or P_M) in D_f, if $\nexists p \in B(P)$, such that p is an outdoor;
- A plateau P is called a *non-minimum plateau* (or P_N) in D_f, if $\exists p \in P$, such that p is an outdoor.

3.2.3 The Local Watershed Transformation

Unlike the standard watershed algorithms, the aim of the approach described in this section is to provide a strategy for basin computing, which does not require preprocessing or other global information about the data. Unfortunately, all approaches published in the literature require such preprocessing steps, in order to either sort all pixels of the input image [Vincent and Soille, 1991], to pre-compute the local minima from where the basins are flooded [Meyer, 1994], or to introduce a metric for pixels with equal altitude: plateau pre-computing [Moga et al., 1995]. The last approach seemed to us to be the most suitable for modification toward developing a locally working basin extraction technique. However, the authors of [Moga et al., 1995] concentrate on the parallelization and neither the local applicability of the proposed approach, nor the processing of large datasets (i.e. volume data) are discussed. On the other hand, since the plateau computing can be performed only when this is required (i.e. a plateau is reached), we utilize a modified version of this technique in our algorithm. In the preprocessing step applied in [Moga et al., 1995] the entire input data is traversed. This makes the approach time and memory consuming, when applied on large volume datasets. Moreover, the authors of [Moga et al., 1995] do not sort the outdoors of a plateau before it is flooded. Hence, an error is introduced compared with the immersion based watershed algorithms (see below). Finally, a second error source is introduced due to the arbitrary choosing of a pixel in case 4 as described in the next section.

In order to explain the proposed local watershed transformation, we first explain how a basin is extracted given a starting pixel. Afterwards, we describe how the user input is processed based on the local basin extraction. Finally, we introduce a set of merging criteria and depict how they are applied for extracting the target object consisting of many merged basins.

3.2.3.1 The Local Basin Extraction

During the data processing, a raindrop-hit is simulated for a given pixel. Then the raindrop follows the path of the steepest descent due to gravity, until a regional minimum is reached. A *regional minimum* M is a single pixel (isolated minimum) or a set of pixels with equal altitude (minimum plateau), from which it is impossible to reach a point of lower altitude on an always-descending path. When such a minimum is reached, the pixels attracted on the path of steepest descent are marked with the label of the latter.

The reached minimum M is now flooded in the following way. For each of the yet unmarked pixels q adjacent to the minimum M, again the steepest path is followed. In case the minimum reached from q is M as well, q is assigned to the catchment basin of M denoted by C (see Figure 3.1). In this case, the unprocessed pixels q' adjacent to each q are assigned to the set of unprocessed pixels Q. These pixels are considered in the next pixel iteration for the current basin. Otherwise, if another local minimum M' is reached, M' is put on the queue Q_B containing the yet unprocessed adjacent basins/minima (considered in the next basin iteration). The pseudo-code outline of the algorithm for computing the catchment basin for a given pixel follows:

```
/* given a pixel $p$ */
0  follow_steepest_path($p$,$M_i$);      /* stores reached minimum in $M_i$ */
1  $C_i \leftarrow M_i$;      /* initialize $C_i$ with the pixels in $M_i$ */
2  $Q \leftarrow$ border($M_i$);      /* initialize $Q$ with the pixels surrounding $M_i$ */
3  while (not_empty($Q$))
4     $q \leftarrow$ pop_pixel($Q$);
5     follow_steepest_path($q$,$M_j$);
6     if (Id($M_j$) == Id($M_i$))
7        add_pixel($C_i$,$q$);      /* assigns $q$ to the basin $C_i$ */
```

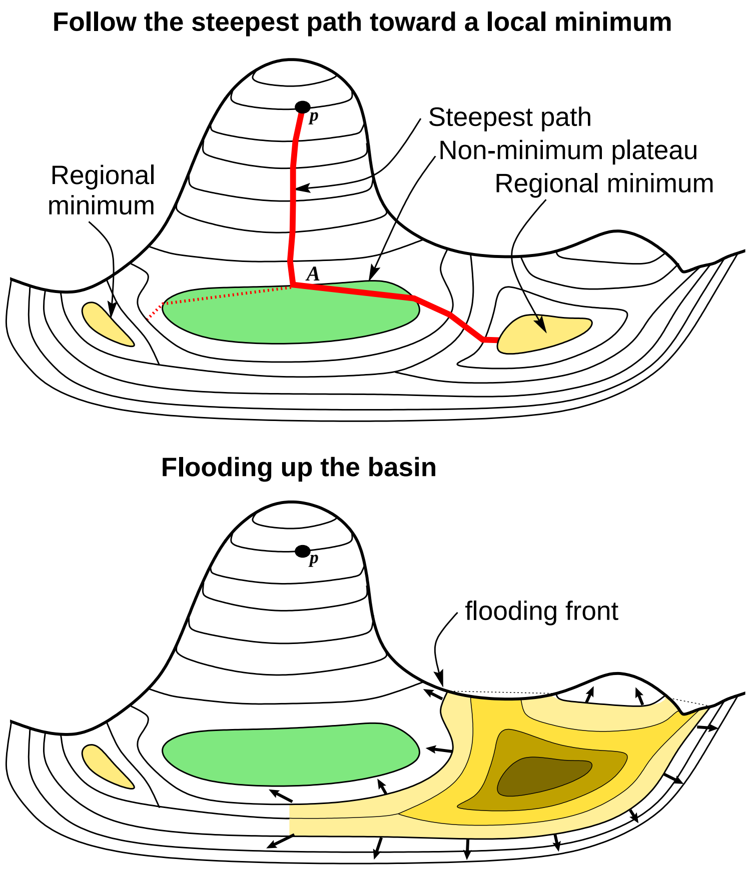


Figure 3.1: A raindrop hits the pixel p. The algorithm follows the steepest path toward a local minimum (upper diagram). Afterwards, the basin is flooded with water coming up out of the reached minimum (lower diagram).

```
8        get_unmarked_neighbors($Q$, $q$);       /* result- stored in $Q$ */
9      else
10       push($M_j$, $Q_B$);        /* the queue $Q_B$ stores the adjacent basins */
11     endif
12  end while;
```

The only complex procedure requiring detailed discussion is the process of following the steepest path toward a local minimum, starting at a given pixel (see line 0 and line 5 above and Figure 3.1). In order to explain the single steps, we assume that we start with pixel p, which has not been processed yet. Thus, four cases can occur (see also Figure 3.2):

1. $\nexists q \in N(p)$ with $f(p) > f(q)$, hence p has no adjacent pixel with lower altitude and is an isolated regional minimum;

2. $\exists! q \in N(p)$ with $f(p) > f(q)$, hence p has only one adjacent pixel with lowest altitude q. This is the regular case, where the algorithm follows the steepest path along the shortest topographic distance;

3. $\nexists q \in N(p)$ with $f(p) > f(q)$, however $\exists q \in N(p)$ with $f(q) = f(p)$, which means that p has at least one adjacent pixel with the same altitude, hence p belongs to a (minimum or non-minimum) plateau;

4. $\exists q_i \in N(p),\ 1 \leq i \leq m$ with $f(q_i) = f(q_{i+1})$ for $i = 1,..,m-1$ and $f(p) > f(q_i), \forall i$. In this case, p has more than one adjacent pixel with lowest altitude q_i. The algorithm cannot determine which of the adjacent pixels is the one the raindrop should flow to.

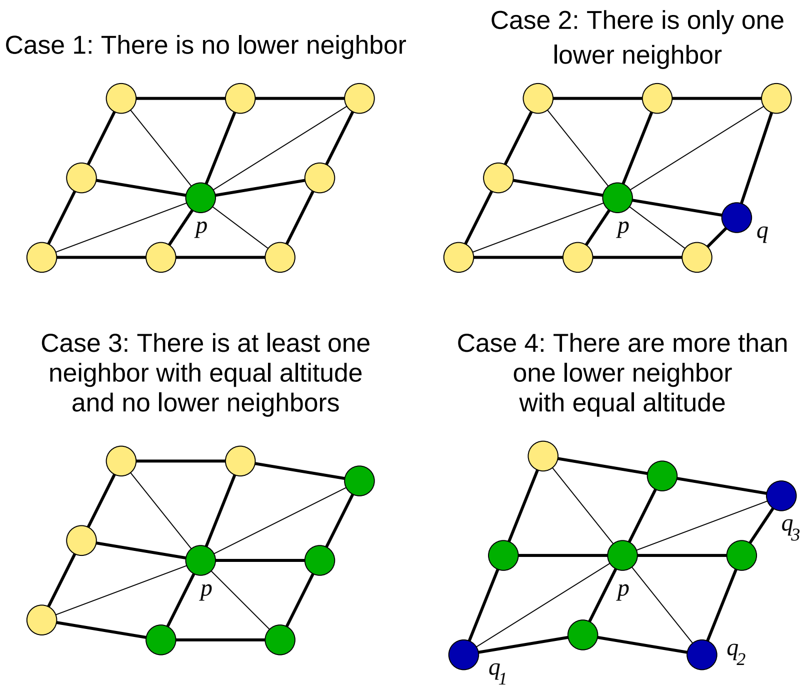

Case 1: There is no lower neighbor
p
Case 2: There is only one lower neighbor
p
q
Case 3: There is at least one neighbor with equal altitude and no lower neighbors
p
Case 4: There are more than one lower neighbor with equal altitude
p
q3
q2
q1

Figure 3.2: The four cases, which can occur when the steepest path toward a local minimum is followed.

The first two cases are the trivial ones (see Figure 3.2). When case 1 occurs, an isolated regional minimum is reached and a new Id is assigned to the basin. In case 2, the current pixel is assigned to the path and if the lowest neighbor q is not marked yet, it is considered as the next processed pixel: $p \leftarrow q$. If q is already marked, the current path is terminated and its pixels are labeled with the label of q. This technique is called *early path termination* and can be formulated as follows: let the sequence $(q_1, \ldots, q_n)$ be a path with q_n belonging to a regional minimum or being an isolated minimum. If $(p_1, \ldots, p_m)$ is the path with already marked pixels, $q_i \in N_G(p_m)$, and $q_i = q$ is the pixel, chosen to be the successor of $p_m = p$, then $(p_1, \ldots, p_m, q_i, \ldots, q_n)$ is a *complete steepest slope path*. Notice, that q_i needs not to be the beginning of a steepest slope path, but can be any arbitrary pixel lying *on* a processed steepest slope path.

In case 3, the reached plateau has to be processed first, since the steepest path cannot be unequivocally determined within plateaus. Therefore, when a plateau P is reached, its border pixels $p_{bi} \in B(P)$ are determined (see Figure 3.3). If there are no border pixels with lower altitude, the plateau is a minimum plateau and the algorithm proceeds like in case 1, namely, all pixels are marked with a unique label. Otherwise, the *outdoors* of the plateau P are sorted and used as starting points for flooding P, as depicted in Figure 3.3.

In order to prevent multiple flooding of the same plateau, the flooding results are saved in a special data structure. In this structure, the pixels are *wired*, storing for each pixel the distance to the nearest outdoor and the direction of the lat-

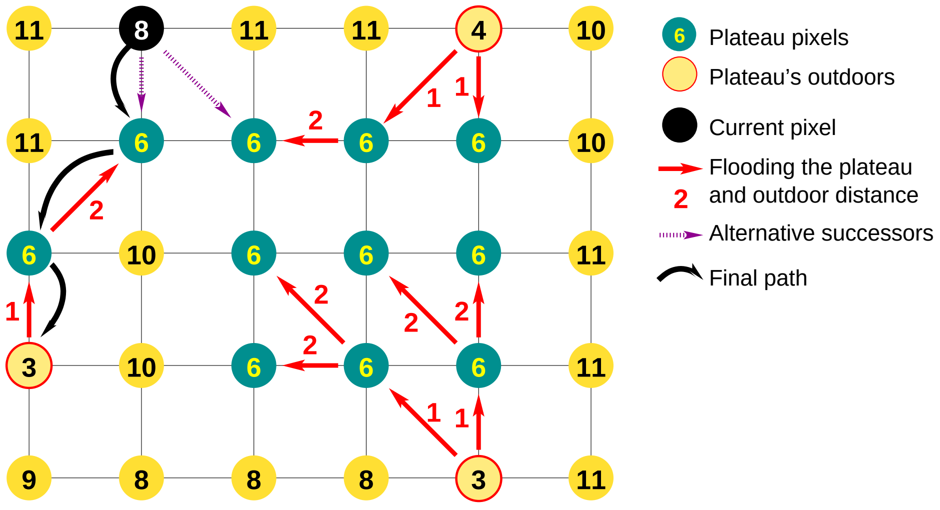


Figure 3.3: When the plateau with altitude 6 is reached, the algorithm first determines its outdoors and then floods the plateau, wiring the pixels within it. The pixels with the shortest distance and if these are equal, the one with the lowest outdoor is chosen to be processed next.

ter (see Figure 3.3). This approach is similar to the *arrowing* technique described in [Vincent and Soille, 1991] and [Meyer, 1994]. Later on, when a pixel of a processed plateau is reached, the arrows to the nearest outdoor are followed and no additional computations are performed. Thus, the nearest and lowest outdoor considering p is now set to be the next pixel in the current path and is processed next. If a marked pixel is reached within the plateau, the current path is terminated and labeled with the Id of this pixel (as in case 2).

When case 4 occurs, the algorithm cannot unequivocally decide which pixel should be processed next. In this case, all adjacent lowest pixels are traversed as if they were hit by a raindrop. Since the lowest pixel $p_n = p$ of the current path $(p_1, \ldots, p_n)$ has a greater altitude than the pixels q_i and a path always follows the steepest slope, none of the pixels $(p_1, \ldots, p_n)$ is affected while q_i are being processed. This allows the algorithm to remain consistent in this case. Hence, after processing all q_i, the pixel q_j with the lowest and nearest outdoor is chosen to be the next processed one $p \leftarrow q_j$ and the computation of the steepest path continues. For pixels q_i not belonging to a plateau, the outdoor-distance is assumed one.

This technique allows us to determine for each pixel the regional minimum it belongs to, without performing a global flooding (typically performed in immersion based watershed transformation) or preprocessing the data. In particular, a raindrop follows the steepest path toward a local (gradient) minimum. The path is then traversed and its pixels are labeled with the Id of the reached minimum. Afterwards, the basin is flooded "piercing" its minimum and letting the water come up out of the ground. This procedure is repeated until all boundary pixels of the current catchment basin have adjacent pixels belonging to other basins. In this way, we are able to extract a single catchment basin given an image/volume dataset and a starting pixel/voxel. When volume data is processed, 26-connectivity grid is used, instead of the 8-connectivity grid applied in regular 2D images. All other steps remain unchanged. Moreover, the proposed approach can be applied even on data structures with higher connectivity in the same way.

3.2.3.2 Flooding the Selected Region

As introduced above, the local watershed transformation requires a selection of a region of interest (ROI) from the user. For user convenience, this is performed with the original input image I. In a separate preprocessing step, a simple edge detection based on the standard Sobel edge detector [Pratt, 1991] is applied to the entire input data. This is a straightforward process with insignificant time cost, compared to the global watershed transformation, hence not slowing down the entire process.

When this is accomplished, the gradient of the input area is flooded, simulating raindrops starting with an arbitrary pixel within the selected area. Afterwards, the above transformation is repeatedly performed on I' until all pixels and basins *within* the ROI are processed (as shown in Figure 3.4 for the 2D case). Since these basins be-

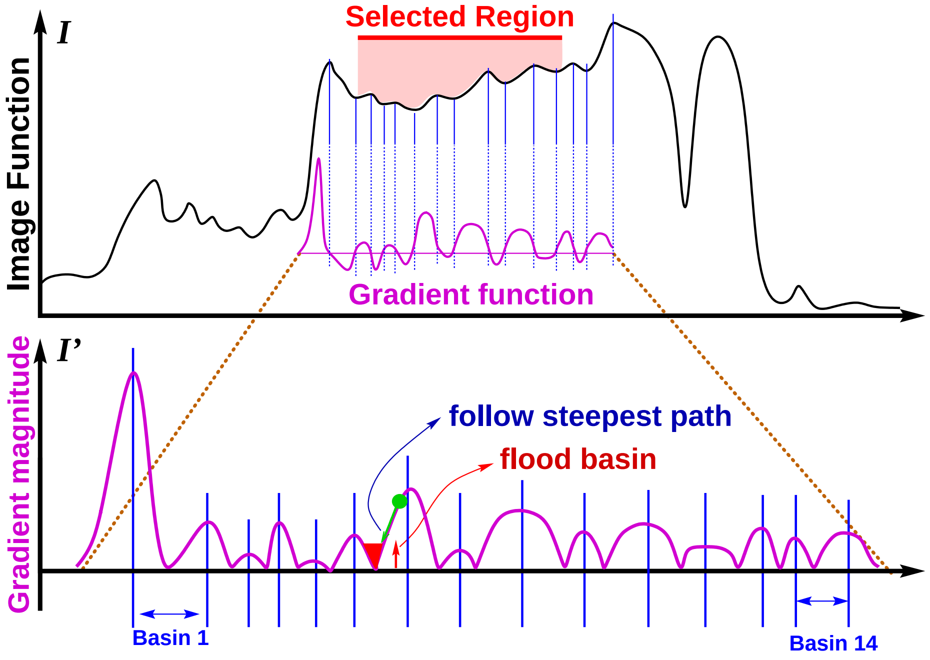


Figure 3.4: The selected ROI in a simple 2D example, the corresponding gradient magnitude image (lower diagram) and its basins.

long to the same region due to the user selection, they are used to derive thresholds for the *merging criteria*. As shown in Section 3.2.4, both the original image data I and the gradient magnitude image I' are applied.

Unfortunately, there are basins that are not completely, but only partially within the user-selected ROI. These basins are processed as follows: for each pixel, not belonging to a basin with local minimum within the selected region, again a rain-falling simulation is started (see also [Stoev, 2000]) and the regional minimum of the new basin is determined as described above (eventually outside the user selection). Each of these minima is used to start a flooding process as described before (in the Section 3.2.3.1, p. 23). The so computed layer of basins surrounding the user-selected area is utilized to derive a second set of auxiliary thresholds for the merging criteria as discussed next. When this step is completed, all basins that are partially or completely included in the selected region are processed.

3.2.4 Merging Criteria

The most important part of the proposed approach is the suitable definition of the basin-merging criteria. These criteria define the quality of the segmentation results. As introduced before, various merging criteria are described in the literature. The most prominent ones are based on gradient-watersheds on graphs [Vincent and Soille, 1991], basin dynamics [Najman and Schmitt, 1996], and multi scale gradient analysis [Jackway, 1995]. Most of these approaches define appropriate merging criteria based on the basin's features, hence using the information about the gradient image. In contrast, we implemented a method, which uses a combination of the first two approaches, additional information about the original image data I, and the second derivation of the input image I''. Thus, a more sound merging condition is introduced. Additionally, it is very difficult, often even impossible, to define a merging criterion matching the attributes of all objects in the input image. Applying a local watershed transformation and merging only basins fulfilling local ROI-specific merging criteria yields significantly better results as shown in Section 3.2.7 (p. 34).

3.2.4.1 Merging Criterion A

Now let us consider two adjacent basins as depicted in Figure 3.5. As proposed in [Najman and Schmitt, 1994] the basin's dynamic introduces a good criterion for

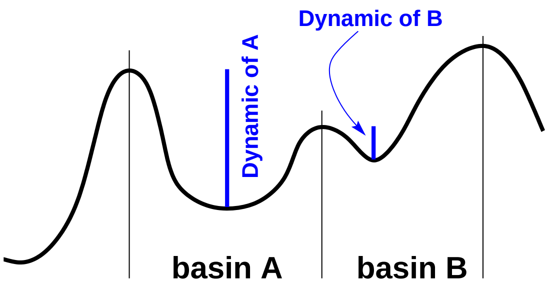


Figure 3.5: The basin's dynamic is the minimum height, which has to be overcome, in order to reach a basin with lower or equal minimum altitude.

merging adjacent basins. The dynamic of a basin is defined by the minimum height, which has to be overcome, in order to reach a basin with lower or equal minimum altitude. Basins with dynamics lower than a given value d_t (*dynamics threshold*) are now merged in larger regions. The usual values of d_t are in the interval $[5, 20]$ for 8-bit gray-scale images and depend on the image content and it's noise. Applying this strategy for basin merging helps us to merge insignificant basins, introduced by various noise sources in I and I' (see Figure 3.10 image ***b***, p. 32).

3.2.4.2 Merging Criterion B

Nevertheless, there are still situations, in which even basins with a great difference between the (mean) gray-values in I have to be merged when only their dynamic is taken into account (as shown in Figure 3.6). To tackle this problem, we applied additionally to the concept of basin dynamics a second criterion. This is based on the mean gray value of the region in I, corresponding to the basin in I'. A similar strategy is introduced in [Vincent and Soille, 1991], where the authors use the infimum of the

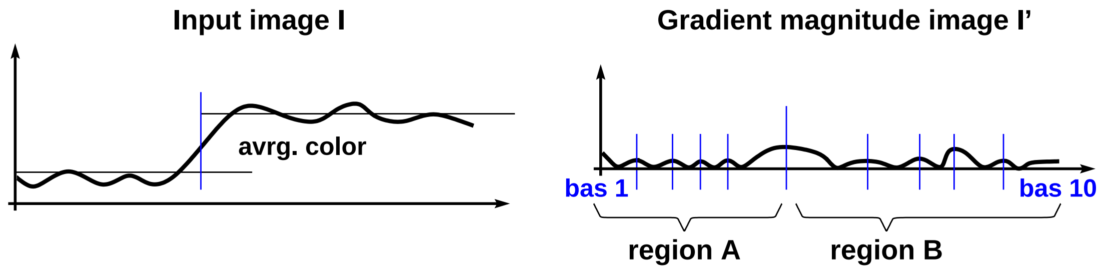


Figure 3.6: A case is shown in which the basin's dynamic only does not provide a good merging criterion.

area in I, corresponding to the basin's regional minimum in I'. Furthermore, the authors apply a watershed algorithm on a graph with nodes representing the basins and arcs linking adjacent basins. Thereby, the nodes are colored with the infimum value of the basins minimum in I. For our application, however, it turned out that applying the average color c_a of the catchment basin in the original image I is a better criterion for basin merging. Unlike [Vincent and Soille, 1991], we compare the values c'_a and c''_a of each two adjacent basins in order to determine whether they should be merged or not ($|c'_a - c''_a| < c_t$ - *color threshold*). In this way, we define a second criterion for basin merging based on the global basin attributes in I and in I'.

Another problem occurs with these merging criteria: too many basins fulfilling the requirements, but not belonging to the same region are merged. This may significantly worsen the segmentation results (see Figure 3.10, image ***c***). To circumvent this obstacle, we introduced two additional criteria for preventing basins from merging.

3.2.4.3 Merging Criterion C

The third criterion we applied in our algorithm is based on the characteristics of the dam between two catchment basins C_A and C_B. During the basin extraction, we record the length, the average color, and the color of the lowest pixel p_b on the dam between C_A and C_B in I'. If the altitude difference between the lowest common border pixel p_b and the average height of the border b_a is greater than a given threshold value b_t (*border threshold*, with values typically within the interval $[5, 15]$), the basins are not merged even if the conditions stated in the first two criteria are fulfilled (see Figure 3.7). In case the basins have more than one common border, the border pieces are processed separately. This criterion prevents from merging basins, which have small dynamics due to noise-containing borders.

3.2.4.4 Merging Criterion D

Finally, we introduced a measurement for the *steepness* of the border between two basins in I'. For this, we consider each border pixel p_b and the next pixel on the path toward the local minimum p_1, as shown in Figure 3.8. For each such border pixel p_b, we determine the pixel p_1 and compute the magnitudes of the second derivations p''_b and p''_1, hence the values of p_b and p_1 in I''. The absolute difference between these values $p_s = |p''_b - p''_1|$ gives a measurement of the gradient's variation in I'. If p_s is greater than a given threshold s_t (*steepness threshold*), for any pixel on the border between adjacent basins, this basins are not merged even if the criteria A-C are

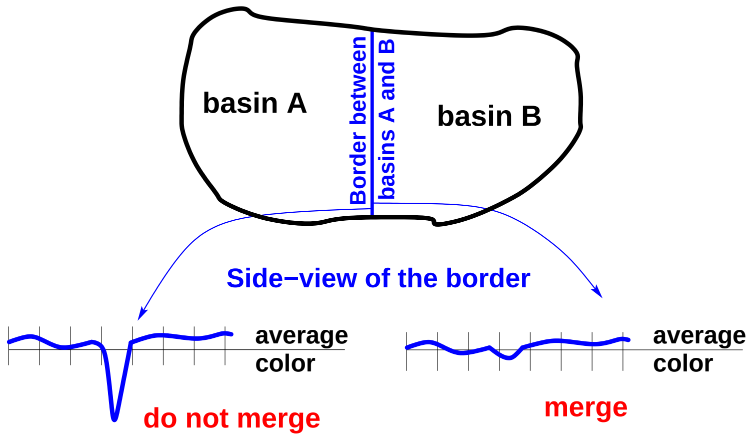


Figure 3.7: The basins are not merged if the gap between the lowest border pixel, defining the dynamic, and the average border altitude is too large as shown on the left.

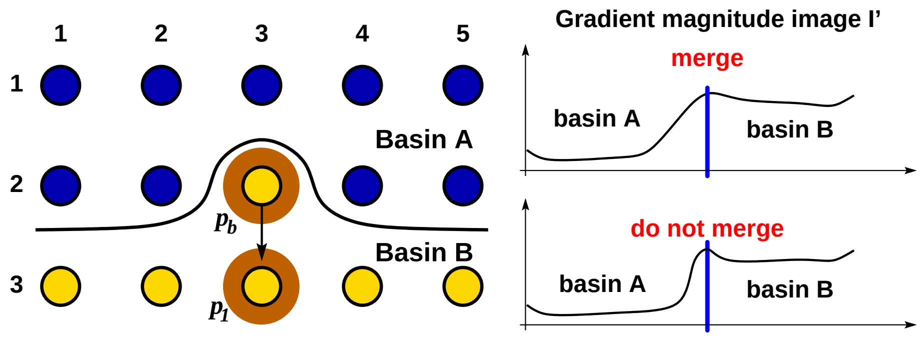


Figure 3.8: The magnitude of the second derivation is computed at each inner border pixel and at the next pixels on the path toward the basin's minimum (pixels (3,2) and (3,3)). If the absolute difference between these two values is greater than a given threshold s_t for any two border pixels, the basins are not merged. On the right, a case is shown in which the steepness of the basin's border prevents from merging two adjacent basins.

fulfilled (see the right of Figure 3.8). This criterion is applied to each two adjacent basins (for instance A and B in Figure 3.8). The border steepness of both of them has to be below the given threshold (in general within the interval $[5, 20]$), in order to merge the basins.

To summarize, for each extracted catchment basin A, we evaluate the following data structure in order to determine whether it should be merged with an adjacent one B:

- the average gray value of the basin in I (criterion B);
- the lowest and highest altitude of a basin pixel in I' (criterion A,C, and D);
- lowest and average border value for each pair of adjacent basin (A, B) (criterion C);
- for each inner pixel of the border and each adjacent basin the maximum p_s (criterion D).

Results of applying the various merging strategies are shown in Figure 3.9 and Figure 3.10. We first show the result of applying the standard global watershed transfor-

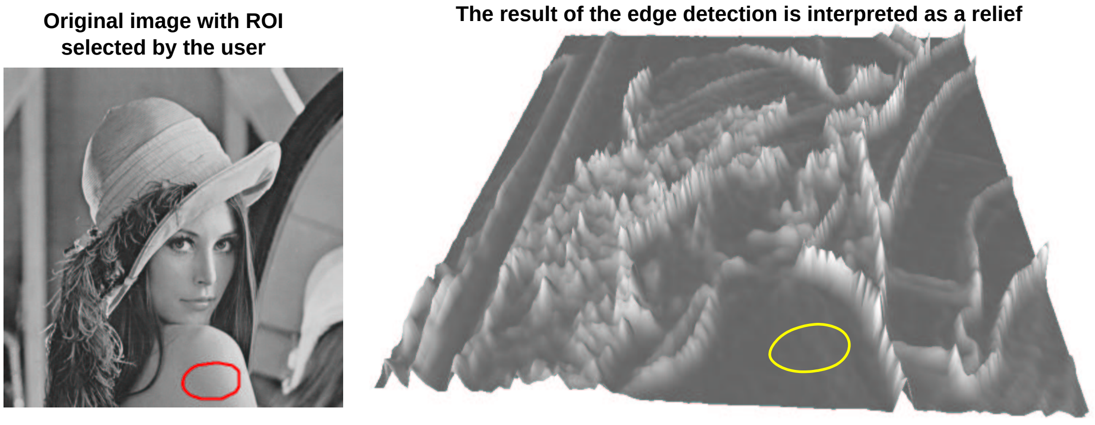

Original image with ROI selected by the user
The result of the edge detection is interpreted as a relief

Figure 3.9: Selected area in the original image and the corresponding area in the edge detected image.

mation and the application of the criteria A and B. Afterwards, the results of applying the local watershed transformation are shown evaluating the criteria A and B in image ***c*** and criteria A-D in image ***d***.

3.2.5 Evaluation of the Merging Criteria

After flooding the ROI selected by the user, we compute the values for the thresholds: $T = (d_t, c_t, b_t, s_t)$. The value of d_t is the value of the greatest dynamic for basins within the selected ROI (criterion A). c_t is defined as the greatest allowed mean color deviation for two adjacent basins (criterion B). When all basins in the given ROI are computed, the mean color of these regions is used to define the reference gray value. c_t is now applied for defining the deviation threshold relative to the reference gray value. b_t is initialized with the maximum difference between the average border color and the lowest border pixel for each pair of adjacent basins (criterion C). In contrast to d_t and c_t, the value of b_t may require additional adjustment depending on the noise of the basin border in the selected region. Finally, the magnitude of the greatest second derivation of the gray value in I' on the inner border is assigned to s_t (see Criterion D above).

In order to determine whether a pair of computed catchment basins A and B match the initial user selection, we evaluate the following *merging rule*, applying the thresholds T:

```
IF Criterion_A($A, B$) AND Criterion_B($A, B$)
   AND Criterion_C($A, B$) AND Criterion_D($A, B$)
THEN Merge_Basins($A, B$).
```

Since there are basins, which are only partially covered by the selected ROI, these border basins are utilized in the same way as this was done for T, for deriving an additional threshold set: $T' = (d'_t, c'_t, b'_t, s'_t)$. When a new basin is extracted, we first check whether the premise of the above rule is true for the threshold set T. If this is the case, the basin is assigned to the initial ROI. Otherwise, the second threshold set T' is applied and if the rule's premise is 'false', the basin is definitely marked as not matching the user selection. Finally, if the condition of the above rule is true applying of the second set of thresholds T', the current basin is assigned to the layer of basins surrounding the user selection and processed as described next (see Figure 3.10 gray area in image ***d*** and Figure 3.11).

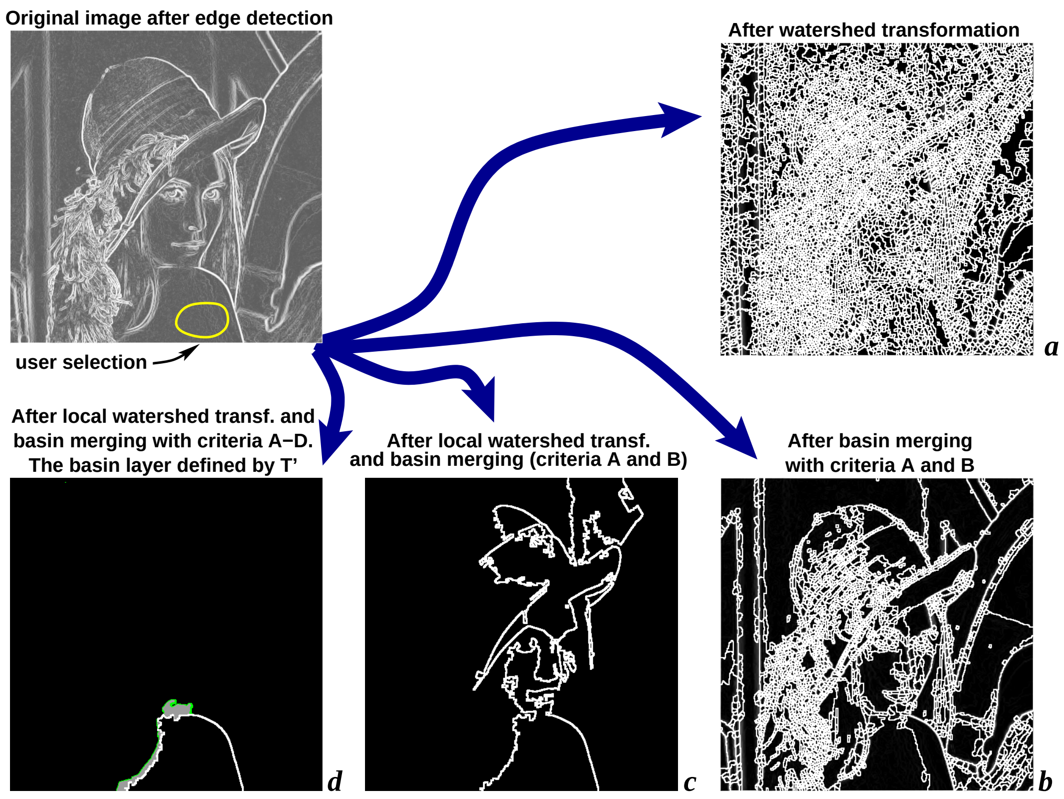

Original image after edge detection
After watershed transformation
user selection
a
After local watershed transf. and basin merging with criteria A–D. The basin layer defined by T'
After local watershed transf. and basin merging (criteria A and B)
After basin merging with criteria A and B
d
c
b

Figure 3.10: Results of applying the watershed transformation. Image ***a*** shows the result of applying the standard algorithm, which obviously produces heavily over-segmented images. After merging the basins with the described criteria, more meaningful segmentation is achieved as shown in image ***b***. Image ***c*** shows the results achieved with the proposed local watershed transformation, starting with the selected ROI and applying the criteria A and B. Applying all of the proposed criteria (A-D) results in image ***d***. The basins in $\mathcal{B}_c$ are also shown in image ***d***.

3.2.6 Basin Growth

After the user input is processed so far, the algorithm starts with the controlled basin-level region growth. At this stage all basins completely or partially within the selected ROI and a one-pixel thick layer surrounding these basins are processed. When following the path of steepest descent for each of the pixels in this layer, the reached minima are completely outside the given ROI and thus not processed (flooded) yet. The regional minima of these basins, however, are known and the flooding process can be started as described in Section 3.2.3.1 (p. 23). In this way, in every basin-iteration a set of basins surrounding the ones extracted in the previous step are computed.

Every time a new catchment basin C is extracted, its attributes are applied for evaluating the merging rule with the threshold set T. If C matches the user selection, that is if the rule's premise is true, it is assigned to the initial ROI. Unprocessed basins, adjacent to C are pushed on a queue $\mathcal{M}$ containing the yet unprocessed minima and are flooded in the next basin iteration. Otherwise, if C does not match the user selection, the merging rule is evaluated again with the thresholds T'. If the rule's premise is true this time, the basin is assigned to a set of potential candidates for merging $\mathcal{B}_c$. Any extracted basin C' adjacent to a basin in $\mathcal{B}_c$, but not having any adjacent basin belonging to the currently extracted ROI can be only added to $\mathcal{B}_c$ but not to the ROI itself. On the other hand, if there is an adjacent basin in the ROI, the basin C' can be assigned to the ROI if the rule's premise is true for T. Finally, in case the rule's condition is false for both threshold sets T and T', the current basin is a *border basin*, not matching the initial user selection (see Figure 3.11). In the following, this algorithm is stated in pseudo-code:

```
/* $\mathcal{M}$ – contains yet unprocessed minima */
do
  $N \leftarrow$ card($\mathcal{M}$);
  while ($N > 0$)
     $N \leftarrow N - 1$;
     $M \leftarrow$ pop($\mathcal{M}$);
     $C \leftarrow$ flood($M, \mathcal{M}$);        /* basins adjacent to $C$ are added to $\mathcal{M}$ */
     if (merging_rule($T, C$))         /* check threshold set $T$ */
        assign_to_ROI($C$);
     else if (merging_rule($T', C$))         /* check threshold set $T'$ */
        assign_to_potential_ROI($C$);        /* assign to $\mathcal{B}_c$ */
     endif;
  end;        /* while ($N > 0$) */
while (card($\mathcal{M}$) $\neq N$)        /* until no new basins are added */
```

The mosaic of un-merged basins in $\mathcal{B}_c$ and their adjacency we utilized for the definition of an adjacency graph $\mathcal{G}$. In this graph, the nodes correspond to the basins, the arcs link adjacent basins, similar to the approach introduced in [Vincent and Soille, 1991]. The nodes within the initial ROI and the ones merged during the basin growth are pooled together in a node U (as depicted in Figure 3.11). The basins, for which the merging rule turns true only for the second threshold set

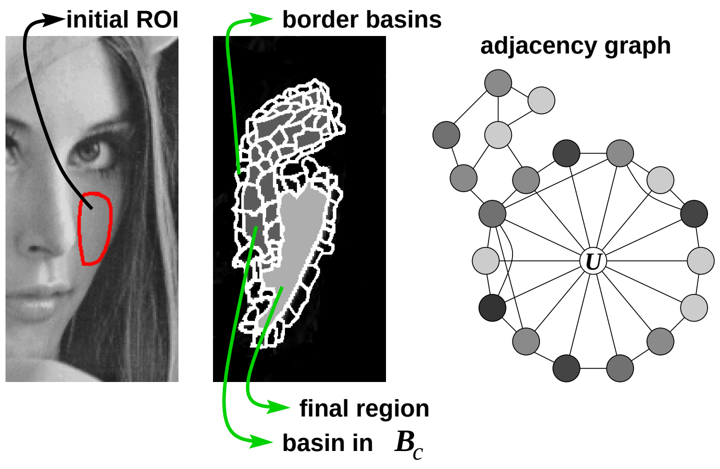


Figure 3.11: The user selection on the left and the extracted region in the middle. The graph on the right shows (a part of) the (colored) adjacent basins in $\mathcal{B}_c$.

T', are also assigned to the graph. However, they are not merged with the node U, but are connected with each other and with U. In addition, the following condition can be added to the previous algorithm: if the basins C_i adjacent to a flooded basin C' are not part of U, thus not meeting the merging criteria with threshold set T but only with threshold set T', C' is automatically assigned to the set of potential regions of interest $\mathcal{B}_c$. Each of these basins contains the attributes listed at the end of Section 3.2.4. Thus, in a post-processing step, the user can manually assign basins out of $\mathcal{B}_c$ to the initial ROI. Furthermore, the threshold set T can be updated each time a basin is manually assigned to U. This makes a precise final basin-level-adjustment possible, without having to explicitly manipulate single parameters.

In both cases, the result of the proposed algorithm is a set of connected basins, extending the initial ROI. In other words, the input data is classified and marked in such a way, that there is a connected region of interest and an area not belonging to the ROI. In the 3D case, this data can be used to generate a surface surrounding it (see Figures 3.12, 3.13), for volume rendering, or for raycasting [Tiede et al., 1998].

3.2.7 Results

To demonstrate the power of the proposed approach, it was applied to several images and two volume datasets. Each dataset was first preprocessed and the standard Sobel edge detector [Pratt, 1991] was applied either to the image data in the 2D case, or to each slice in the volume data.

In the next step, a slice was selected out of the two volume datasets and a region

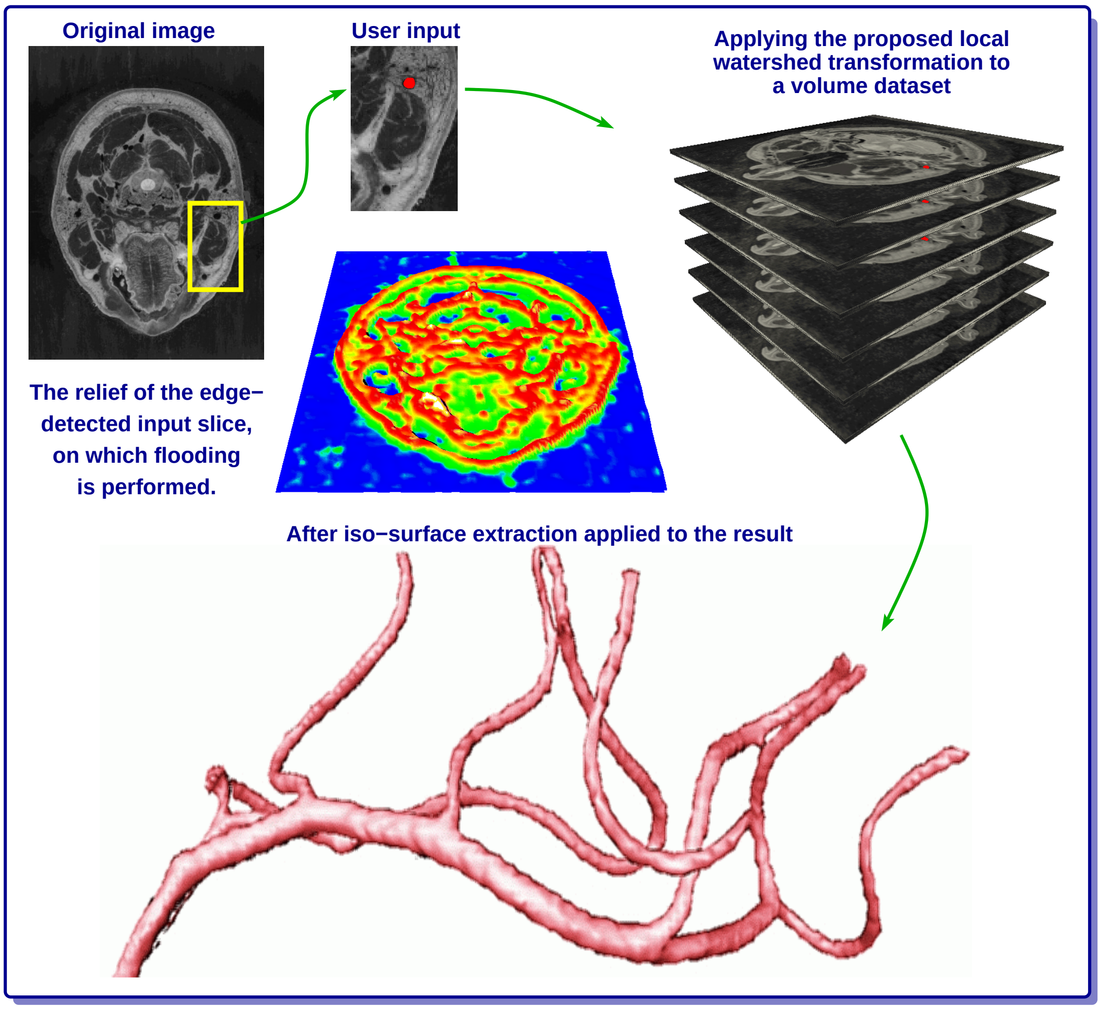

Original image
User input
Applying the proposed local watershed transformation to a volume dataset
The relief of the edge−detected input slice, on which flooding is performed.
After iso−surface extraction applied to the result

Figure 3.12: The surface in the bottom is extracted with the marching cubes algorithm, after performing the basin growth approach.

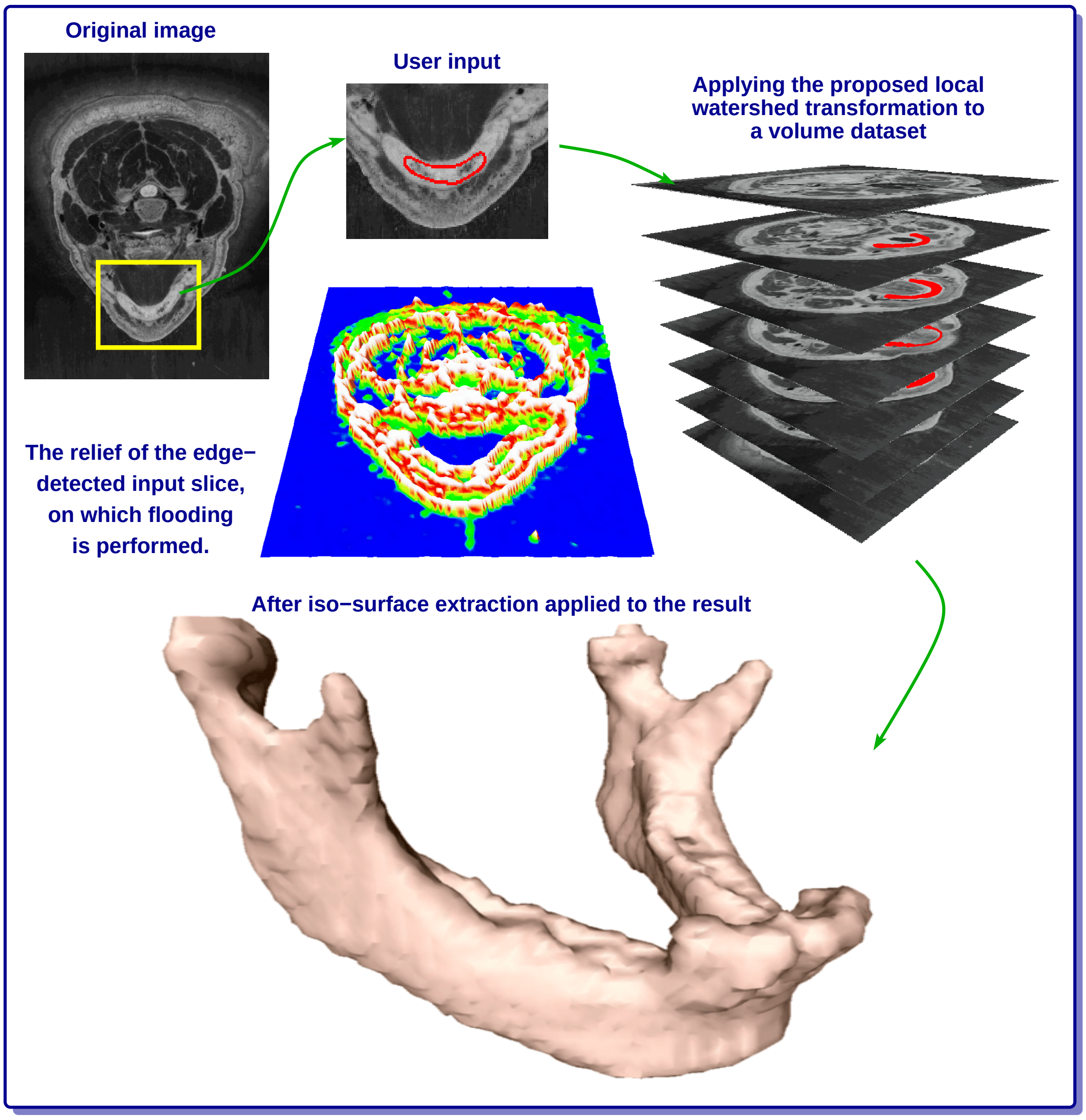

Original image
User input
Applying the proposed local watershed transformation to a volume dataset
The relief of the edge−detected input slice, on which flooding is performed.
After iso−surface extraction applied to the result

Figure 3.13: The surface in the bottom is extracted with the marching cubes algorithm, after performing the basin growth approach.

was marked (as depicted in Figures 3.12 and Figure 3.13). Afterwards, the algorithm described above was applied on a 26-connectivity grid. The computed results consist of a set of marked pixels. In order to visualize this region we extracted a surface applying the standard marching cubes algorithm [Lorensen and Cline, 1987]. Unfortunately, this may introduce 'stairs' in the surface. To solve this problem, the basin extraction has to be combined with the surface extraction. Since borders between adjacent basins are in general smooth, this would improve the quality of the generated surface.

The image data was processed in a similar way, however, without applying a contour extraction algorithm. Instead, after selecting a region and applying the proposed technique, the pixels defining the border between the marked region and the unmarked background, were colored red as shown in Figure 3.14. The proposed approach was applied three times, for three different objects of interest in the original image. Each time the coarse contour of the target object was extracted appropri-

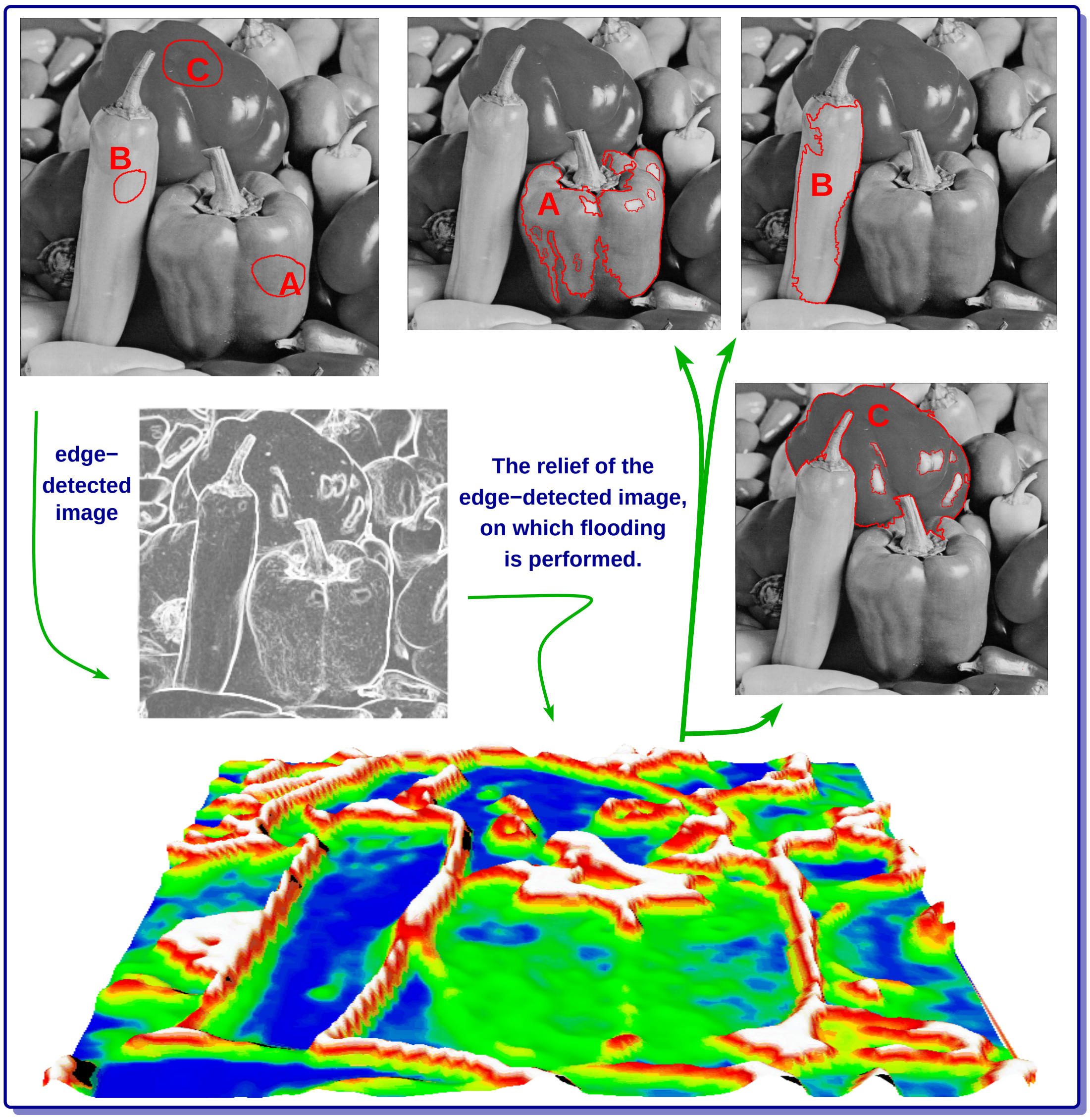


Figure 3.14: Results of applying the proposed local watershed transformation on a 2D image with three different user selections.

ately, even though the original picture contains noise and is hard to segment. For each of the examples, the 2D images and the 3D volume sets, we have displayed the computation times in Table 3.1. These times depend on the size of the extracted object and not on the size of the dataset. As expected, the required time for the three-dimensional input datasets were significantly longer than for the 2D images.

3.2.8 Summary

In this section, we presented a new semi-automatic method for extracting regions of interest based on a local watershed transformation. We described how a catchment basin is computed, given an initial pixel in the dataset. The proposed technique does not require preprocessing of the data, while applying a modified rain-falling simulation.

Unlike the standard watershed transformation, which floods the entire input

Dataset	Data size	Number of regions	Computation time
Jaw bone	300x400x50	36542	14.07 sec
Vessel tree	512x512x64	54263	21.6 sec
Peppers Region A	512x512	2320	2.8 sec
Peppers Region B	512x512	3859	3.91 sec
Peppers Region C	512x512	3619	3.34 sec

Table 3.1: Computation times for the example volumes and images measured on an SGI O2 with 180MHz R5000.

data, the presented approach computes and merges only the catchment basins fulfilling a set of criteria. In addition, the standard approach attempts to define basin-merging criteria for all objects in the input data. In contrast, the proposed technique utilizes the introduced criteria applying local attributes of a given user-selected region of interest. The features of a coarse manual user selection are utilized for introducing the threshold values for four criteria for merging the catchment basins in a meaningful region.

Furthermore, the described technique can be applied more precisely, since it is easier to define merging criteria for basins within *one* object of interest in the input data, than for the entire dataset. The merging rule introduced in this way, utilizes basin attributes like the basin's dynamics, its average gray-color, as well as border characteristics like it's steepness and gradient magnitude. This significantly improves the results computed with the proposed technique, offering a reasonable solution to the segmentation problem.

3.3 Historical Data Visualization

The second visualization area we address in this work is the visualization of historical data. Despite the progress in the area of scientific visualization and information visualization, historians often still utilize the approved 2D maps (see Figure 3.15), combined with drawings and accompanying data sheets. These requisites, however, make it very difficult to analyze, interpret, and comprehend events in time, looking

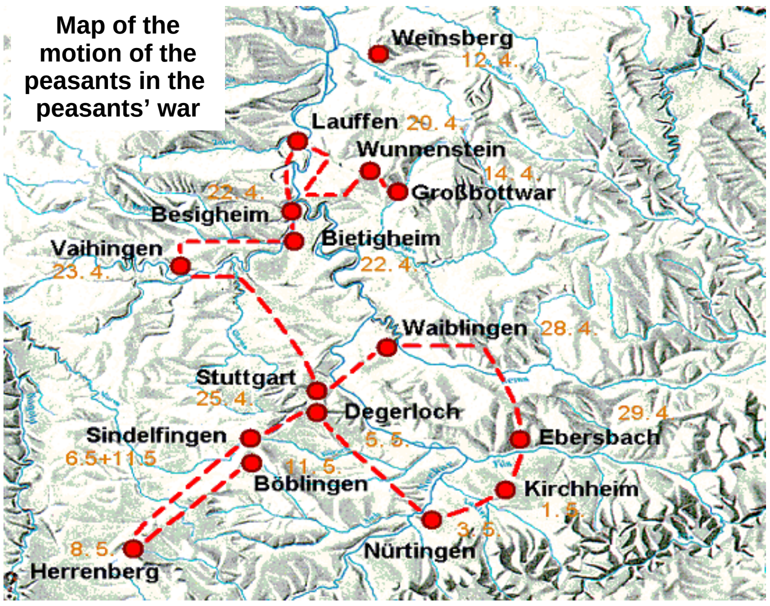


Figure 3.15: Example of a map used by historians to visualize events in time.

for answers to particular questions. For instance, it is often incomprehensible why a given motion of people was performed in a particular way. Displaying the motion on a three dimensional map, on which terrain facts and other local conditions are shown, often gives explanation to such questions. Furthermore, the visualization of the time-dependent events may also be hard to understand (see Figure 3.15). In this case, the chronological display of events and the ability to interactively explore the time dimension may significantly contribute to the detailed study of the data. Therefore, the employment of computer graphics may significantly facilitate the exploration, analysis, and teaching of complex *time dependent* events in the *space*.

Although we are able to visualize almost any kind of data nowadays, the visualization of historical data is still a challenging task. The problem with 3D space *combined with* an additional time dimension in general, and with historical data in particular, is the fourth dimension– the time. Until now, the visualization of and the interaction with this kind of data has not been subject of active research. There have been only few attempts to offer interactive exploration. Unfortunately, the results are usually (VR-) techniques tailored to the specific problem. Thus, the developed concepts are hard or even impossible to adapt to applications other than the one they have been developed for. Nevertheless, the exploration of historical data in a VR-application may significantly improve the interpretation and help understand the visualized events and dependencies. Especially when complex migrations and battles are displayed on a 3D map (e.g. using tabletop displays), a VR-application can be a valuable tool offering comprehensive visualization.

Conceptually, the problem of visualizing historical data is similar to the problem of visualizing particles and their motion in time. The data contain a time-invariant component (e.g. terrain) and a set of event points on the time axis. Unlike particle visualization, however, we are not interested in the entireness of the particle system, neither in all particle paths representing the particle flow. Instead, we are interested in the single time steps and the events happening at a given point in time or in a given time interval. Therefore, we cannot adapt techniques applied in particle flow simulation application to the historical data visualization context.

The simplest way of visualizing two-dimensional data containing a time dimension is the generation of traditional image sequences (movies). Such image sequences are in general displayed with a constant speed. Thus, the way the "movie-time" (*event time*) is elapsing relative to the user time is fixed. Since we cannot control the user time, the only way to manipulate the relation between both time scales is to let the event time elapse faster or slower with respect to the user time.

More challenging is the scenario where the visualized data consist of three-dimensional data and an additional time dimension. Often this case is reduced to the definition of a fixed path through the three-dimensional dataset with predefined speed of motion and elapsing event time. Thus, the result of the visualization is an image sequence, where the challenge is the implementation of the appropriate camera control. A number of researchers have addressed the issues behind camera control in manipulation and exploration applications [Brooks, Jr., 1988, Ware and Osborne, 1990, Ehricke et al., 1993]. Unfortunately, the resulting image sequences are not as interactive as desired and do not allow free motion in space and time. Such a simultaneous motion in all four dimensions is difficult to display, since we are unable to view more than three dimensions at a time.

There are also various contexts, in which the visualized data contain time-dependent component. Fedak et al. [Fedak et al., 1996] describe a system for underwater tracking and visualization of seals and whales. However, they only address the three-dimensional data exploration and do not consider the interactive exploration of the fourth dimension. When the time is taken into account, most of the efforts result in a movie generation, for which the challenging task is the camera motion in space and time. Palamidese [Palamidese, 1996] describes a context driven camera motion. Drucker and Zeltzer [Drucker and Zeltzer, 1994] propose camera controls based on an analysis of the tasks, which are required in a specific environment. Gleicher and Witkin [Gleicher and Witkin, 1992] address a similar problem. They propose methods for camera manipulation using features seen through the lens of the camera. Finally, Galyean [Galyean, 1995] presents a different approach called *the river analogy*. He describes a system, which pulls the user within a given time along a predefined path. This guidance guarantees that the user will not miss given events, while providing some degree of interactivity. He applies a spring/damper system with variable spring/damper constants. One end of the spring is attached to a particle continuously moving along the predefined path. The virtual camera is attached to the other end. Depending on the location of the moving particle on the path, different values for the spring/damper system are applied. Thus, different degree of interactivity is given to the user. Even though the river flow is a simple and powerful concept, it is predefined and constant, thus, no interactive *time* control is possible. Another drawback of the approach is the inability to manage free exploration of the space, i.e. the user cannot move freely in space.

Throughout this work, we demonstrate the entire process from the data acquisition to the visualization and guided exploration of a particular event: the peasants' migrations and battles during the peasants' war in 1525 in Germany. Even though, we present only this example, the proposed techniques can be used for visualizing any type of motion in time, no matter whether this is a 90 minutes hiking trail or species migration with duration of many thousands of years. In addition, the techniques proposed in Section 3.3.3 allow simultaneous exploration of any arbitrary number of motions in time. The only requirement is that the underlying terrain model does not change within the visualized time interval. If this is not the case, we cannot guarantee that the projection of the data contained in a given time interval parallel to the time axis will produce a sound image.

The remainder of this section is organized as follows. First, we introduce in detail the data acquisition and the preparation steps. Afterwards, we describe how the data is visualized and different parameters are encoded in the displayed extrusion. We also present a set of techniques for improving the exploration of the data, which are independent of the interaction. In Section 4.5, we discuss in detail a set of techniques for enhancing the interaction and visualization of historical data in virtual environments. We also provide a summary of the interaction concept, as well as a taxonomy of the interaction types in the four-dimensional space.

3.3.1 Data Acquisition and Preparation

The data originate from notes and delineations recorded during the peasants' war in the spring of 1525. They contain the route of the peasants, from village to village gathering people and protesting against the feudal system and the growing taxes. Unfortunately, the original data sheets were destroyed during world war two. We only have copies of the delineations and notes made before they were destroyed. The copies show the motion of the peasants as a simple polygonal line l_o (see Figure 3.16). These documents, originating from the time of the peasants war, are hand made and hence not very accurate. Therefore, no direct mapping onto a terrain model followed by instant visualization is possible, since the high precision terrain model would not match the inaccurate hand-made drawings. Instead, in the first step we had to transfer by hand these data (l_o) onto a 2D topographic map l_m.

On the other hand, we have a triangulated terrain model of the corresponding 2D topographic map. Thus, the next step of the data preparation is to assign height to the 2D line l_m. For each point, we can determine the height by interpolating the z-coordinate of the corresponding triangle points. For each pair of points (a, b) of l_m, not lying in the same triangle, all intersections of the line (a, b) with triangle edges are computed (see Figure 3.17). This may cause the addition of auxiliary points on each edge intersection. Applying this approach to all point pairs in l_m results in a three-dimensional line l_t, which lies exactly on the surface of the terrain.

The digital elevation model is acquired from a real digital elevation model with 5 meters resolution. For performance reasons, we have simplified the model. Thus, our final dataset consists of not more than 25'000 triangles. This amount of data can be rendered with reasonable frame rates of at least 25fps on currently available consumer graphics hardware.

At this stage of the data preparation, we have a three-dimensional line l_t. However, we do not have yet information about the time values associated with it. Fortunately, for some of the places visited by the peasants, we know the exact arrival and departure times, which were recorded by participants in the war. These points define the mapping of the 3D polygon l_t into the four-dimensional space (l_f). The times associated with the points between two time-sampled points are interpolated. In other words, we assume that the speed of the peasants' movement is constant in

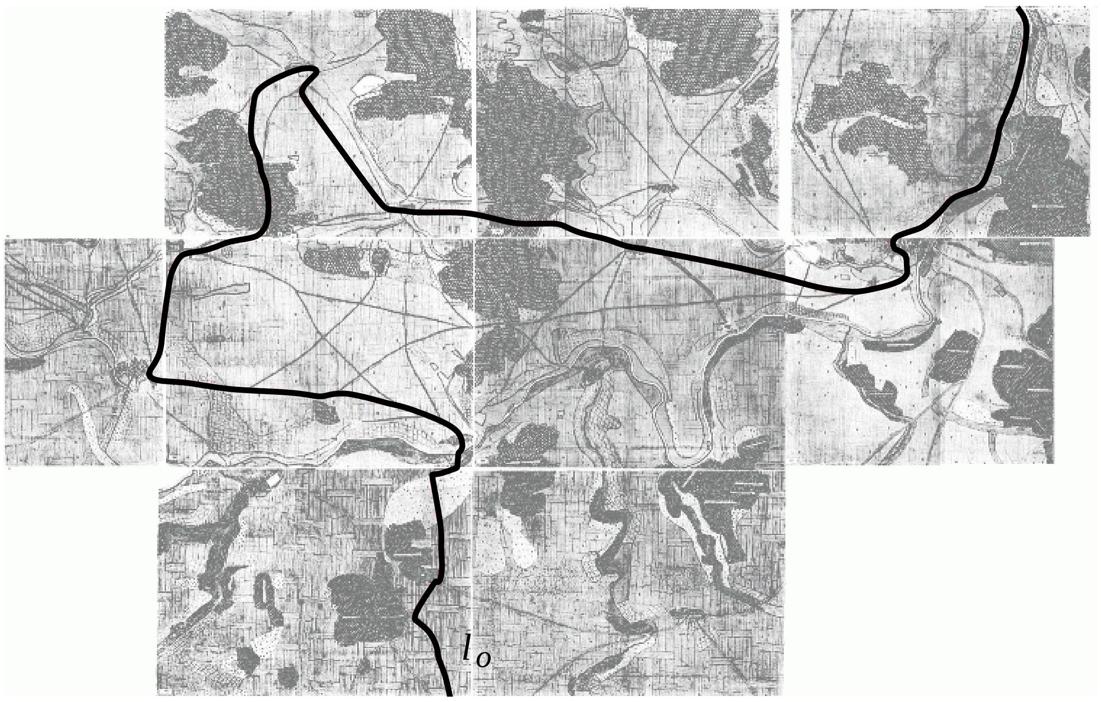


Figure 3.16: The copies of the original maps showing the path of the peasants. We only have single pictures (therefore the gaps between the images), copied before the originals were destroyed.

each segment between two time-sampled points. Thus, to each intermediate point we assign a time value depending on the distance to the two adjacent time-sampled points, concerning the time. In the remainder of this work, we call the polygon pieces between two time-sampled points *timed segments*. The entire data preprocessing pipeline is shown in Figure 3.18.

In order to enhance realism, instead of assuming constant speed, we also introduced constraints to the interpolation of time between two time-sampled points. In particular, we assigned decrease of the velocity depending on the elevation, respectively acceleration of the velocity in parts with decreasing elevation. This last step in our data preprocessing pipeline is done for each pair of points on the line l_f. After it is accomplished, we can start with the data visualization.

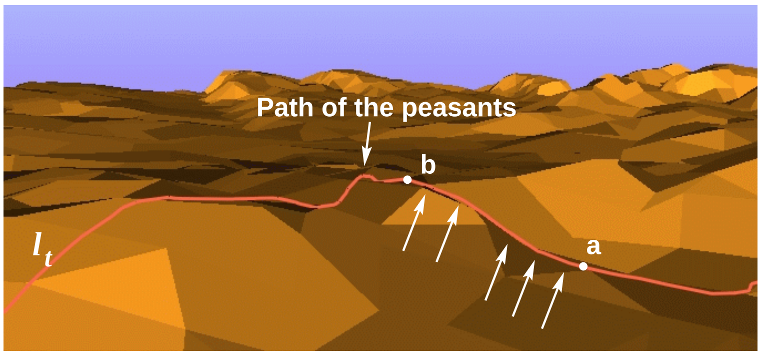


Figure 3.17: The 3D path mapped on a triangulated terrain model. The arrows indicate the points inserted on triangle edge intersections with the line l_m.

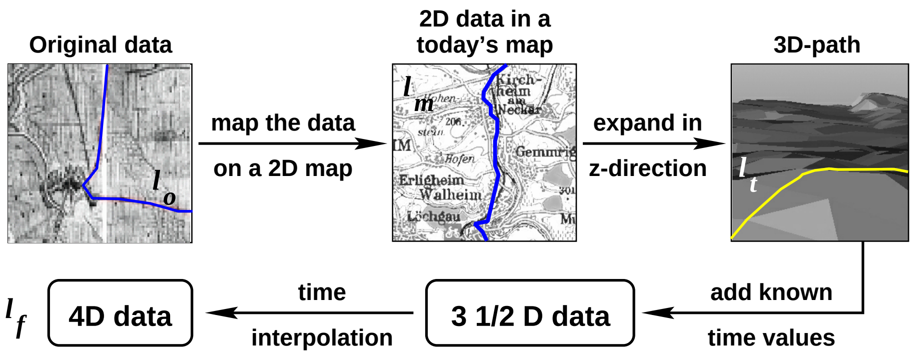


Figure 3.18: Data preprocessing pipeline. Starting with the original data, the arrows show how it is transformed in order to generate real 4D data.

3.3.2 Data Visualization

There are various ways of displaying four-dimensional data. The most convenient way is to compute a projection along one of the four axes. This reduction of a four-dimensional problem to a three-dimensional visualization is applied in our historical data visualization as well. In particular, we are always displaying three-dimensional slices of the four-dimensional space orthogonal to the time axis. In contrast to a movie, which undergoes the same transformation (from 3D to 2D), the tools we propose in Section 4.5 enable *interaction* with the 4D data, and not only equidistant 2D slicing of a 3D dataset (i.e. showing a movie). In our scenario, each displayed 3D scene is either a real slice of the 4D space orthogonal to the time axis, or a projection of a *time interval* parallel to the time axis. In the latter case, the result of the projection is a set of points each of which originally lay on a time slice of the 4D space. In case the time value of a queried point (the fourth dimension) lies between two time-sampled points (thus 3D slices of the 4D space), we perform four-dimensional linear interpolation before the point is projected parallel to the time axis. The connection of all projected points in the resulting 3D image corresponds to the path (i.e. part of l_f) of the tracked subject within the given time interval.

In general, the path of the peasants is displayed always for a given time interval. In other words, we have points in time, which are connected in order to express the motion of the migration. Aiming to display more than the sole way of the migration, we use geometry with varying size in order to encode the number of participants in a particular time segment. Therefore, the migration is displayed as a set of truncated cones. For the time-sampled points, we also know the exact numbers of the participants, which are interpolated between the time sampled points (see Figure 3.19). Furthermore, the geometry's color can be used to encode any other three-dimensional parameter (RGB-space) (see image B in Figure 3.20). This may be the mood of the troops, the affiliation of the troops, their religion, belligerence, etc.

For visualizing the data, we utilize a modified version of the OpenGL's extrusion library [Vepstas, 2000]. This library provides a set of functions for extruding a 2D geometry shape along a given three-dimensional path. However, the extrusion has to be computed for every frame, even if the input data do not change. For performance reasons, we modified this library in such a way that the extrusion is computed only once in a preprocessing step, whereas the resulting triangle strips are stored. During visualization, triangles belonging to sample points within the currently activated time interval are displayed using the pre-computed geometry. The beginning and

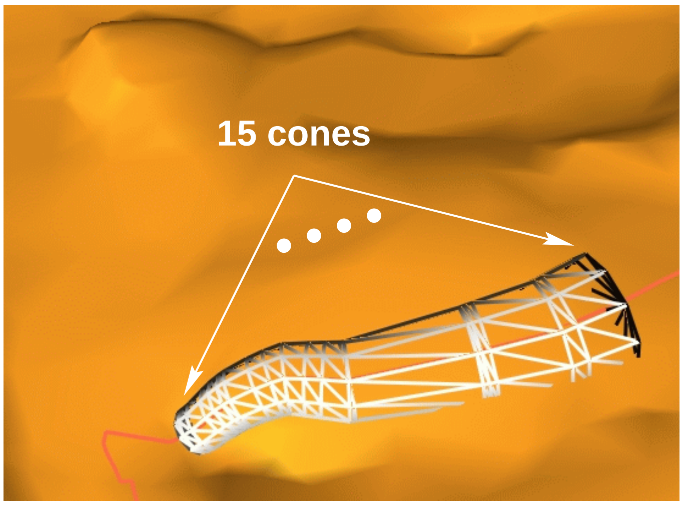


Figure 3.19: For the visualization we utilized truncated cones with different radii, in order to enable displaying of time varying data.

the end of the extrusion have to be computed each time the current time (interval) has changed. This step is necessary, since the interactively adjusted time values (beginning and end of the time interval) can lie, considering the time, between two points on the four-dimensional polygonal line l_f. Thus, we have to interpolate between these two points.

With this preprocessing, we achieve rendering times of 40 fps on average with a terrain model consisting of 25'000 triangles and extrusion with approximately 40'000 triangles: 750 sample points, for each pair of points 40 triangles, plus additional triangles for the joints between each two truncated cones[1].

3.3.3 Simultaneous Display of Multiple Extrusions

When we intend to visualize many timed segments simultaneously, we treat each of them as a single unit, which is displayed the way described above. For the realization, we had to introduce a single *global event time* parameter that can be interactively manipulated by the user. During rendering, each timed segment draws its parts in the given global time interval, if there are any visible parts at all. This enables the visualization of complex battles (see Figure 3.20) and even tracking (parts of) several people's lives simultaneously. The only required information is the mapping onto a 3D terrain and assigning time values to the extrusion, vide supra.

3.3.4 Landmarks and Orientation

In Section 3.3.1, we have described how the original polygonal line is mapped on a today's topographic map. This map can be used as a texture, which is mapped onto the terrain model (see image B Figure 3.20). Even though the texture may improve the orientation and help understand the data, due to the limited resolution and the poor readability it does not provide the expected enhancement.

On the other hand, we have the exact descriptions of the timed segments including, besides the number of the people being there and the arrival/departure times,

[1]All measurements are performed on a PIII 766 with NVidia GeForce 2 GTS.

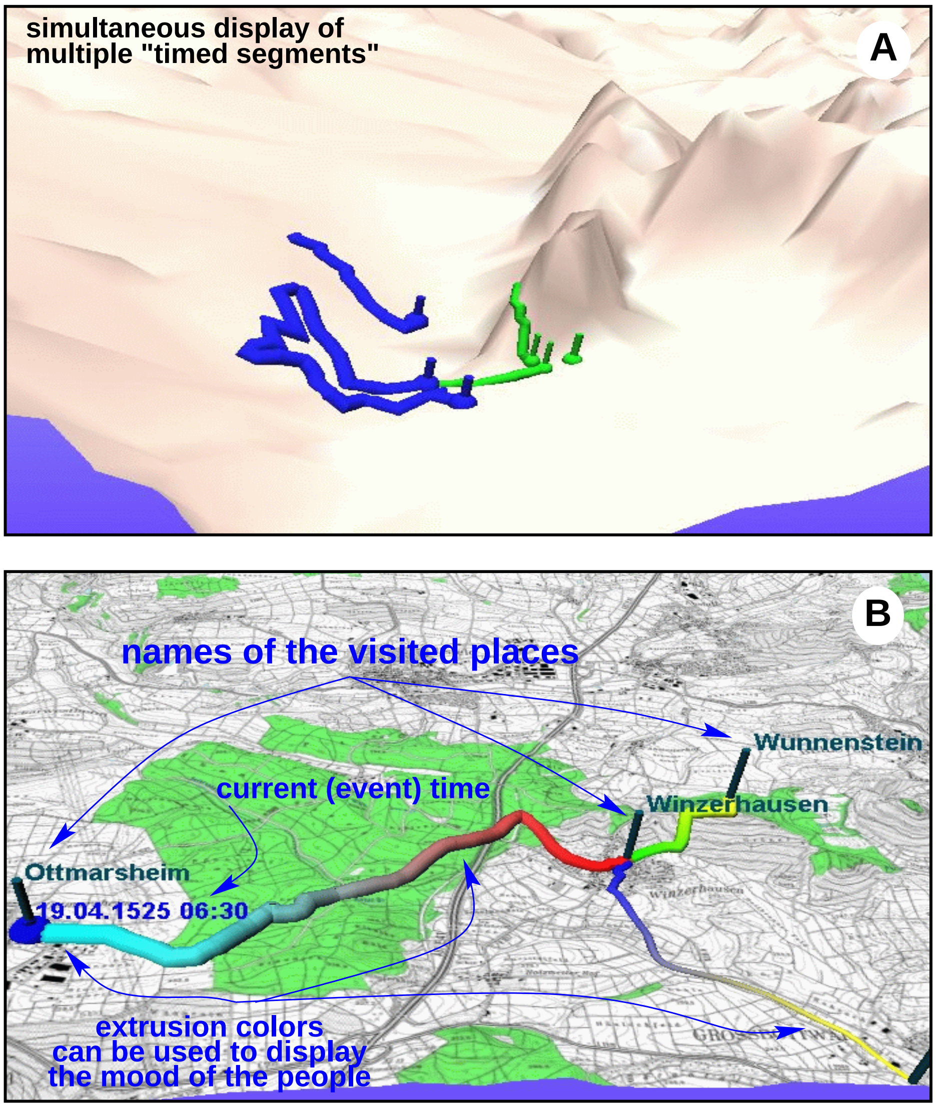


Figure 3.20: Image A shows a battle consisting of several single timed segments active in the current time interval. Image B shows some of the additional visualization aids including the names of the places, the time of the extrusion's tip, and the topographic texture.

the names of the places they stayed at. In order to provide this useful information, we display flags with the names of known places visited by the peasants. These names often offer more concrete information about the drawn extrusion than the flat texture.

Furthermore, the user is free to activate an option for displaying the current time above the tip of the extrusion. Thus, the user knows not only the position in the space, but also the current position in the time dimension. In order to enhance the navigation we discuss in Section 4.5, we also display the events within the current time interval (date/time and place's name) and the closest event in the future time with its name and date/time. In this way, the user is enabled to plan the navigation and to estimate the duration of a currently not moving extrusion. For instance, when the migration stays at a particular location overnight, the current time and the departure time, thus, the next event in time, are displayed simultaneously.

3.3.5 Daytime Modeling

Since the time can be interactively adjusted in our system, the user can feel lost in time. In other words, it is hard to display how fast/slow the time is elapsing, or what the current time is. This is especially confusing when the visualized data is not moving for a longer period of time (e.g. couple of days) and when the time display option is not active.

In order to circumvent this obstacle, we implemented a daylight colored background. Depending on the current daytime, which is always the end of the active time interval (tip of the extrusion), the colors used for drawing the background are computed adaptively (see Figure 3.21). Thereby we do not consider the seasonal

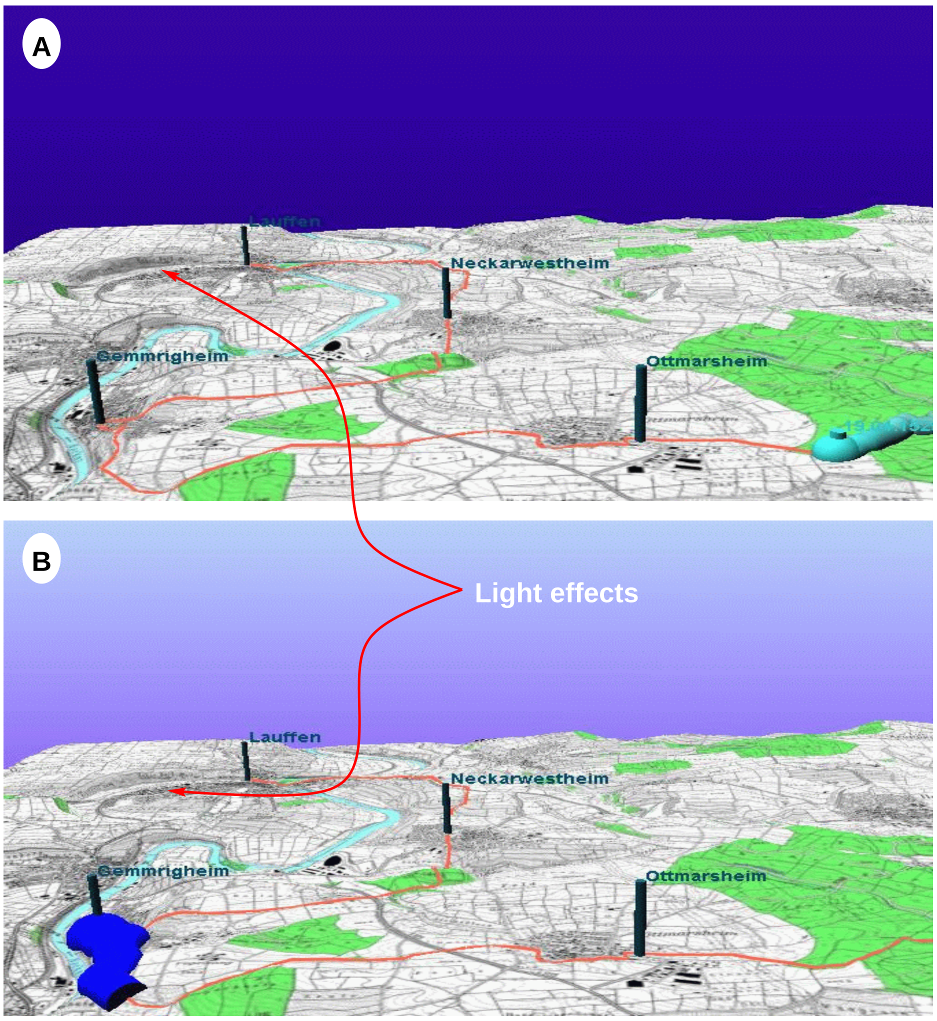


Figure 3.21: Image A shows the extrusion without the additional light source (sun light) and with dark background rendered as a night image. In contrast, image B shows the extrusion with bright background and with additional light source as rendered during the daytime.

variation of the daylight. In particular, we simulate nightlight, daylight, sunrise, and sunset. Each time the scene is redrawn, we display an interpolated value as a background.

For completing the realistic effect, we also let a light source with time dependent color rotate among the y-axis around the scene, thus simulating the motion of the sun. Since it has been shown that shadows and lights are important cues for the depth judgment of a subject, besides the visualization, this technique also improves the depth perception.

3.3.6 Summary

Armed with these display techniques, we are able to visualize slices of the 4D space for particular time values and time intervals between two such slices. These visualization tools make up the basis of the analysis and the interactive exploration of historical data. The interactive manipulation of these time values, as well as other forms of interaction and display of the data are discussed in detail in Section 4.5. Before discussing these techniques, we first introduce the main concept of the proposed tools and elaborate on the implementation of general techniques for interaction in virtual environments.

Chapter 4

Interaction Concepts

The research in this direction came into being because in the author's opinion VR-applications still lack techniques implementing tools for exploiting the *virtual* nature of applications and allowing more versatile insight into the virtual world. Although there are various navigations and visualizations concepts available, we often still need tools avoiding the typical side effects of immersive virtual environments: the loss of orientation, inadequate navigation, and the cause of fatigue due to indirect viewpoint manipulation. Viewpoint motion control, or *navigation*, is the task of interactive positioning the viewpoint. Since the space physically available in a virtual reality setup is in general limited compared to the size of a virtual world, navigation metaphors have to be used in order to travel through the virtual world. Thus, head tracking alone is not sufficient as a sole navigation tool. This fact, however, implies that an indirect viewpoint manipulation has to be implemented in order to provide navigation that is more convenient. For instance, a vehicle-like navigation control may be used. With this concept, the user maneuvers the vehicle, while his/her actions are applied to translate and rotate the viewpoint in the space.

Indirect viewpoint manipulation often causes fatigue and motion sickness in immersive VR-setups. The main reason for this, despite the latency often introduced by the tracking device, is that the participant views a completely virtual world. The head tracking applies every change of the position and the orientation as a transformation to the virtual camera. Due to the fact, that the entire world is affected by this transformation, the human brain can be fooled in its visual perception. Since there is no physically fixed point in space to orient at and the eyes see something that the other organs (e.g. vestibular organ) cannot confirm, this may lead to motion sickness. The situation becomes even worse when the viewpoint is manipulated indirectly, as this is the case when applying a navigation metaphor.

Therefore, one of the goals of this work was the development of techniques and concepts for overcoming this drawback, while not restricting the interaction and the visualization. As a result, we present in this section the basic concept of the *through-the-lens* metaphor (TTL), as well as set of navigation, remote object manipulation, and visualization tools based on this concept. We also present a taxonomy for the various different states of the through-the-lens-metaphor in general and discuss their application in particular. Some of the cases of the proposed taxonomy are well-known to the VR-developers. However, the entireness of all possible combinations of the states of the virtual worlds when applying the TTL-metaphor has never been investigated before. In this section, we identify and classify a set of known concepts and present a set of new metaphors based on the TTL-concept. The basic idea of the latter is to introduce an additional viewpoint besides the one representing the user in the virtual world, as described in Section 4.2.

4.1 Introduction and Related Work

Since their introduction in 1968 [Sutherland, 1968], VR-setups and applications are widely used in many different areas of the computer science. Some of them have made it even beyond the experimental stage and proved to be valuable tools for research, training, therapy, and tele-working.

In each of these traditional virtual environments, various interaction techniques, based on problem-tailored hardware, have been implemented. The goal of each of them is clearly to provide vivid, powerful and on the other hand easy to understand and to use tools. In this chapter, we will not discuss the entire hardware palette applied for constructing interaction tools, but will concentrate on two-handed interaction applying the hardware presented in Chapter 2, while not restricting the utilization of the proposed tools to it. Beforehand, we briefly review related work on two-handed interaction and navigation in order to compare our tools with other available navigation techniques and to motivate the utilization of two-handed interaction.

4.1.1 Two-Handed Interaction

The first milestone developments allowing the user to feel as comfortable as possible in the way he/she interacts with and within a virtual environment were set by the work of Buxton and Myers [Buxton and Myers, 1986] and Guiard [Guiard, 1988]. Buxton reported on significant performance increase when bimanual navigation or selection is applied compared to accomplishing the task unimanually (as already introduced in Chapter 2). Guiard's work showed how a prop in the non-dominant hand is used to define a (coarse oriented) coordinate system, a kind of reference frame, while the dominant hand is used for fine positioning relative to that coordinate system. Kabbash et al. evaluated two-handed interaction techniques and compared them with unimanual interaction [Kabbash et al., 1994]. The conclusion of their work is that bimanual interaction can significantly improve overall performance, especially when asymmetric partition of labor is possible.

Based on the above observations on the asymmetric use and coordination of human hands, different research groups developed two-handed interaction techniques for different VR-scenarios (e.g. [Cutler et al., 1997]). The resulting tools are known under various names like *pen and palette* [Sachs et al., 1991], *pen and tablet* [Angus and Sowizral, 1995], *physical clipboard* [Stoakley et al., 1995], *3D-Palette* [Billinghurst et al., 1997], *personal interaction panel (Pip)* [Szalavári and Gervautz, 1997, Schmalstieg et al., 1999], and the *virtual palette and remote control panel* [Coquillart and Wesche, 1999].

The idea is quite simple: the user is provided with a tracked pad used as a frame of reference for the interaction with computer generated widgets. Interaction menus and tools are projected on this pad. In addition, a tool for manipulating these interface elements is provided, e.g. a virtual pen. The virtual tool is a visual duplicate of a six-degrees-of-freedom input device, providing tactile feedback, similar to the virtual tricorder introduced by Wloka and Greenfield [Wloka and Greenfield, 1995]. This two-handed interaction concept turned out to be very intuitive to use and suitable for application in various kinds of virtual environments (augmented and immersive, e.g. CAVE-like, tabletop displays, and head mounted displays). Further work in the area of interaction techniques and systems applying bimanual interaction is presented in [Turner et al., 1996, Cutler et al., 1997, Brooks, Jr., 1988].

As described earlier in Section 2.3.2 (p.11), we implemented the concept of two-handed interaction in our system. We applied three tracked devices for determining the position and orientation of the viewpoint and the hands of the participant. A detailed description of the developed tools is given in the remainder of this section.

4.1.2 Navigation

Besides the two-handed interaction, another key issue in a virtual environment is the navigation. This is a problem with increasing importance, since the size of virtual worlds is rapidly growing. Moreover, the adequate navigation is one of the most important features defining the usability and user acceptance of a virtual environment.

Andries van Dam and coauthors [van Dam et al., 2000] divide the navigation into three groups of techniques:

- *Searching* is the motion to a particular location in the virtual environment.
- *Exploration* is defined as navigation without particular target.
- *Maneuvering* is the high-precision adjustment of the user position in order to perform other tasks.

Due to the application of these navigation techniques for performing particular tasks, each of them has a different application range. Searching and exploration techniques are utilized for overcoming large distances, while maneuvering is applied rather locally. Considering these three categories, the research towards developing navigation tools can be divided in two main research directions: (1) adaptation of known and well established approaches and metaphors (like maps, zooming, and driving a vehicle); (2) development of new tools and aids supporting easy to use navigation in VEs.

Fortunately, people are often confronted with the counterpart of the navigation problem in every day life, which facilitates the exploration of this subject. In their work, Darken and Sibert [Darken and Sibert, 1993] presented a toolkit for navigation applying principles from real world navigation aids (e.g. maps). They also compare the strengths and weaknesses of such aids. Stoakley et al. [Stoakley et al., 1995] extended this work to three-dimensional maps. He defines navigation as a term covering two related tasks: movement through a virtual space and determining the orientation relative to the surrounding environment. Considering these two issues, Stoakley introduced the *World-in-Miniature* or WIM-technique. A WIM is a scaled down 3D copy of the virtual world displayed on a hand-held panel. Originally, the WIM was applied for interaction in virtual worlds, i.e. manipulation of objects in space. Pausch et al. [Pausch et al., 1995] extended this approach to provide a navigation tool for accomplishing searching and exploration tasks, enabling the user to directly manipulate the current viewpoint. For this, they utilize a doll representing the user in the miniaturized world. However, they also reported that despite the intuitive application of the WIM, the direct viewpoint manipulation was confusing to many users. This is caused by the fact that the world surrounding the user moves simultaneously with the movement performed on the WIM. In addition, due to the applied miniaturized fixed-size copy of the virtual world, precise manipulation and navigation tasks are difficult to perform.

Another work discussing and comparing navigation tools was presented by Ware and Osborne [Ware and Osborne, 1990]. They describe and evaluate three navigation metaphors: the "flying vehicle control", "eyeball-in-hand", and "scene-in-hand", concluding that: "None of the techniques is judged the best in all situations, rather the different metaphors each have advantages and disadvantages depending on the particular task". Similarly to the WIM-technique, the main problem with the *eyeball-in-hand* and the *scene-in-hand* techniques is that the viewpoint is directly manipulated and the resulting image immediately displayed. This, however, often leads to confusion of the user or may even cause loss of orientation.

A more detailed analysis of navigation considering its basic components: direction selection, velocity selection, and input conditions is discussed in [Bowman et al., 1997]. In their work, Bowman et al. introduce a taxonomy for viewpoint motion in virtual environments. Furthermore, they discuss experiments showing that "pointing"-based travel techniques are advantageous compared to "gaze-directed" steering techniques. In addition, they found out that the instant user teleport is correlated with increased user disorientation. This cognition is closely related to the techniques described in the remainder of this chapter. Instead of teleporting the user, we offer a sort of preview window, through which the location seen through the window can be entered.

4.2 Through-The-Lens Concept

The main idea of a *through-the-lens*-tool (introduced in [Stoev et al., 2001e, Stoev et al., 2000c]) is to provide a viewpoint, additional to the one used to represent the user and thus to display the surrounding scene. Thereafter, the scene as seen from this viewpoint is shown in a dedicated *output window* W_o (as shown in Figures 4.1, 4.2 and 4.4). In other words, we assume that there are two synthetic worlds existing simultaneously in the physical space. The user is surrounded by one of these

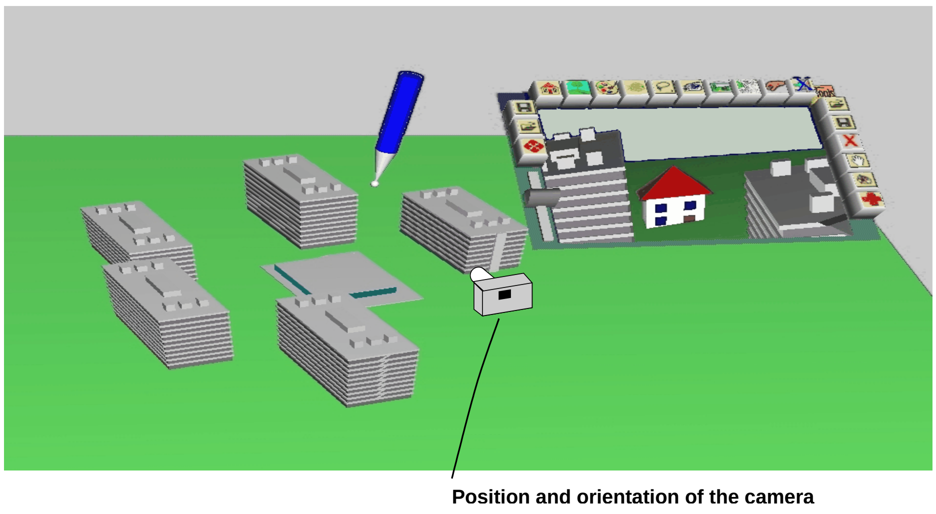


Figure 4.1: The primary world surrounds the user, while the secondary world can be explored only through a window in the primary world (e.g. mapped on the interaction pad).

worlds, called the *primary* world. He/she is represented by the *primary* viewpoint in the primary world. The *secondary world* is the world not visible from the current viewpoint in the primary world. It can be viewed only through a sort of *magic lens* [Bier et al., 1994, Viega et al., 1996] in the primary world, which displays images seen by a virtual camera in the secondary world. The so defined lens acts as a gate to the location seen by the virtual camera in the secondary world. A simple example of this scenario is depicted in Figure 4.1: the window mapped on the interaction pad shows the image of the scene as seen by a virtual camera in the secondary world. In this case, the primary and the secondary scenes consist of the same data. Thus, the position of the camera in the secondary world can be depicted in the primary world. The house visible in the secondary world exists in the primary world as well. However, it is not visible from the current viewpoint in the primary world.

Conceptually, there are two virtual worlds and thus two windows, one in each of the worlds. The window in the primary world, through which the user views the secondary world, we call *output window* (W_o). The virtual counterpart of this window in the secondary world we call *viewing window* (W_v). Although in practice this is the same window, since both are always attached to each other, for clarity we use these two different terms throughout this section, depending on the world we refer to.

4.2.1 Taxonomy for the States of the Two Worlds

Regarding the relation of the two worlds to each other and the windows connecting them, we distinguish different states, as depicted in Figure 4.3. Let us first consider the output window W_o, hence, the window in the primary world. W_o can have three different states in the primary world (as shown in Figure 4.2):

- (case O1) fixed in the space explored by the user (primary world);
- (case O2) fixed in the image plane of the user;
- (case O3) mapped onto a pad that is held in the non-dominant hand of the user.

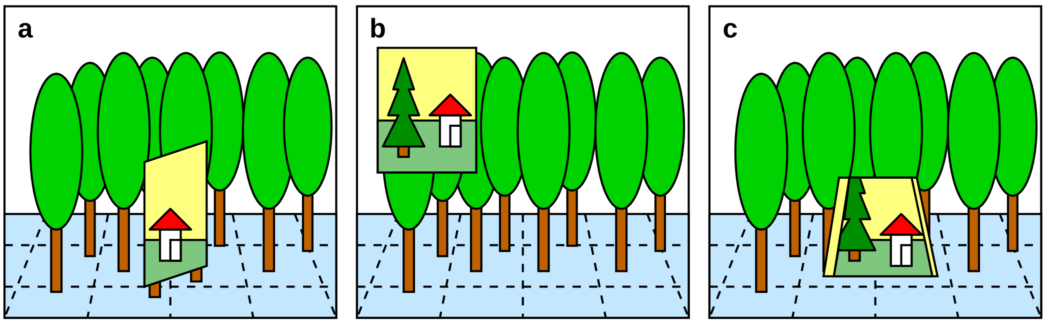


Figure 4.2: Figures (a)-(c) display the states O1-O3 respectively.

In the first case O1, the window is only visible when viewed from the right direction. Since it is fixed in the primary world, the user cannot move it. Changes of the user position in the primary world allow viewing the virtual world behind the window from different angles. This window can be thought of as a real window placed somewhere in the primary space explored by the user.

In the second case O2, the window is fixed within the image plane of the viewer. This means, that when the user moves his/her viewpoint, the window remains at the same position in the image plane. In other words, this scenario can be compared with a sort of glasses worn by the user. When the view direction is changed, the position of the glasses remains unchanged relative to the view frustum.

Finally, for the realization of the last case O3, we utilized the *Personal Interaction Panel* (*PIP*) concept [Szalavári and Gervautz, 1997, Schmalstieg et al., 1999]. The PIP consists of a tracked palette, on which the virtual tools are displayed in such a way, that the user sees them on the pad's surface. This is accomplished either using back-projection and a transparent palette (Virtual Table, Virtual Workbench, Cave, Powerwall, etc.) or front-projection (opaque and transparent head-mounted-displays). In either case, the window is mapped onto the interaction pad (see Figure 4.1). In contrast to the first two scenarios, where the window W_o is fixed either with respect to the explored primary world, or with respect of the viewing frustum, in this case the pad, thus the window mapped on it, can be freely moved within the primary space. This is usually performed with the user's non-dominant hand.

These were the possible states of the output window in the primary world. Regarding the additional viewpoint and the scene seen through it, there are also three conceptually different states of the *viewing window* W_v and the secondary world seen through it:

- (case V1) the secondary world is fixed in the primary world's space,
- (case V2) the secondary world is fixed with respect to the viewing window;

State of W_O in the primary world	**State of the secondary world**
O1 - fixed in the user space (primary world)	**V1 - fixed in the space (secondary world)**
O2 - fixed in the image plane	**V2 - fixed with respect to W_V**
O3 - freely movable in the space (on interaction pad)	**V3 - fixed with respect to the primary viewpoint**

Figure 4.3: Summary of the possible states of the windows in both spaces: the primary and the secondary world.

- (case V3) the secondary world is fixed with respect to the primary viewpoint.

In the first case V1, the coordinate systems of the two worlds are fixed with respect to each other. The window connecting them can be positioned arbitrarily in the primary space. Depending on the position of the window, different areas of the secondary world are visible through it. This is the only scenario, in which the primary viewpoint is coupled with the secondary viewpoint. Thus, the motion of the primary viewpoint in the primary world causes the motion of the secondary viewpoint in the secondary world. This is the only natural way of entering the secondary world as shown in Section 4.2.2, p. 56.

In contrast, in the second scenario (V2), the secondary world is fixed in the output window's coordinate space. This means, that independent of the position of the output window, the viewing window remains fixed in the coordinate space of the secondary world. Thus, looking at the output window from different viewing angles enables exploration of different areas behind it. This is due to the fact that changing the position of the primary viewpoint with respect to the output window in the primary world (or vice versa), causes the scene seen through the viewing window to change its position with respect to the secondary viewpoint. On the other hand, the motion of the primary viewpoint in the primary world does not cause a motion of the secondary viewpoint in the secondary world, since the secondary world is always fixed with respect to the viewing window.

Finally, in the third case V3, the secondary world is fixed with respect to the primary viewpoint in the primary world. In other words, independent of the *position* and the *orientation* of the primary viewpoint in the primary world, the area of the secondary world seen through the window remains the same if the position of the output window in the viewing frustum does not change. Should the position of the output window change, different areas behind the output window can be explored similar to the case V1.

4.2.1.1 W_o Fixed in the Primary World (O1)

When the output window W_o is fixed in the primary space (case O1), the secondary world and the primary world are fixed with respect to each other. Thus, we cannot distinguish between the cases V1 and V2. This scenario was first described in [Schmalstieg and Schaufler, 1999], where the window is used for sewing two different virtual worlds together. Once this is done, the user can travel from one world to the other moving the viewpoint through the provided window. Unfortunately, the authors elaborate only on this single case and its application, not discussing further states of the primary and the secondary world.

On the other hand, in case (O1/V3) the secondary world moves with the user motion in the primary world. This means, that the position of the secondary viewpoint in the secondary world remains unchanged, when the primary viewpoint is moved in the primary world. Only the clipped part of the secondary world may change. Thus, when the user moves in a particular direction, different parts of the secondary world can be examined with the output window W_o, which is static in the primary world. Even though we can identify this constellation of the primary and the secondary world and the windows connecting them, it does not make much sense and is not been reported in the literature.

4.2.1.2 W_o Fixed in the Image Plane (O2)

When W_o is fixed in the viewing frustum of the user (case O2), two cases O2/V1 and O2/V23 are theoretically possible. Since the output window is fixed with respect to the primary viewpoint, the cases O2/V2 and O2/V3 are the same, since the position of the output window does not change with respect to the primary viewpoint.

In the first case (O2/V1), when the primary viewpoint is moved in the primary world, the output window and thus the viewing window move with the image plane and the user sees different parts of the secondary world. This corresponds to moving the viewpoint in both worlds simultaneously. Since both worlds are fixed with respect to each other, we achieve the effect described above.

This scenario is often applied in semi-transparent head mounted display systems, where the primary world is the physical world surrounding the user. The secondary world seen through the HMD is a virtual world, allowing for superimposing information aligned with the primary world, which does not exist or is not visible in the primary world (e.g. pipes or conduction in the wall).

The second case (O2/V23) is not widely used, since a secondary world fixed with respect to the viewing window would result in displaying always the same image independent of the viewing direction and position of the viewer in the space (primary world). This is the same as attaching the secondary world to the primary viewpoint, since the window is fixed in the image plane. The only scenario in which this feature may be useful is when the user intends to “keep an eye” on a given location in the secondary world, regardless the position/motion in the primary world. In this case, the state O2/V23 allows viewing the same image of the secondary world at a fixed position in the viewing frustum.

4.2.1.3 W_o Mapped on the Pad (O3)

Until now, we considered cases in which the output window was fixed either with respect to the primary world or in the image plane. The most interesting case, however, is case O3. In this scenario, we are able to interactively position the output window in the primary world and thus the viewing window in the secondary world. Let us now consider the different possible applications of the different state combinations.

In case O3/V1, the output window W_o mapped on the pad is used to explore parts of the secondary world that is fixed in the primary world’s space (see Figure 4.4(c) and (d)). Moving the pad with the output window W_o in the primary world causes

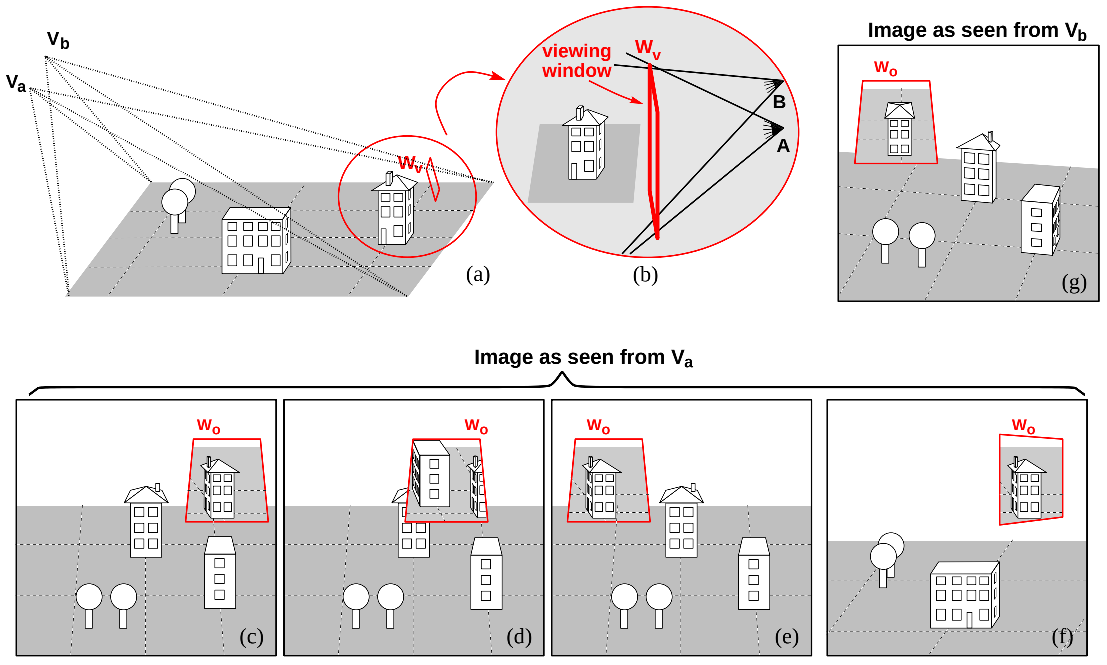


Figure 4.4: The position of the viewing window (W_v) is shown with respect to the scene seen through it (a). V_a and V_b are two different viewing positions. (b) shows the two viewing positions A and B, derived from the current camera positions V_a and V_b. In case the viewing window W_v is fixed in the secondary scene and the output window W_o is moved in the primary scene, the secondary scene moves with the viewing window as shown in (c) and (e). (d) illustrates the "scene fixed in space" scenario (O3/V1) with the secondary scene frozen in the space of the primary world (compare with (c)). Moving W_o (compare (c) and (d)) allows viewing different parts of the scene. Detaching the secondary viewpoint from the primary, allows the user to travel the primary world, while staying at the same position in the secondary world (compare (c) and (f)). When the viewpoint changes (e.g. from V_a to V_b), the scene shown in W_o can be viewed from different angles as depicted in (e) and (g).

the motion of the viewing window W_v in the secondary world's coordinate space. Thus, a magic lens-like tool [Bier et al., 1994, Viega et al., 1996] is realized.

In contrast, in the second scenario (O3/V2) the window can be adjusted to show a given part of the secondary scene (location of interest), such that even if the output window W_o is moved, the virtual window W_v remains fixed in the secondary world's space (see Figure 4.4 (c) and (e)). In this way, an "eye is kept" on a target location in the secondary world independent of the user's motion in the primary space (similar to cases O2/V23, however with freely adjustable position of W_o). In addition, in this scenario the user still can view the world behind the window from different angles and thus explore different areas of it.

Finally, case O3/V3 makes it possible to travel the primary world, without applying the changes of the primary viewpoint to the secondary viewpoint in the secondary world. This is similar to the case where the secondary world is anchored to the viewing window (case O3/V2). However, unlike in case O3/V2, moving the output window in the primary world enables exploration of different parts of the secondary world, while looking at the window from different angles does not enable exploration of different areas in the secondary world. This is an analogy to case O3/V1, in which the output window is also used for exploring the secondary world. State O3/V3 is especially useful when the user travels the primary world and wants to keep the position of the secondary viewpoint unchanged in the secondary world. If the user position does not move and only the window W_o is moved in the primary space, the scenario is equal to case O3/V1, hence to the case where the secondary world is fixed with respect to the primary world.

4.2.2 Entering the Secondary World

As introduced above, the proposed concept of providing an additional viewpoint for an explored scene is a more generalized 3D magic lens [Bier et al., 1993, Viega et al., 1996] and extends the *seam* concept (seam stands for spatially extended anchor mechanism) described in [Schmalstieg and Schaufler, 1999] (therefore *through-the-lens*-tools). In the original *seam* implementation, the output window W_o is fixed in the primary world and the viewing window W_v is fixed in the secondary world (case O1/V1). In contrast to this, in our scenario both windows can be freely moved in both spaces.

When we set the primary and the secondary world to be the same, this corresponds to introducing two viewpoints to one scene: a primary and a secondary viewpoint, as this is the case in Figure 4.4. In this way, we provide a dynamic viewpoint for exploring distant locations in one and the same virtual world. Thus, an interactively defined *wormhole* is introduced, through which a location is viewed from the secondary viewpoint. Due to the flexible positioning of the output window attached to the interaction pad, the remote location can be viewed from any arbitrary location in the primary world. In other words, the user travels through the primary world with the pad held in the non-dominant hand, hence, with the window to the secondary world attached to it.

In addition, the proposed concept can be used not only to view the scene from a different viewpoint, but also to immerse the secondary scene through this viewpoint. In order to enter the secondary world as seen from the additional viewpoint, the user has to move the pad towards her/his face until the image on the pad completely covers the viewing area. Once this is done, the system automatically detects this state and sets the remote viewpoint v_r to be the current viewpoint $v_p \leftarrow v_r$. The old current viewpoint v'_p is saved as an icon for the case the user intends to return to the previous location. In this case, the same procedure is performed, namely the old viewpoint v'_p is entered through the window mapped on the interaction pad.

In the above discussion, we assumed that when the secondary world is entered, it is fixed with respect to the primary world. In other words, moving the output window W_o in the primary world corresponds to moving a magic lens (the viewing window W_v) and exploring the distant location (case O3/V1). This is the natural way to enter a new location in the way described above, since the final state of the system is a virtual space that is fixed with respect to the former primary world. Theoretically, it is possible to enter a world fixed with respect to the output window W_o (case O3/V2), however, in this scenario the remote viewpoint v_r depends on the position and orientation of W_o. Hence, a state change has to be performed: the world is switched from window-fixed to fixed in the primary space, as the target state is always virtual secondary world fixed in the primary space.

Finally, in case O3/V3 the secondary world cannot be entered, since the secondary viewpoint is detached from the primary viewpoint and the motion of the primary viewpoint is not transferred to the secondary viewpoint. Hence, the secondary viewpoint is always fixed in the secondary world. In order to enable entering of the secondary world in this scenario, a state change has to be performed similar to case O3/V2.

This way of entering the secondary world through the window mapped on the interaction pad, helps to avoid the application of the "teleporting" mechanism that was found out to be often confusing to the user [Bowman et al., 1997]. Furthermore, we provide a vivid preview of the remote location, which can be remotely explored before it is "smoothly" entered.

4.2.3 TTL Requirements

The hardware realization of the proposed tools requires a virtual environment setup with three tracked devices: one for the user's head position and orientation and one for each hand. This is the only information we need in order to realize the described concept. Moreover, the application of the TTL-idea can be realized in any type of immersive VR-setup (Virtual Table, Virtual Workbench, Cave, Powerwall, as well as opaque head mounted displays), in which the systems provide this tracking capability.

If the virtual world is organized in a scene graph-like structure, the software realization is simple as well. In order to display the virtual world as seen from an additional viewpoint v_r, the entire scene S, preceded by an appropriate transformation (T_a), passes the rendering pipeline once again (see Figure 4.5). The projection of the

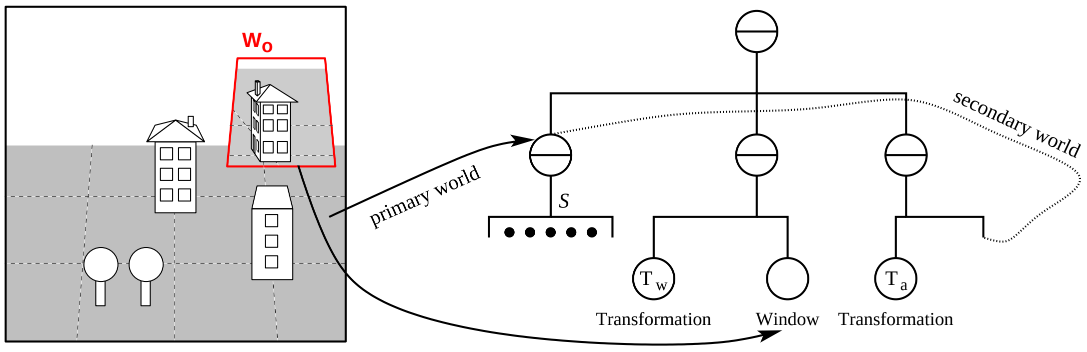


Figure 4.5: The transformation T_w positions the window. T_a is responsible for the position of the secondary world.

output window (e.g. mapped on the pad) defines the area in which the rendering of S as seen from v_r is performed. During drawing, the virtual counterpart of the physical pad is applied to fill the OpenGL's stencil buffer with an appropriate mask. Afterwards, S is displayed only within the masked area as seen from v_r, which is the same as transforming the scene with T_a before it is rendered. Further implementation details are discussed in [Schmalstieg and Schaufler, 1999].

4.3 Through-The-Lens Navigation Tools

After introducing the different states of the output and viewing windows within the primary and the secondary world, here we address the adjustment of the secondary world in such a way that a particular target location can be viewed through the output window W_o. In other words, we discuss how the secondary viewpoint v_r is positioned in the secondary world such that given location is within the viewing frustum. Once this task is accomplished, the new location can be entered as described in Section 4.2.2, thus providing navigation capabilities.

Various navigation techniques belonging to one or more of the navigation types introduced in Section 4.1.2 are reported in the literature. The navigation tools we present in this work are inspired by the *eyeball-in-hand*, *scene-in-hand* (we call *grab-and-drag*) [Ware and Osborne, 1990], and WIM-techniques [Pausch et al., 1995], but attempt to overcome their limitations. We introduced to these tools the above *through-the-lens* concept, extending their functionality and improving their usability. In particular, we apply the manipulation described in the original techniques to the secondary viewpoint. Hence, the effect of the manipulation is observed through the window, rather than applying direct transformation of the primary viewpoint. In this way, the presented navigation aids provide a set of flexible and powerful tools, covering all of the navigation categories introduced in Section 4.1.2.

In all through-the-lens tools described next, the primary and the secondary worlds are the same. Hence, the viewpoints are in the same synthetic world. Depending on the manipulation of the secondary viewpoint, we distinguish two different categories of techniques: applying direct viewpoint manipulation (e.g. *eyeball-in-hand*) or indirect manipulation (e.g. TTL-WIM and TTL-*grab-and-drag*). Note that this manipulation is always applied to the secondary viewpoint. Thus, the disadvantages of the indirect primary viewpoint manipulation are circumvented. In this section, we first describe the indirect manipulation techniques, which allow intuitive adjusting the scene seen through the second viewpoint without having to explicitly define a virtual camera position and view direction. Afterwards, we discuss in detail the *eyeball-in-hand*-technique for direct viewpoint manipulation.

4.3.1 TTL Scene-In-Hand Concept

This technique was first presented in [Ware and Osborne, 1990]. It provides a handle attached to the scene, where the translations and rotations of the handle are applied one-to-one to the scene. It is akin to have such direct linkage of the scene and a tracked device. It is easy to understand and apply even for motions extending the hand's reach: a button is used to attach and release the scene to/from the virtual handle. This approach has shown to be useful for manipulating discrete objects and changing the viewpoint of the user for scene exploration [Ware and Jessome, 1988, Mapes and Moshell, 1995].

In our implementation, we start with two aligned viewpoints, which correspond to two aligned (primary/secondary) synthetic worlds. Now, the user is able to manipulate the scene seen from the secondary viewpoint by grabbing a point of it (e.g. an object or event the air) and dragging it in the desired direction (Figure 4.6). Note that the secondary scene remains always fixed with respect to the primary world (case O3/V1), except when it is grabbed. In this scenario, the secondary scene can be grabbed at any arbitrary location, using the second button of the interaction pen. Once the button is pressed, the secondary world is attached to the interaction pen.

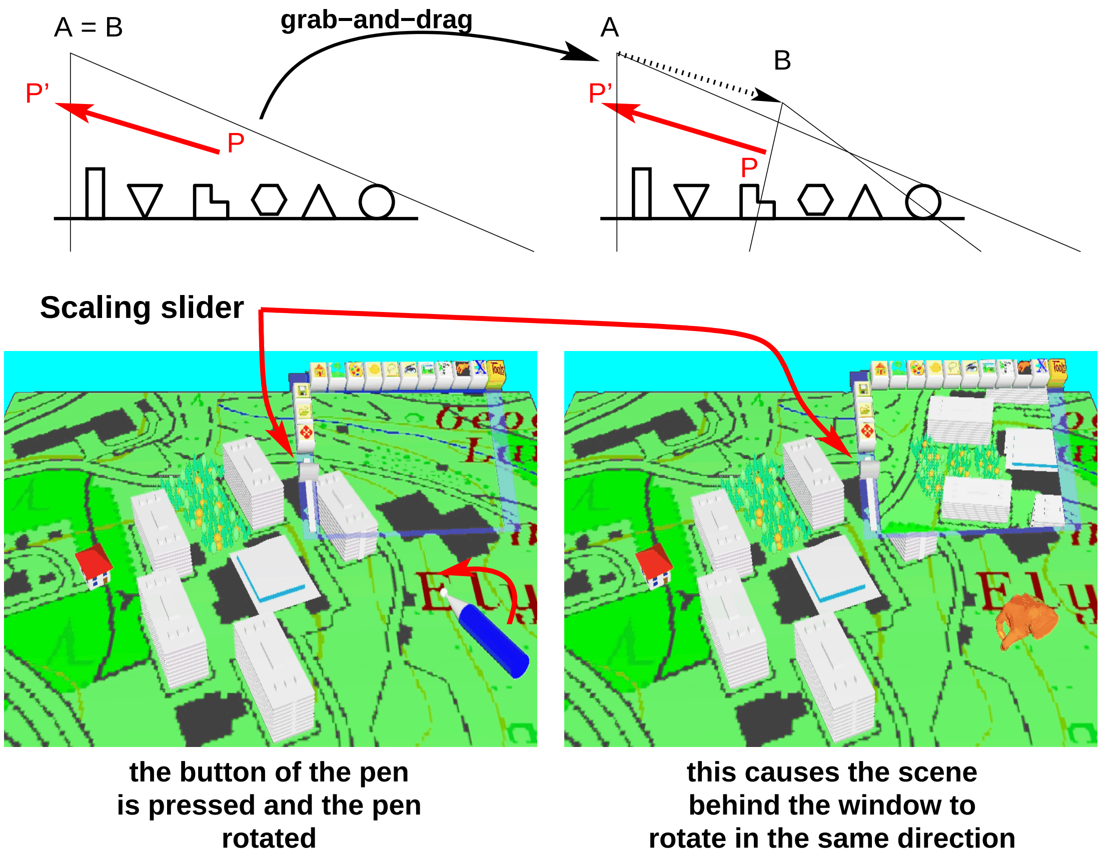

A = B
grab–and–drag
A
B
P'
P
P'
P
Scaling slider
the button of the pen is pressed and the pen rotated
this causes the scene behind the window to rotate in the same direction

Figure 4.6: Initially, both viewpoints are aligned as shown on the left. Grabbing the scene at point P and dragging to point P' corresponds to a translation (eventually combined with rotation) of viewpoint A to viewpoint B, as shown on the right.

Moving and rotating the pen corresponds to moving and rotating the secondary world. In contrast to the original implementation [Ware and Osborne, 1990], we did not fix the center of rotation to the center of the scene, since, as the authors point out, rotations are difficult to perform when the viewpoint is far from the fixed center of rotation. Instead, the rotation is mapped one-to-one to the secondary world.

This approach is similar to the scaled-world grab locomotion metaphor described by Mine et al. [Mine et al., 1997]. They propose a technique for grabbing distant objects using a form of image-plane interaction. Thus, the user can pull him/herself towards any visible object. Unlike the scaled-world grab where the authors apply the motion immediately, we (a) provide a preview window, (b) use a direct technique for grabbing and dragging the *secondary world*, and (c) do not require from the user to grab an object, instead any point in the space can be grabbed. The scene seen through the output window is now manipulated applying a simple grab-and-drag handle. In this way, the viewed part of the scene can be chosen very precisely, without requiring the user to physically or virtually walk/fly to the location of interest.

In addition to the grab-and-drag mechanism, the user can also scale the secondary scene if needed (see slider in Figure 4.6), hence, making this tool especially suitable for final high-precision adjustment. Furthermore, the scaling facilitates the traveling of large distances. When the user intends to view a distant location he/she can scale down the secondary world, place the target location underneath the center of the output window using the grab-and-drag mechanism and scale the secondary world up again. With this additional aid, the application range of this technique is not limited to the reach of the user's hand. Nevertheless, when applied for viewing distant locations, the proposed technique may be circumstantial. This drawback can be overcome by combining our through-the-lens technique with other techniques for remote object grabbing (e.g. go-go technique [Poupyrev et al., 1996], image plane techniques [Pierce et al., 1997], or scaled-world grab technique [Mine et al., 1997]).

Similar to the original eyeball-in-hand tool, we implemented a one-to-one mapping for the translation. Moreover, we also implemented a one-to-one mapping of the rotation, instead of allowing rotations only about the center of the (secondary) scene. In this way, we can easily compute the transformation T_a (see Figure 4.5, p. 57). For T_a, we only need the position and orientation of the dragging handle at the beginning of the dragging motion and the current position of the handle. If another grabbing technique is applied, the transformation T_a has to be generated properly.

4.3.2 TTL World-In-Miniature

Originally, the WIM metaphor was applied for remote object manipulation [Stoakley et al., 1995]. The miniaturized copy of the world is mapped onto a hand-held device. Now the manipulation of the icons of the objects in the miniaturized world causes the corresponding objects to move in the surrounding world. Pausch et al. [Pausch et al., 1995] extended this concept to traveling in immersive environments. They found out, that the direct mapping of the manipulated user viewpoint icon in the miniaturized world to the full-scale virtual world causes disorientation. Furthermore, they state that the user's attention is focused on the miniature and not in the full-scale virtual world. In order to circumvent these drawbacks, the authors propose a slow-in-slow-out animation [Lasseter, 1987] of the viewpoint given the current user position in the full-scale world and the target position set in the miniature. This, however, is still confusing to the user, as the authors note. Instead, they propose the user to become a doll. This means, that the doll representing the user is animated into the miniature. Afterwards, the user is animated to the new location and the graphics forming the old full-scale virtual world are faded out and a new WIM is presented to the user.

In contrast to the original WIM tool, with the TTL-WIM we do not map the miniature copy of the virtual world *on top* of the pad. Instead, we display the latter underneath the pad's surface. In this way, we create the impression of looking into the miniaturized virtual world through a window defined by the pad on top of an imaginary box. Instead of explicitly defining the final position of the user in the miniaturized world, the user interactively selects a region of interest dragging a box around it. This approach is well-known from the traditional two-dimensional desktop application. The main differences are that (a) the user views a three-dimensional world and (b) the miniaturized world is mapped onto a hand-held interaction pad. The selection is made on the top of the bounding box of the virtual world. Afterwards, the miniaturized world is scaled up in such a way, that the selection fills up the viewing window and the top of the secondary scene's bounding box is still aligned with the surface of the interaction pad (as shown in Figure 4.7). Due to the fact that the miniaturized world is displayed on top of the interaction pad in the original scenario, the free choosing of a scale factor is not possible.

With the proposed technique, the selected part can be examined not only

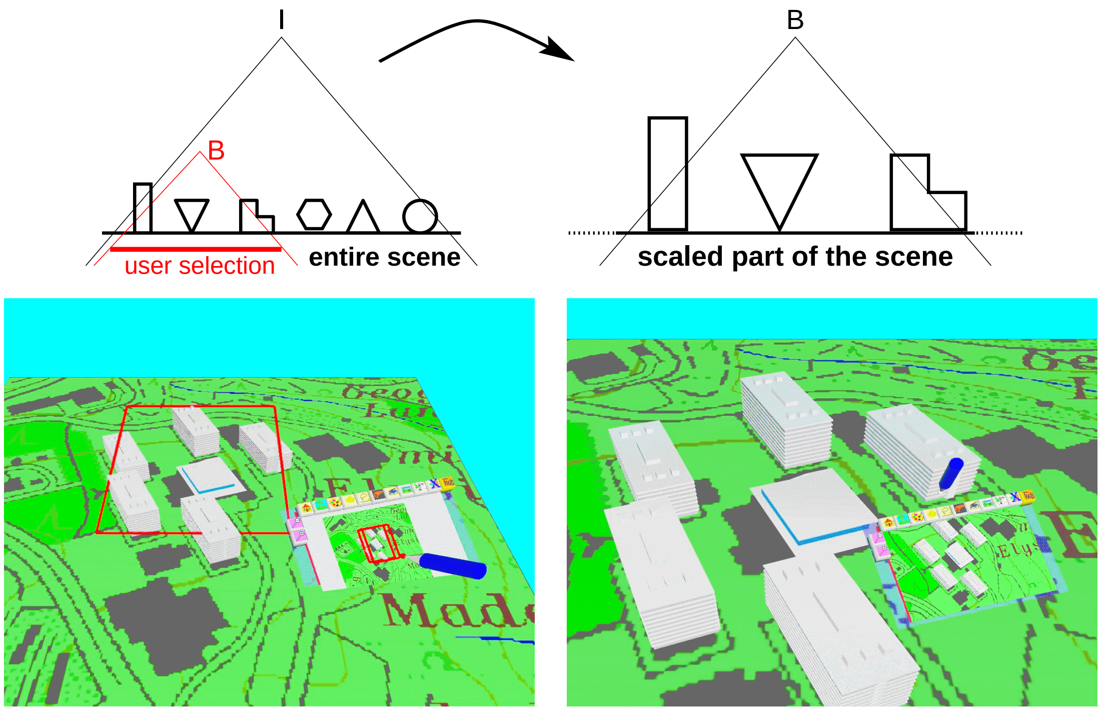


Figure 4.7: Initially, a miniaturized copy of the entire scene as seen from viewpoint I is displayed on the interaction pad (do not confuse with the current viewpoint A). During the interactive selection of a region of interest, the selection is shown in the primary world as well. After completing the selection, the viewpoint is moved to B, such that only the selected region is visible through the pad. In addition, the lower right image shows the transformation applied to the current viewpoint in the primary world.

through the pad, but also in the virtual world surrounding the user (if visible from the current viewpoint). The size of the dragged box defines the position of the virtual camera in the secondary world, while the orientation always remains orthogonal to the virtual world (z-axis of the world) as shown in Figure 4.7. In this way, the viewing window W_v is always fixed in the secondary scene. Thus, this scenario corresponds to case O3V2 (see Figure 4.3, p. 53), whereas an indirect manipulation of the secondary viewpoint is applied.

This technique is primarily used for coarse selection of the viewed area. It is especially suitable for subsequent dragging a box around the target area at different scale levels. Once the user has adjusted the desired part of the scene to be seen through the output window on the pad, there are two ways of entering the new location:

(a) Either the user is automatically teleported to the new location, whereas the view orientation relative to the surrounding world remains unchanged. Only the primary world is appropriately scaled taking into account the center of the interactive selection (see Figure 4.7), or

(b) The secondary scene behind the window is released from the window and can be further adjusted applying another through-the-lens technique, or directly entered as usual (see Section 4.2.2, p. 56).

In order to immerse the world seen through the output window on the pad, this world has to be fixed in space (state V1). Since this is not the case with this tool, after selecting the location of interest the scene has to be explicitly fixed in the space, which corresponds to releasing it from the viewing window it was attached to.

We can easily determine the scale factor and the point of interest (the middle of the dragged box). Thus, the computation of T_a is simple as well. Knowing the size and location of the dragged box, the camera in the secondary world is positioned right over the point of interest in order to display the new miniaturized world under the pad's surface, as shown in Figure 4.7. When this tool is used to automatically scale the underlying terrain, the appropriate transformation (derived from T_a) is applied to the primary world as well.

4.3.3 TTL Eyeball-In-Hand Concept

This technique has been introduced and explored by several researchers (e.g. [Badler et al., 1986, Brooks, Jr., 1988, Ware and Osborne, 1990]). The eyeball-in-hand originally uses a tracked device as a virtual camera that can be moved about the virtual scene. The scene acts like an invisible model surrounding the user that can be explored only through the hand-held camera. Thus, the participant sees on the screen what the camera sees through its lens.

With this implementation, however, the boundaries of the scene have to be located inside the boundaries of the device's domain. In other words, in order to move the camera around the scene, the user has to physically walk. This is due to the fact, that the camera can only be positioned within the hand's reach. Therefore, Ware and Osborne [Ware and Osborne, 1990] presented an extension enabling to travel through the scene using a button for activating the display of the camera view. In this way, they implemented a form of grabbing: when the button is pressed the camera view is tracked, otherwise, the camera is detached from the tracked device. A side effect of this feature, however, is that the mental model of a fixed invisible scene surrounding the user is destroyed. Instead, the user must imagine that the invisible scene can be grabbed using the camera as a kind of handle. In practice, this model turned out to be confusing to the user and was not discussed further by Ware and Osborne [Ware and Osborne, 1990].

Despite the intuitive mental model applied with this metaphor, the main problem is the often-caused disorientation. This is due to the motion of the virtual viewpoint, which is directly applied to the current camera. Moreover, the one-to-one mapping of the hand to the virtual viewpoint makes precise adjustment of the virtual camera very hard. Even though, the eyeball-in-hand metaphor is simple to understand and requires a simple mental model of the scene, the above limitations make it unsuitable as a sole navigation technique. In order to circumvent these limitations, while still supporting the features of this metaphor, we introduced a preview window (TTL) to the eyeball-in-technique. This makes it possible to view the scene from various viewing positions (in the hand's reach) without changing the current viewpoint of the user in the primary world, thus, reducing confusion and disorientation.

In particular, the pen held in the dominant hand is used to define the secondary viewpoint in the surrounding virtual environment (see Figure 4.8). It defines the position and the orientation of the secondary virtual camera. The scene, seen from the secondary viewpoint, is displayed in the output window, which is mapped on the pad held in the non-dominant hand. In this scenario, the secondary world is always in state O3V2, thus it is fixed with respect to the output window. Since the user sees the position of the virtual camera in the primary world (surrounding environment) and the scene as seen by the positioned camera *simultaneously*, the virtual camera

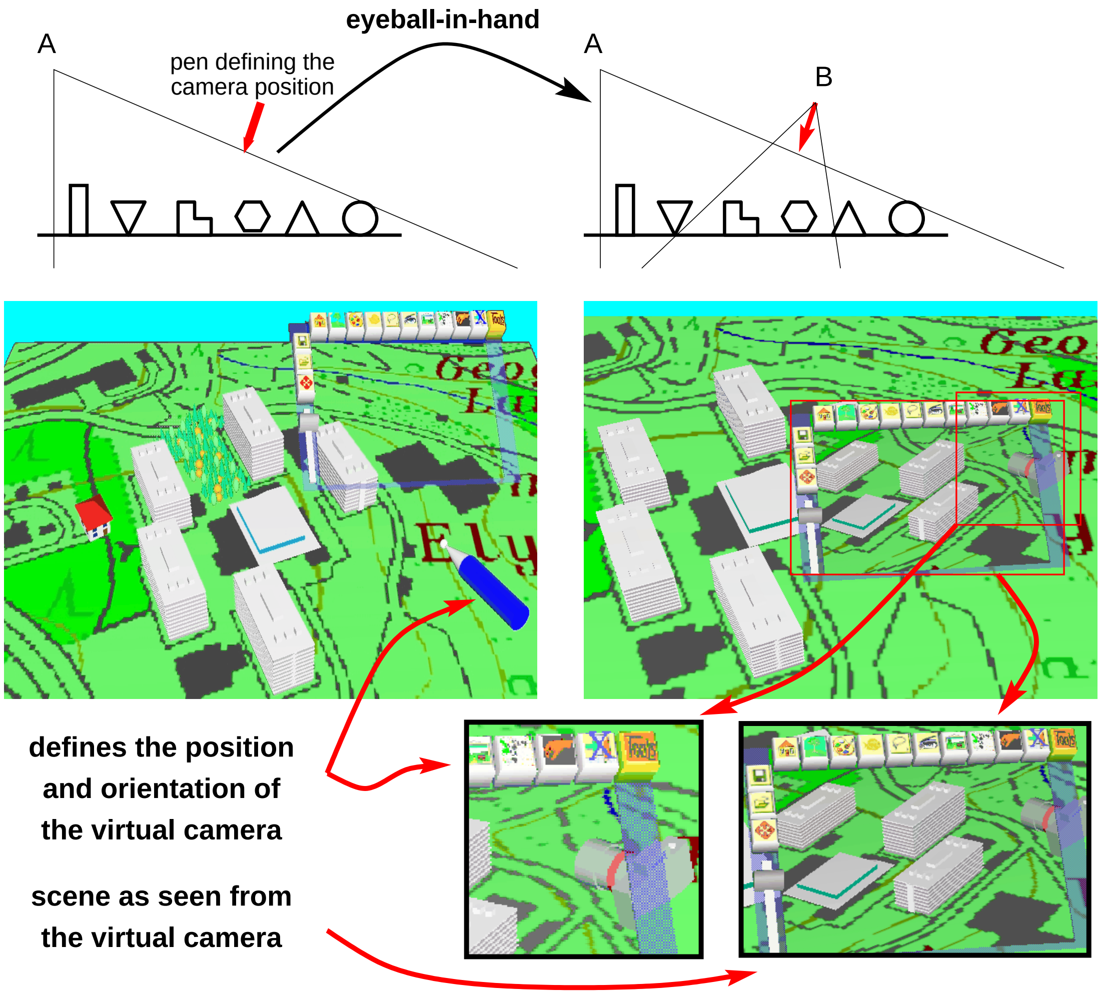

eyeball-in-hand
A
pen defining the camera position
A
B
defines the position and orientation of the virtual camera
scene as seen from the virtual camera

Figure 4.8: Applying the eyeball-in-hand tool, the secondary viewpoint can be positioned directly by defining a position and orientation of the new virtual camera B. A is the current viewpoint.

can be positioned very precisely. In this way, our tool overcomes the limitations of the original eyeball-in-hand metaphor, while still supporting its features.

In order to display the appropriate part of the scene, we first compute the transformation T transforming the current position and orientation of the pen to the current camera's position and orientation. The matrix T^{-1} is now used to compute the transformation T_a. This is set to be the transformation preceding the scene drawn in the output window (see Figure 4.5, p. 57).

4.3.4 Usability

Event though we have not performed detailed quantitative usability studies yet, preliminary qualitative evaluation of interactive sessions with a virtual world assembly application have shown that the TTL grab-and-drag and the TTL WIM tools are intuitive and do not require training time in order to apply them appropriately. The eyeball-in-hand tool, however, turned out to be confusing for some users due to the 6DOF manipulation (see Table 4.1). This conclusion was reached when a group of 50 users was provided the opportunity to work with the TTL-tools. The participants' experience in interaction in virtual environments reached from novice users with no experience to expert users in this area.

Each of the users was first shown a short demonstration of how the principle interaction works. This was primarily geared towards explaining the interaction pen and tabled paradigm. This was followed by a short introduction to each of the proposed techniques. Finally, we let the users work with the application on their own. We did not ask the participants to complete some questionnaires, instead, we observed their reaction and how fast and how accurate the techniques are applied.

4.3.5 Conclusion

One of the most important features of the proposed through-the-lens concept is that the surrounding virtual world remains unchanged during the two-handed manipulation of the secondary viewpoint. This strategy prevents from causing motion sickness and loss of orientation, which are the main problems of the original navigation tools.

Although each of the proposed techniques has some limitations, the combination of all of them provides a powerful toolkit for exploring distant locations in a virtual world, as well as navigating in the virtual environment (see Table 4.1). The described through-the-lens world-in-miniature technique overcomes the main drawbacks of the original WIM-metaphor. The arbitrarily adjustable scale factor allows the user to view large virtual worlds at a freely chosen scale, which is not supported by the original WIM. Since the miniaturized world is not displayed *on top* of the hand-held palette, the scale size is not limited to the distance between the user's head and the projection pad. The presented through-the-lens WIM is primarily used for accomplishing searching and (coarse-level) exploration tasks.

Technique	**Features**	**Limitations**
TTL grab-and-drag	• suitable for searching tasks, and precise final adjustment tasks • intuitive viewpoint manipulation	• circumstantial for distant objects and locations
TTL WIM	• suitable for exploration and searching tasks • supports multiple scale levels	• scene cannot be entered until not "fixed in space" • improper for fine viewpoint manipulations
TTL eyeball-in-hand	• requires very simple mental model • easy to use for fine precision camera adjustment	• unsuitable for exploration • may be confusing (too many degrees of freedom)

Table 4.1: Brief comparison of the proposed navigation tools.

In contrast, the grab-and-drag technique can be applied for all of the navigation categories introduced in Section 4.1.2, namely: exploration, searching, and maneuvering. The only limitation is that it is not easily used for setting the viewpoint to very distant locations. However, there are various alternative ways for overcoming this obstacle. The most intuitive one is to use the slider for scaling up and down the secondary world, as proposed in Section 4.3.1, p. 58. Another alternative is the combination of the grab-and-drag tool with other techniques for remote grabbing.

The eyeball-in-hand tool is mainly used to directly define the view position and orientation within the reach of the user's hand. Hence, it falls into the category of the maneuvering-tools. This technique is very intuitive and requires a simple mental model for the correct adjustment of the secondary viewpoint. With the proposed preview window, this tool provides a valuable navigation aid, while circumventing the limitations of the original eyeball-in-hand tool.

In summary, the application of the through-the-lens concept for navigation in virtual environments provides a powerful mechanism for implementing preview-enriched navigation tools. This concept allows exploring distant locations or hidden features of the virtual world surrounding the user, without having to virtually or physically fly/walk to the remote location. It allows viewing locations of interest, while still being at the same location in the primary virtual world. This main contribution of this section enables the enhancement of existing navigation aids and development of new tools exploiting the through-the-lens concept.

4.4 Through-The-Lens Remote Manipulation Tools

The through-the-lens metaphor described in Section 4.2, is not only suitable for utilization as a preview window during navigation, but it can be even extended and applied as a remote object manipulation technique. In this case, the TTL-concept provides an intuitive and powerful prop for moving objects from a remote location to the user and vice versa, as well as direct manipulation of objects at the remote location [Stoev et al., 2001d, Stoev et al., 2002]. Similar to the through-the-lens navigation tools, in this scenario the primary and the secondary worlds are the same. This is an important condition, since otherwise the objects manipulated through the output window would not be in the same scene. Consequently, the effect of manipulating and observing the changes in the surrounding scene would not be achieved.

4.4.1 Introduction and Related Work

Even though it does not have a counterpart in real life, remote object manipulation is both useful and intuitive. It allows the user to work with objects not within the reach of his/her hand and to examine the virtual world as seen from the current viewing position. Without the ability to perform remote object manipulation, the user would have to navigate to the location of interest, apply the intended manipulation and travel back in order to examine the results of the manipulation.

Many researchers have addressed the subject of remote object manipulation in virtual environments. Pierce et al. [Pierce et al., 1999] presented the *Voodoo Dolls*-technique for remote object manipulation. A doll looks like a minified (or magnified) copy of an object. The user creates a doll by framing an object with her/his hand on the image plane and pinching his/her fingers together. The system then instantaneously creates a copy of the object, scales it (up or down) so that the new doll reaches a comfortable working size, and moves the object to the user's hand. Afterwards, depending on which hand holds the doll, the object associated with it can be manipulated in various ways: either the user changes objects features, when the object is held in the non-dominant hand, or the position and orientation of the original object can be adjusted, when it is held in the dominant hand.

Mine et al. [Mine et al., 1997] presented another approach for remote object manipulation: the scaled-world grab. The basic idea of this technique is to automatically scale objects in such a way, that their projected size remains unchanged, while bringing them close to the user. He/she can now manipulate them as if they were in the hand's reach. After the manipulation is completed and the object released, it is scaled back to its original location.

Poupyrev et al. [Poupyrev et al., 1996] described the go-go mechanism for non-linearly extending the arm of the user, thus, enabling manipulation of objects out of the reach of the user's physical hand. This metaphor provides the user with the traditional one-to-one mapping of the translation of the tracked device to the virtual hand within given application radius. Outside this area, the mapping extends the virtual hand applying a quadratic increase of the arm extension. This results in a tool extending the physical reach of the user, however, with still limited application radius.

Bowman and Hodges [Bowman and Hodges, 1997] gave a brief evaluation of this and other existing techniques for grabbing and manipulating objects at remote locations. In their work, they report on a user study and compare the go-go-technique, other arm extension techniques, and a ray-casting technique [Mine, 1995]. The authors also propose the *HOMER* technique, which carries out a combination of the ray-casting technique for object selection and in-hand object manipulation. The paper concludes that none of the tested techniques is a clear favorite, because none of them were easy to use and efficient throughout the entire interaction consisting of grabbing, manipulating, and releasing.

Pierce et al. [Pierce et al., 1997] presents a set of image plane techniques, which enable selection, manipulation, and navigation in virtual environments. Their idea is to work not with the objects, but with their projections onto the image plane. In this way, interaction with any visible objects is possible.

Finally, as stated before, the WIM technique [Stoakley et al., 1995] can also be used to remotely manipulate objects in the space. The user can grab, manipulate and release the objects in the miniaturized world, which are linked with the full size world and its objects. Hence, manipulating the miniaturized objects causes the objects in the surrounding world to change their position, orientation, and other properties.

4.4.2 Remote Object Manipulation – a Taxonomy

In general the remote object manipulation can be realized in two different ways, considering the underlying concept of the technique:

- (UR) the manipulated object (or an icon of it) is brought into the reach of the user's hand (User Reach - techniques);
- (ER) the manipulation tool is extended to reach the remote object (Extended Reach).

To the first set of techniques count the Voodoo Dolls, the WIM, the world scale grab, and the image plane interaction. Within this set, the techniques can be divided in two main categories:

- projection plane techniques;
- manipulation of copy of the target object.

The first category consists of techniques that make use of the projection of the object being manipulated. The second provides an appropriately scaled copy of the target object. This copy (icon) is linked with the original, in such a way, that actions performed on the icon are immediately applied to the original object.

The idea of the second category (ER) is to extend the physically limited reach of the user's hand. To this set count techniques like the go-go interaction, the HOMER technique, and the ray-casting technique. The latter provides a light-ray-like extension of the user's arm that is used mainly to select objects and translate them orthogonal to the ray direction. In contrast, with the HOMER-technique the manipulating hand is extended to reach the remote object using ray-castin-based selection. Afterwards, the manipulations are performed directly as if the object is in the user's hand, so that any rotation can be achieved.

The common feature of the techniques in the first category (UR) is that all of them support interaction with objects in the local environment. In contrast, with the second set of metaphors (ER), the manipulation is performed in the remote location. Unfortunately, none of the referenced techniques allows spontaneous combination of both. This capability would make it possible to exploit the best features of both remote manipulation concepts simultaneously.

Furthermore, none of the above techniques supports precise translation. The approaches in both main categories suffer from this disadvantage. In order to translate an object with an ER-technique, the user can translate and rotate the arm extension. The rotation component, however, has a great impact on the translation, since with increasing distance between the user and the object small rotations can result in significant translations. The only ER-technique allowing adequate translation is the HOMER-technique, since the grabbed object can be translated as if it is within the physical reach of the user.

On the other hand, with the UR-techniques, the user performs the manipulation with scaled (by factor s) copies of objects, which enables precise interaction with the object itself. However, another problem arises, when the aim of the interaction is to translate an object at its remote location. Namely, if the translation is mapped one-to-one and the scale factor s is significantly greater or smaller than one, the precision of the translation is too coarse/fine. For instance, let us assume a house is selected, which is significantly scaled down in order to allow convenient interaction. When the translation is mapped one-to-one, moving the scaled down house by one unit in the desired direction will cause the full-scale house to move in the desired direction by only one unit, which will induce very small translation relatively to the size of the object. The same problem occurs with an object, which is small in the full-scale world, thus has to be scaled up. In this case, the translation would cause too coarse positioning in the full size world, since small displacements of the icon induce significant displacements in the full size world.

Similarly, if the translation is scaled with the scale factor s (e.g. WIM interaction), every little change of the scaled object will cause significant change in the full-scale world. Due to the limited spatial resolution of the tracking hardware, the limited visual resolution of the display system, and the limited precision of the fine hand motion, the correct translation is a challenging task.

4.4.3 TTL Remote Manipulation

What we would like to have is a tool, which allows working with the remote objects in their natural environment at a freely chosen scale. The through-the-lens remote object manipulation is an improvement allowing both modes, ER and UR, to be arbitrarily combined. The basic idea is to allow reaching through the window and manipulating the objects in the secondary world seen through it.

We have shown in Section 4.3 (p. 58) how the secondary world viewed through the output window can be adjusted such that a target location is viewed through it. In this way, a kind of preview window to a remote location is provided, similar to a wormhole known from science fiction. In our immersive setup, this window is mapped onto a hand-held pad tracked with 6DOF for convenient placement.

For the application of the remote object manipulation, we assume that the secondary world is fixed in the space (state V1) as discussed on page 52. In this way, the pad becomes a magic lens revealing the remote location. Once the secondary world is adjusted as desired and fixed in the space of the primary world, the window has to be detached from the surface of the interaction pad. Decoupling the window from the pad's surface allows projecting interaction tools on the latter and applying them as usual. This scenario corresponds to case V1/O1 (p. 52), namely, the secondary world and the output window are fixed with respect to the primary world's space.

After accomplishing this step, the tracked stylus is used to interact with the remote objects. The user can manipulate remote objects by reaching with the stylus into the frustum volume defined by the lens and the current viewpoint (see Figure 4.9). If the stylus is outside this volume, it acts in the local environment in the

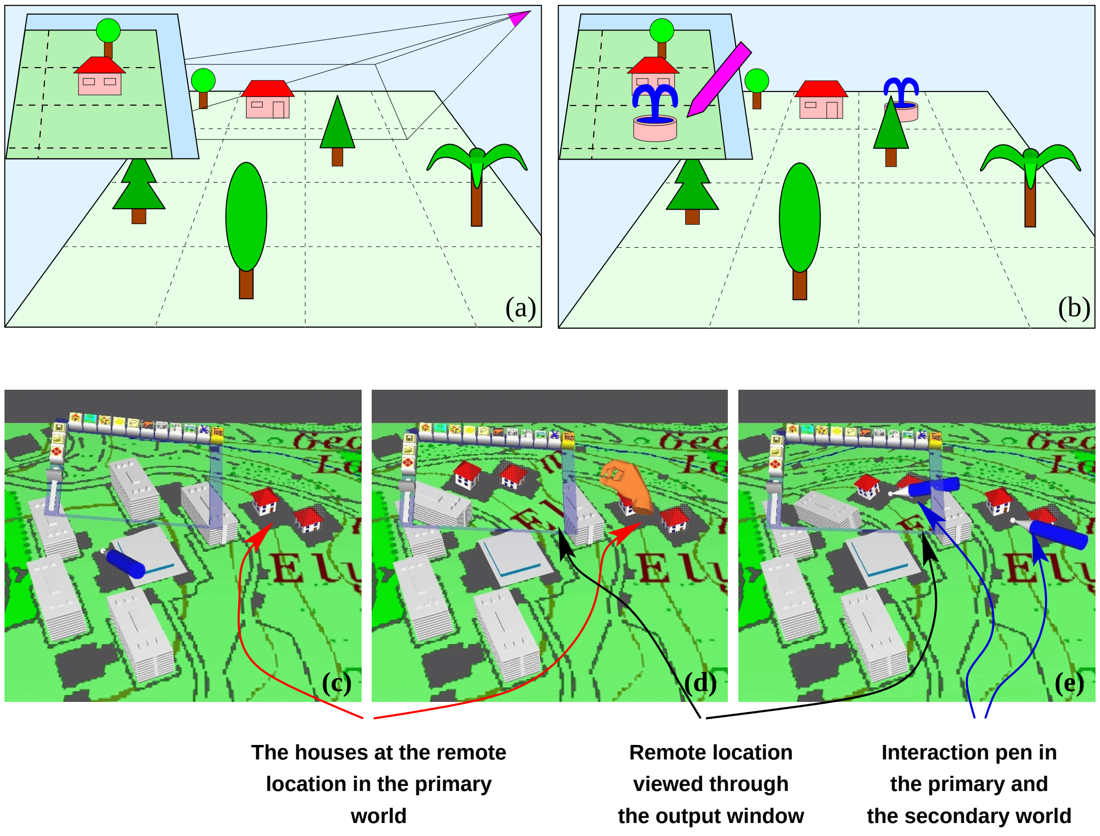

(a)
(b)
(c)
(d)
(e)
The houses at the remote location in the primary world
Remote location viewed through the output window
Interaction pen in the primary and the secondary world

Figure 4.9: (a) and (b) show a sketch of the remote object manipulation. The left sketch shows the output window and the remote location in the primary world, while the right shows an object (fountain) added through the lens at the remote location. The snapshots (c), (d), and (e) show the proposed technique in action. After defining a window to the secondary world, the user can manipulate the objects at the remote location. While the tip of the pen is behind the output window, the pen appears in the primary and the secondary world at the same time (e).

normal way. Moving the stylus from the remote volume to the local volume and vice versa instantly changes the context of interaction. This context switching method is similar to the point-to-focus policy popular in some 2D windowing systems.

This scenario enables the user to select an object at the remote location and change its properties. Furthermore, the proposed tool can be used to rotate and translate the object at its original position in an intuitive way, not supported by most of the ROM techniques published in the literature. Since the secondary world can be viewed at an arbitrary scale, the translation problem addressed above is elegantly circumvented, and the remote objects can be moved with high precision at any desired scale size.

4.4.4 TTL Remote Drag-And-Drop

The change of context applied when the stylus is moved can be exploited to teleport object between locations by drag-and-drop operations between volumes. As soon as the interaction pen and an object picked with it leave the view volume described above, the object is dragged to the primary world (the test is performed for the tip of the stylus). Now the object can be arbitrarily manipulated. When the manipulation is completed, it is put back to its original location.

This remote manipulation strategy is a mixture of the conceptually different techniques discussed in Section 4.4.2 (p. 67). It has some similarities with the Voodoo Dolls: the object is brought into the physical reach of user's hand. It also resembles some features of image plane techniques: the pen is used to grab through the output window to the secondary world. On the other hand, similar to the go-go technique the virtual arm is extended in such a way, that the user can interact with objects at the remote location in the secondary world without being there.

In a slightly more complex scenario, objects can be even transferred between *multiple* remote locations with drag and drop operations. In this way, the user can assemble a complex scene with arbitrary fine details without having to change his/her position in the primary world, while still being able to examine the scene from different viewing positions. Thus, our approach provides a solution to the problem of changing and examining the scene from the current viewpoint, while manipulating objects in distant locations of the virtual world.

4.4.5 Conclusion

We have found through-the-lens manipulations to be both: intuitive and efficient. The user is not required to navigate to the remote location in order to manipulate objects, but can stay at the current location and examine the result of the remotely performed actions. The proposed scenario is useful even if the "remote" location is in the reach of the user, since the scale and the position of the secondary viewpoint can be arbitrarily chosen. For example, a magnifying lens allows precise manipulation of details, while a minifying lens allows manipulation similar to using a world-in-miniature approach [Stoakley et al., 1995]. In addition, the proposed approach offers a sound solution to the translation problem, while carrying out a tool that supports the features of numerous remote manipulation techniques presented in Section 4.4.1 (p. 66).

The through-the-lens remote object manipulation provides a universal technique for working with objects out of the user's physical reach. It proved to be a valuable tool for assembling virtual worlds, circumventing some of the disadvantages of other known remote manipulation metaphors.

Considering the usability of the TTL remote object manipulation, we found out that once the secondary world is adjusted appropriately, the manipulation at the remote location is easy to perform. This is due to the fact that the applied tools behave like normal tools for interaction in the surrounding environment.

4.5 Tools for Interaction with Historical Data

So far, we have described in Section 3.3 (p. 38) the acquisition, preprocessing, and display of historical data. In particular, these steps were performed on data representing the migration and the battles during the peasants' war in 1525 in Germany [Stoev and Straßer, 2001]. In this section, we discuss in detail the interaction with the data in a virtual environment [Stoev et al., 2001a]. The latter is a promising alternative to the traditional desktop paradigm, since in the virtual environment the user can perform various forms of interaction not supported by 2D desktops. Instead of studying 2D maps and attempting to imagine the motion in time, the terrain facts, the participants' speed of motion, etc., the more convenient computer aided interpretation within a synthetic virtual environment can be used. This in turn greatly facilitates the analysis and the visualization of the underlying data, helping to answer questions like: "Why did the participants in the war react in a given way in a particular situation"? Another typical props for analyzing the peasants' decisions offer the display of their range of sight and the range of their weapons. An additional feature of the proposed system is that it corroborates our supposition that the peasants did not have any military education, training or war experience. Their strategy in the battles, the location of the troops, and spread over the battlefields prior to the battle clearly certify this fact

4.5.1 Interaction with 4D Data

Due to the fact, that there is no simple way of interacting with four-dimensional data, we have to apply projection parallel to the time axis as described in Section 3.3.2 (p. 42). Thus, we introduce two different times to our system: the user time t_u, which is constantly and unidirectionally elapsing and the *event time* t_e, which is the time associated with the dataset. In contrast to the user time, the event time can be freely adjusted or set to be continuously incremented or decremented.

In order to enable interactive visualization concerning all four dimensions of the dataset and not restrict the visualization to a simple movie, we consider the following conceptually different interaction strategies:

(a) Interaction with fixed time value/interval and adjustable position in space;

(b) Interaction with a fixed user position and adjustable time;

(c) Interaction with freely adjustable time *and* space parameters (3 1/2 D).

Case (a) is a trivial case, in which the interaction is reduced to regular terrain visualization, since we display a 3D slice with a certain thickness (depending on the time interval) orthogonal to the time axis. In case (b) the interaction is more powerful and enables us to visualize not only the peasants' position, but also to display their movements within an interactively adjusted time interval. Although the alternating combination of (a) and (b) allows gaining insight into the explored data, it does not allow interactive simultaneous manipulation of both parameters: time and space. Thus, the most interesting case is case (c), which is discussed in detail in the following sections.

4.5.2 Continuous Time Increment

Even though it is difficult to navigate through time and space simultaneously, one of these parameters can be adjusted to be continuously incremented. This implies that the incremented parameter is a one-dimensional parameter, or there is another way of mapping one continuously changing value to a multi dimensional parameter. While this parameter is being continuously incremented, the space spanned by the remaining axes can be freely navigated. In this way, we artificially reduce the dimensions of the data and visualize a three dimensional dataset.

When this is applied to the time parameter in our scenario, the event time t_e is incremented by a fixed value. In other words, the event time is elapsing, while the user can navigate freely through the space. If we imagine a two-dimensional dataset and a time dimension, this concept corresponds to the movement of a slice orthogonal to the time axis (i.e. a movie, consisting of 2D images and a time axis). In the four-dimensional scenario, a 3D hyper-plane is continuously moved in the 4D space. The analogy to a *time interval* in 4D is the motion of a slice with certain thickness in the three-dimensional scenario. The data within this slice are projected parallel to the time axis and the resulting image is displayed. This is the most popular way of displaying data containing a time dimension. Most of the 4D datasets are visualized applying such an elapsing time parameter (mostly in an infinite loop) and enabling three-dimensional navigation.

In order to allow the user free exploration of the four-dimensional space, we increase the current time value by a user-adjusted increment. While the time value is continuously increased, the three-dimensional projection parallel to the time axis can be freely navigated at each point in time. Note that the time increment cannot only be adjusted with arbitrary precision, but it can also have a negative value, thus enabling backward motion in time. Figure 4.10 shows a sequence of snapshots, which demonstrates this tool in action. While the time is continuously elapsing, the user is viewing the terrain from different angles. In this case, we show only the viewpoint change caused by the head position change. In practice, any other navigation technique for motion of the primary viewpoint in the primary space can be applied as well.

We also implemented an adaptive strategy for the time increment t_i. In particular, if the space distance between two points in time (P_t and P_{t+t_i}) is smaller than a given value d, the time increment t_i is increased until d is exceeded. In other words, in time intervals in which the displayed geometry does not change significantly, t_i is increased. On the other hand, if the distance in space between two points P_t and P_{t+t_i} is greater than the value d, the time increment t_i is decreased (e.g. halved) until either the distance drops below d, or the minimum time step is reached. In the latter case, the speed of motion is too large to allow display of smaller pieces of the way. Optionally, we provide a time scale that displays to each point in time the speed of motion of the tracked subject. This additional information helps to easily discover intervals in time with no motion at all (e.g. during the night time) or such with high speed of motion.

Despite the fact that this technique is easy to use and therefore often applied in similar visualization scenarios, it has some limitations. For instance, it is suitable for visualizing only a couple of simultaneous time events and can be hardly utilized for depicting many complex events and motions in time. If we aim to provide such functionality, one of the following techniques has to be applied.

4.5.3 Navigation with Continuous Space Movement

A strategy, similar to the introduced time increment strategy has been implemented for the space parameter. Even though, the space is not a one-dimensional parameter,

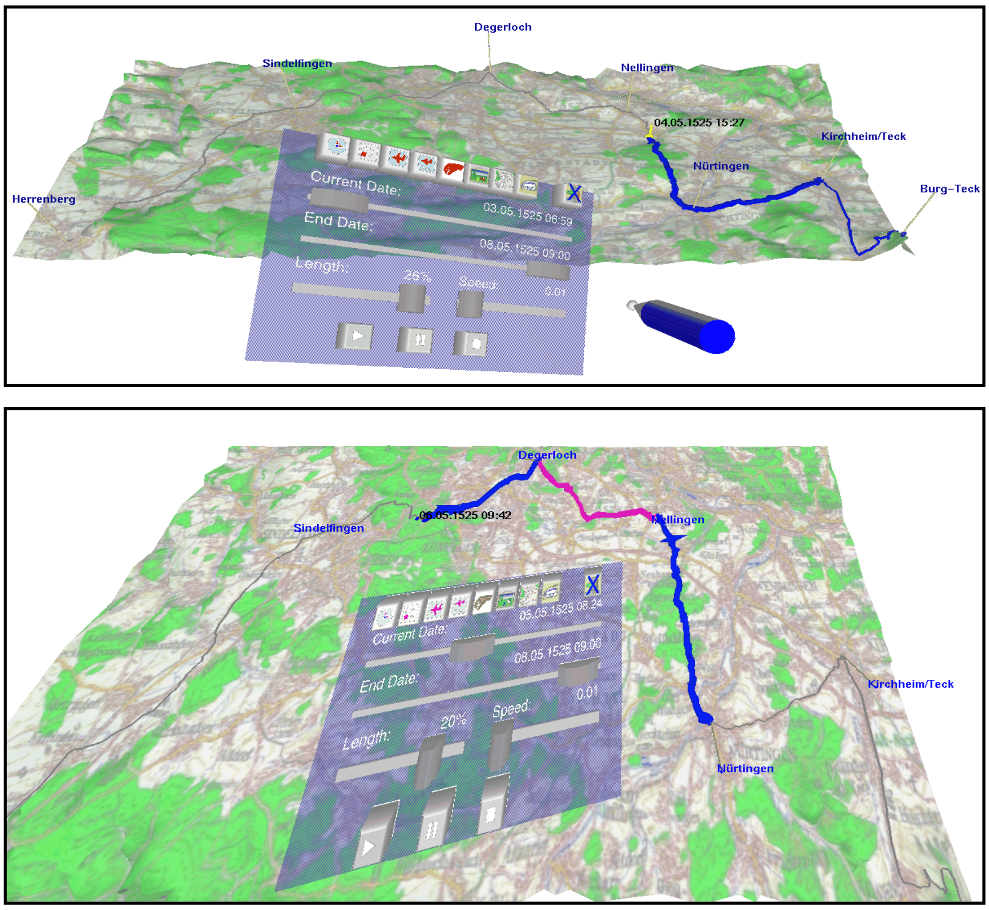


Figure 4.10: While the time is continuously incremented, the user can navigate the 3D space. The upper and lower images show the change of the viewpoint while the event time is elapsing ("extrusion is moving").

we can use a (pre-)computed parameterized path in space. Thus, the problem is reduced to a one-dimensional problem. Instead of permanently increasing the event time, the user can now interactively explore it, while being guided through the space.

We define a path in space in the following way: the user is required to interactively select points, which are interpolated with Hermite splines [Farin, 1988, Hoschek and Lasser, 1993] in order to provide a smooth, interpolating camera path. Note, that in this case, the view direction is not taken into account. Instead, the user can interactively select a viewing direction, while the system pulls her/him along the predefined path.

Once the virtual camera path is defined, the user is required to define the velocity of the flight. This is done interactively modeling the curve of the velocity for the generated path. The result of this step is the function $f(x)$, which maps to each x a camera position in the space. Depending on the speed of the motion, applying a constant increment x_i will cause the camera to move with the desired speed.

When this step is accomplished, the user can start "flying" along the path. During the flight, not only the view orientation, but also the event time, respectively the event time interval, can be interactively adjusted and viewed. This exploration method can be compared with flying an airplane along a predefined route and adjusting the event time in such a way, that one can observe the events happening in the currently visible area. Unlike the continuous event time increment described in the previous section, in this scenario the user can freely manipulate the event time, while being pulled along the predefined path. Thus, another way of exploring a four-dimensional dataset is introduced.

Such type of navigation, however, has one main limitation: the manipulated data can be “out of sight”, if the camera’s view direction is set inappropriately. This means that the extrusion corresponding to the interactively adjusted time interval is not in the viewing frustum of the user. Therefore, another way of interaction is to provide fixed viewing direction along the predefined path. If the aim of the visualization is to provide guidance through the four-dimensional space, the set of techniques proposed in Section 4.5.7 has to be used.

4.5.4 Fly-With Mode

For more vivid visualization, we provide a fly-with mode. When this mode is activated, the user can fly with the tip of the migration and view the environment as seen from the perspective of the peasants. Considering the four-dimensional data we are visualizing, this approach is similar to the continuous time increment presented in Section 4.5.2. In particular, we are visualizing three-dimensional slices of the four-dimensional space, applying the constraint that the tip of the migration is always within the viewing frustum of the virtual camera. This implies that the viewpoint *and* the view direction have to be computed appropriately. When displayed, this corresponds to a guided traveling in space.

It has been shown, that such direct manipulation of the user’s viewpoint can confuse the user and even cause disorientation [Pausch et al., 1995, Ware and Osborne, 1990]. Therefore, we applied the *through-the-lens* concept (see Section 4.2, p. 51), allowing the user to stay at the current location, while simultaneously viewing the migration through a window mapped on the hand-held pad (shown in Figure 4.11). The terrain is fixed with respect to the viewing window. This

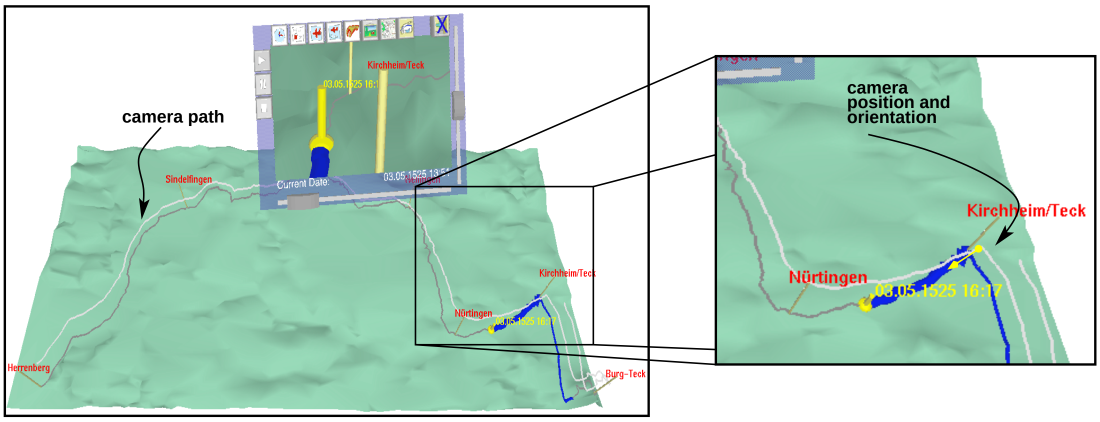


Figure 4.11: The original path l_f is shifted in z direction and interpolated, such that points closer in the time are weighted more than distant points in time. The image also shows the view of the participants projected onto the hand-held pad.

means that moving the window cause the scene behind the window to move with the window, which corresponds to case O3/V2 in Figure 4.3 (p. 53). Additionally, the window can be released from the pad and frozen in space. Thus, the pad’s surface can be used to map another tools on it and interact with the visualized data as usual.

In this scenario, the event time t_e of the data displayed through the window is the same as the event time adjusted in the primary world. In other words, when the user interactively manipulates the event time t_e, the adjusted time is valid in both the secondary world seen through the window and in the primary world surrounding the user. Thus, the secondary world is transformed compared to the primary world, while the event time t_e is valid in both of them.

Due to the sharp turns in the migration's way, if we would follow the path l_f (defined in Section 3.3.1, p. 40), the result would be a camera path with sudden changes in the movement and the orientation of the camera. This fact makes the smoothing of the path indispensable. The simplest way of applying such a smoothing step is to take the original path, to shift the points about a fixed offset z_o in z direction, and to compute for each pixel a (time distance) weighted sum of the time-adjacent points as a function of the time t:

$$P'_t = \sum_{j \in [-k,k]} (P_{t+j}) f_w(j) + P_{zo}$$

Whereas $P_{t+j} = P(t+j)$ is a point on l_f at time $t+j$, $[-k,k]$ is the time interval in which the points are taken into account, P_{zo} is the offset z_o, and $f_w(j)$ is the weight of each point used for the computation of the new point P'_t. The weight function is a simple Gaussian function:

$$f_w(x) = e^{\frac{-x^2}{p}}$$

with parameter p defining the impact of the function in terms of the number of points affecting the final position of P'_t, thus p depends on k.

At the beginning and at the end of the computation, when there are not enough time-sampled points, the first, respectively the last point is extended in the time dimension. Thus, these points are weighted in such a way that the camera path starts respectively ends slowing down the speed of motion at these points.

The result of this interpolation and smoothing method is shown in Figure 4.11. It is clear, that for the path generation we have to take into account not only the pure geometry distance, but also the time. This corresponds to a computation integrating the velocity of the migration's movement and thus of the camera flying with the migration's speed. In order to complete the description of the cameras on the path computed so far, their orientations have to be computed. This is done setting the point of interest to be always the tip of the displayed migration as shown in Figure 4.11.

The problem with this computation appears at places, at which the peasants are staying overnight and then going back the way they approached the place. This causes the virtual camera to stay above the current location, since the tip of the migration is under the camera position, before continuing to follow the moving migration in the backward direction. In this way, the virtual camera always keeps "an eye on the migrations tip", while always flying behind or above the migration. Thus, this tool provides an aid for precise tracking of a moving extrusion in time and space.

4.5.5 The Magic Time Lens

Often we want to know what will happen on a particular location at a particular point in time, while exploring another event time. This means that we have to introduce an additional event time t_{e2}. The question now is, how to display both times without confusing the user and negatively influencing the interaction with the system. Inspired by the magic lens concept [Bier et al., 1994, Schmalstieg et al., 1999], we have implemented a tool called the *magic time-lens* or *time-lens*. With this tool, the event time t_{e2} is valid only within the time-lens-window. Unlike the "fly-with"-mode, with the time-lens the explored area within the window (secondary world) and the terrain surrounding the user (primary world) are always aligned, while allowing for adjustment of different event times t_e and t_{e2} valid in the primary, respectively in the secondary worlds. The time lens enables the user to explore the time and space as described above, while additionally viewing events at interactively chosen location and time.

The time lens provides two time parameters, defining a time interval: the beginning and the ending time of interest (depicted in Figure 4.12). All events happened in

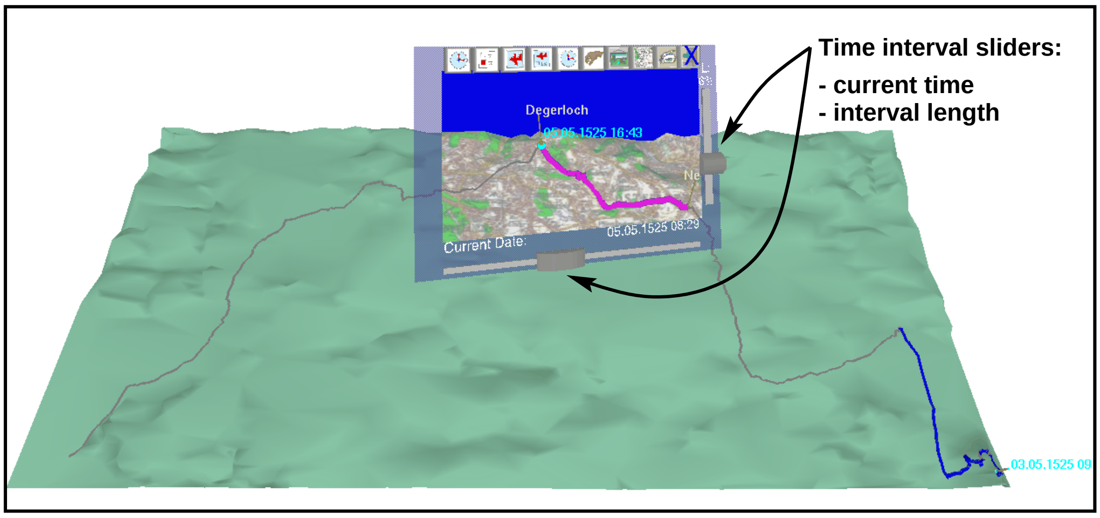


Figure 4.12: The time interval displayed via the time lens can be adjusted using the two slider of the lens. In this case, the lens is used to explore a textured model with names of the places visited by the peasants, as well as a later time than the one explored in the surrounding world.

this time frame are displayed via the lens and are aligned with the underlying terrain, which is assumed constant in the time. When the lens is held close to the viewer's eyes, the entire scene can be viewed through it, thus ignoring the event time t_e and exploring the time t_{e2}.

Combined with the continuous time increment-method (introduced in Section 4.5.2) for the primary world, this concept allows the user to view the virtual world applying the time-increment and to view a region and time of interest simultaneously. In this way, the user is able to quickly discover regions and events of interest, travel to them, and even let the event time t_e elapse slowly in order to study the events happening at the given location.

Furthermore, the time window can be used for viewing migration parameters other than those displayed in the regular view (primary world). Like with the normal magic lens, the color of the extrusion can be set to display parameters other than those shown in the primary view frustum. In addition, a textured model of the underlying terrain can be shown via the lens (as shown in Figure 4.12). This can be a today's texture or even the original data consisting of the hand-made drawings. This imparts the proposed time-lens additional attractiveness and makes it a valuable aid for exploration and interactive study of historical data.

4.5.6 The Time-Space-Window

Combining the time-lens with interactive adjustment of the scene part viewed through the window, results in a time-space-window. In this scenario, we apply the through-the-lens manipulation of the scene viewed through the window as described in Section 4.3 (p. 58). Since the extent of the terrain is limited, the "grab-and-drag"-tool is applied for adjusting the location of interest behind the viewing window. This tool implements one-to-one mapping of the transformation of the interaction pen to the scene viewed through the window. Once this step is completed, the user is able to interactively manipulate the event time valid in the secondary world. Thus, he/she can explore the time and the space of the surrounding world, while simultaneously viewing the scene from a different viewpoint with interactively adjusted event time t_{e2}.

Such a tool can be applied in various different scenarios. A typical utilization is to scale up the surrounding scene in order to explore particular events of interest happening in a particular time interval $[a, b]$. In the same time the user can view a map-like miniaturized version of the entire terrain through the additional window, which contains all events happening in a time interval completely covering $[a, b]$. In other words, the secondary world is a sort of world in miniature [Pausch et al., 1995] used for orientation and extension of the time interval $[a, b]$ visualized in the main view.

On the other hand, another typical application scenario is the reversed situation. The surrounding environment is adjusted to show the entire virtual terrain at a given time, while the additional window is set to display a close view of particular events happening in another time interval. In this case, the window displaying the secondary world is utilized as a magnifying lens that additionally enables arbitrary time exploration.

Even though this tool offers a powerful combination of the through-the-lens exploration concept and the time lens, its circumstantial utilization in terms of time and space adjustment may confuse the user. Therefore, a topic of future research will be the introduction of marks in the time and space, such that when the user explores the data, a time-space window is automatically popped up in order to display the target event from a predefined viewing position. In this way, a keyhole intelligent user guidance defined by an expert user can be provided.

4.5.6.1 Observer Points

In order to facilitate the application of the time-space-like window, we developed a mode in which a privileged user can pre-define so called *observer* positions in the space. These positions can be individually activated and allow to view the scene as seen from a given viewpoint. In order to keep track of the displayed extrusion, the view direction is set such that a selected extrusion is always in the middle of the viewing frustum. Thus, even though the observer's viewpoint is fixed in the space, the orientation is adapted to the position of the tracked extrusion, hence, it is computed depending on the current event time (see Figure 4.13).

As with the previous tools, the scene as seen from these pre-defined positions is displayed in a dedicated window mapped on the interaction pad. This concept enables the user to simultaneously navigate the space and observe the scene from the

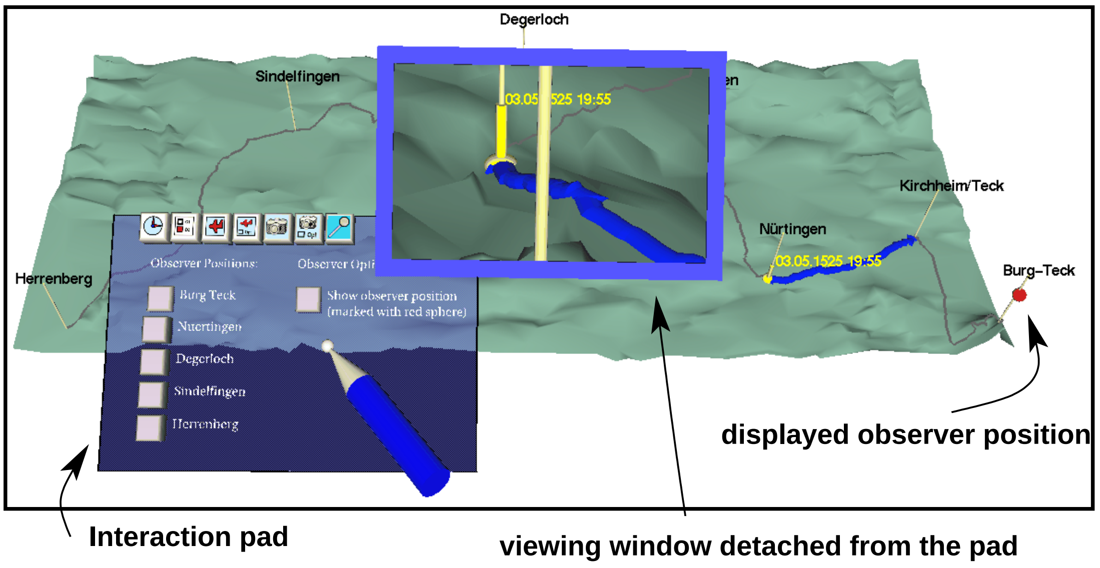


Figure 4.13: The display window is detached from the interaction pad. This allows the user to select a predefined observer position displayed on the pad. The position is displayed as a sphere (see arrow). The extrusion is always in the middle of the display window.

given position. In order to provide better orientation, the observer point is displayed as a sphere in the surrounding full-scale world as well.

4.5.7 Guided Exploration

In order to facilitate the navigation, we also provide guided exploration. This corresponds to motion through the 4D space with predefined path in space *and* speed of motion along this path. Hence, the proposed approach is also suitable for animation generation.

The implementation is an extension of the approach described in Sections 4.5.3 (p. 72): we have to define a (time-parameterized) path in space providing additionally fixed orientation for each of the points on the path. The only condition we have to consider is that the extrusion of interest is within the viewing frustum for each camera position on the interpolated path. This has to be done during the definition of the camera positions.

When a path is interactively defined setting camera positions and orientations, the view direction is set implicitly. Afterwards, we applied (slightly modified) Hermite splines for interpolating the input points and the associated orientations, as depicted in Figure 4.14. In this way, we define a parameterization of the entire camera path. An additional constraint is that the visualized extrusion is visible for each virtual camera position on the interpolated camera path. Hence, we have to consider the motion of the tracked extrusion. This is implemented as a constraint for the computation of the speed of the camera motion along the pre-computed path. As soon as the moving extrusion reaches the border of the image seen from the current camera, the camera is moved slower or faster in order to keep track of it. When the extrusion's motion stops, the speed of the camera motion is slowed down. Depending on the duration of the pause, the virtual camera may also completely stop its motion. Afterwards, when the extrusion continues moving, the camera speeds up slowly. This strategy allows to successfully manage even stops with longer duration,

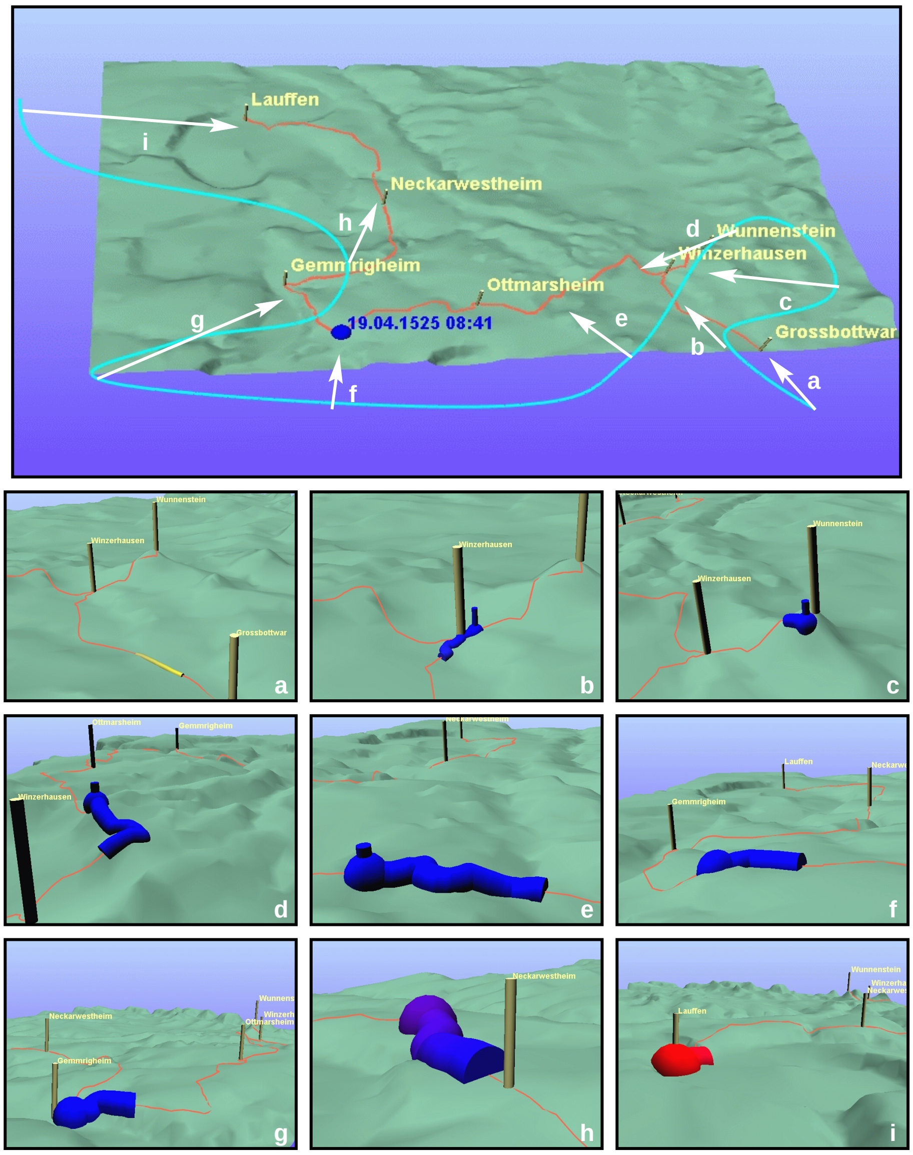


Figure 4.14: Camera path interpolation generated from the points a to i with the given orientation.

providing smooth deceleration and acceleration on sudden "stop and go"-events in the visualized extrusion.

Unlike Gleicher and Witkin [Gleicher and Witkin, 1992], we compute the speed of the camera during the flight generation. Furthermore, we also have to deal with dynamic objects, the positions of which change in the viewed three-dimensional space. However, since we always know the position of the extrusion at the current and later time, these parameters are considered during the computation as well. The result of this step is a smooth path in space, tracking the motion of the time-depending events. Hence, we assigned for each time value t a particular position and orientation of the virtual camera.

Such a path in space cannot only be used for generating animations for guided exploration, but it can be also integrated in the "fly-with"-tool. In this case, the camera path is utilized for setting the appropriate position and orientation of the secondary viewpoint. The scene as seen by that camera is displayed onto the output window mapped on the interaction pad. Like with the “fly-with”-tool, in this scenario the user interactively adjusts the event time t_e. Afterwards, the corresponding camera position for the current t_e is computed and the image seen from this virtual camera displayed.

In doing so, we provide a way for an authorized user to predefine a path guaranteeing that an inexperienced user will not miss particular events when interactively exploring the data. Moreover, displaying the guidance on the pad’s surface in the designated window avoids any impact on the interactivity of the system.

4.5.8 Discussion

In summary, it turned out that the proposed tools were well accepted and judged as valuable and useful by our partners from the Historical Department of the University in Stuttgart. Although each of the techniques can be used for exploration, the combination of all of them provides more powerful visualization tools.

Since each of the tools has advantages and drawbacks, there is no single approach we can judge best. Table 4.2 gives a brief overview of the features and the

Tool	**Advantages**	**Disadvantages**
Continuous time increment	• suitable for interactive exploration of the input data • the user can freely navigate through the space	• the user may miss some features of the data if he/she navigates to a location with no events in the currently explored time
Fly-with mode	• provides a vivid way for tracking the tip of a selected extrusion • through displaying this view in a designated window, the interactivity of the system is not affected • there is still one event time t_e valid in the user view and in the additional window	• only one motion in time can be visualized • each single time segment requires a display window
Time lens	• enables simultaneous view of an additional event time and various extrusion features	• only areas within the primary view can be “enriched” with the superimposed information
Time-space window	• allows to permanently view given location in the primary/secondary world while navigating through the dataset in the secondary/primary world • enables interactive adjustment of the event time t_e	• may be circumstantial for inexperienced users
Guided exploration	• an authorized user can pre-define a path through the space and time and guarantee that given features of the data are not missed during the exploration • does not affect the interactive exploration when displayed in the window on the pad	• too limited interactivity when applied as a sole exploration tool

Table 4.2: Comparison of the proposed tools.

limitations of each tool. The comments in this table are not empirically proven, but were discussed during and after interactive sessions with our system. Therefore, this will be an issue for further research towards finding out which tools are preferred and properly applied by different users.

Based on the observations of interactive sessions, we were able to recognize a typical interaction pattern. In a typical exploration scenario, users with historical background first explored the terrain without manipulating the time parameters. Their aim in this phase was to get familiar with the terrain fact (i.e. cities, mountains, and valleys). Afterwards, in most of the cases they preferred to visualize the events happening in the entire time interval. In this stage of the exploration, the users were creating a mental model of the terrain and the trace of the peasants' motion mapped on it. This was followed by detailed exploration applying the continuous time increment, combined with interactive navigation through the virtual world. Finally, focusing on particular events, the users employed the more advanced techniques, i.e. "fly-with" mode, time lens, etc.

In conclusion, it also turned out that the interactive exploration with continuous time increment is the preferred way to explore the data. It enables intuitive navigation, while allowing the elapsing event time to have the same behavior as the real user time. This was the favorite approach of our partners from the Historical Department of the University of Stuttgart. It was employed for cooperative study of the data, as well as for studying the behavior of the peasants in particular situations.

The second favorite technique was the provided "fly-with" mode, enabling the simultaneous viewing of two perspectives: the interactively manipulated user viewpoint and the miniaturized view of time-tracked data. This technique allows the user to put himself/herself in place of the peasants and to view the situation from their point of view, thus, helping to understand the reasons for some of their decisions. The "fly-with"-mode also allows vivid display of the scene seen from the participants' point of view, while enabling the user to remain at his/her current location. This concept of displaying the synthetic world as seen from an additional viewpoint on a designated window proved to be a useful and powerful visualization approach. Since in this case the primary view position and orientation of the virtual camera attached to the user's head are not automatically changed, confusion and loss of orientation are avoided. Moreover, with the time-lens and the time-space lens, we have presented two tools, which apply the features of the original magic-lens concept, while extending the functionality of the lens into the time-dimension.

Our cooperation with scientists from the Historical Department of the University in Stuttgart has certified the usefulness of the proposed tools. Besides the animation generation, which was a great step towards enhancing the visualization of historical data, the proposed set of tools are valuable companion for visualizing various types of data containing a time dimension.

4.6 Other Through-The-Lens Tools

Despite the remote object manipulation and the visualization of different location and event time, the secondary viewpoint can also be used to provide a “snapshot”-tool. The snapshot mechanism can be used for providing multiple views during the creation and exploration of virtual worlds. Viewing a scene simultaneously from multiple different perspectives is often used in CAD-systems. This in turn allows a better understanding of the explored scene and better orientation. Once the scene seen through the transparent pip is adjusted as desired, a window with this content can be arbitrarily positioned in the space. This *multiple viewpoint* tool can be adjusted to always display a live-scene containing even objects added after the scene was “frozen” in the output window.

Freezing means in this context that the scene, the user sees through the output window, is fixed relatively to the viewing window (compare cases on pages 52 and 52). In the next step, the output window itself can be fixed in the primary space in front of the user, which is the final state O1/V1 (see Figure 4.4, p. 55). Note, that *freezing* does not mean that the scene behind the window is static. It is rather a live 3D picture with frozen viewing window in the secondary world and frozen output window in the primary world.

On the other hand, the snapshot tool can also be used for freezing an unchangeable copy of the scene. This tool may be very useful for reflecting particular stages of development. This corresponds to freezing not only the geometry in the space, but also a particular *event-time* valid only in the secondary world. Each statically frozen window depicts now particular *space and time.*

4.7 Usability and Conclusions

In this chapter, we have presented the through-the-lens metaphor. Based on this metaphor, we have described a set of navigation techniques, a technique for remote object manipulation, and various techniques for interactive exploration of 4D historical data. One of the most interesting questions arising at this point is: “How do the TTL-tools perform and are they easy to understand and use”?

We have not performed usability tests and detailed comparison with other related tools, instead, we have let 50 users work with the proposed techniques. Some of them had no experience with VE-interaction, other were very experienced users. It turned out that all participants were able to easily grasp and immediately apply the TTL-concept. Only a short demonstration was provided, explaining the idea of the tools. Thereafter, the participants were let to work with the application on their own.

We found out that once the concept of the interaction was clear and the participants could adequately apply the pen and pad paradigm, the TTL-tools were easy to comprehend and to use. The only problem the users had, was related to the eyeball-in-hand technique. Since with this tool there are 6-DOF, some users had difficulties positioning the additional camera appropriately. Nevertheless, the feedback we had considering the majority of the TTL-tools was quite positive. Even people with no or little computer experience could interact with the system surprisingly good. This fact testified a good overall usability of the TTL-concept and the tools based on it.

Chapter 5

Rendering Algorithms

Considering the application of the proposed tools and visualization strategies, but also aiming to display the nowadays-typical huge amount of data at interactive frame rates, we face another problem: the efficient rendering. Although the graphics hardware is becoming ever faster, the requirements for new hardware increase with each new graphics hardware generation. Typical consumer graphics hardware boards are able to render more than 25 millions triangles per second (NVidia GeForce2 GTS). Even though these are impressing numbers, scenes of large size and rich in details rapidly reach the bounds of the hardware. Since we are not dealing with hardware issues in this work, we discuss algorithms for rendering (parts of) virtual scenes on existing hardware, using alternative rendering strategies. In particular, we address the representation of large scenes under the constraint of (a) application in stereo-based virtual environments and (b) applying the through-the-lens navigation, manipulation, and visualization tools discussed in the previous chapter.

5.1 Introduction

In the past years many researchers have presented new approaches and techniques aiming to accelerate the rendering of three-dimensional scenes. The most prominent research direction is still the image-based rendering (IBR), which makes use of the fact that the display resolution is limited and that in large scenes most of the data do not significantly contribute to the final image. In addition, the success of image based rendering techniques is mainly based on the fact that the geometrical complexity of a scene, we want to render using image-based representation, does not affect the complexity of the image-based scene representation itself. The image-based representation depends in general on the (limited) resolution of the images used for the generation of the IBR-representation. Due to these facts, IBR-techniques allow rendering of scenes with nearly arbitrary complexity in real-time, which is especially important in virtual environments. This makes IBR techniques a suitable candidate for embedding them into our tools, since with the original through-the-lens tools the entire scene viewed through the output window (the secondary world) passes twice through the rendering pipeline in the straightforward implementation.

In the second part of this chapter, we address another problem: the stereo rendering, which requires the entire scene to be rendered twice, each time with a slightly different camera position representing the observer's eyes. Fortunately, we have found that parts of the images seen from these two different camera perspectives are very similar. This fact we exploited in order to split the scene in two areas: the one close to the virtual camera, which appears to be different for both eyes (near foreground) and the area further away from the user (far background). The implementation details, as well as the user studies performed with this approach, are presented in Section 5.3.

5.2 The Multi-LDI

The research in this direction was motivated by the fact that each through-the-lens tool can only be viewed through a given window, thus the viewing angle, as well as the part of the scene visible through the viewing window are limited. This makes the integration of image-based techniques a logical part of our efforts towards developing fast rendering algorithms for the tools proposed in the previous chapter. In this section, we show how the above features of the through-the-lens tools are exploited in order to provide algorithms for efficient rendering of the secondary world (behind the output window)[Stoev et al., 2000b, Stoev et al., 2001c].

5.2.1 Image-Based Rendering

Various attempts to integrate IBR in virtual environments have been reported in the literature. For architectural walkthroughs [Aliaga and Lastra, 1997] or city scenes [Wimmer et al., 1999a] it is often sufficient to use pictorial information to replace distant parts of the scene. Instead of rendering the complete interior of a distant room seen through a door, Aliaga and Lastra [Aliaga and Lastra, 1997] use a *portal texture*. Such a texture shows a picture of the room and is placed into the door opening. The texture is replaced by real geometry when the user approaches the respective door.

In [Rafferty et al., 1997, Rafferty et al., 1998] the portal textures, which are “flat” images, are replaced by *depth images*. Depth images store to each pixel color additionally the disparity value of the latter. This allows carrying out a correct perspective warp of all pixels with respect to the viewer’s position. In contrast to portal textures, the perspective distortion of the image is handled correctly. Problems with depth images occur if parts of the scene are visible from the viewer’s position, which are not stored in the depth image. Therefore, Aliaga and Lastra [Aliaga and Lastra, 1999] employed *Layered Depth Images (LDI)* [Shade et al., 1998] to replace parts of a complex scene, which are distant from the observer. Unfortunately, the time and memory requirements of their approach are very extensive, because all LDIs (from 180 to more than 5000 LDIs), for all possible view directions have to be calculated. Since this was reported to take from one up to 28 hours, the LDIs are computed in a preprocessing step.

5.2.2 The Image-Based Interaction Props

In Section 4.2, we have presented the through-the-lens concept for rendering scenes seen through a given output window (e.g. mapped on the surface of the interaction pad). Since the scene seen through the output window is in general not fixed with respect to the output window and can be even manipulated concerning size and viewing angle, image based techniques are rather unsuitable for this scenario. In other words, as long as the scene viewed through the output window is not fixed with respect to the output window, image-based rendering techniques are rather slowing down the rendering process. This is due to the slow acquisition of the data for the image-based representation. First, when the secondary world is fixed with respect to the output window, an IBR approach can be applied. This is the case in all scenarios O1-O3/V2 (on pages 52 and 52), as well as O1/V1 and O2/V3. In all these cases, the movement of the output window does not cause the secondary world to change significantly, i.e. only the viewing angle changes when the user moves the current viewpoint in the primary world (e.g. all cases O1-O3/V2).

A strategy implementing case O1/V1 is often used for defining 3D "snapshot"-windows, similarly to the window defined before the remote object manipulation is applied (see Section 4.4, p. 66 and Section 4.6, p. 82). With the snapshot tool a live 3D picture of the scene can be made in order to view the scene from a particular viewpoint. On the other hand, the snapshot tool can also be used for freezing an unchangeable copy of the scene, reflecting particular stages of development of the virtual world. In this case, this tool acts like a simplified time-space window, as described in Section 4.5.6 (p. 77). In this section, we refer to both the remote object manipulation tool and the snapshot tools for exploration of artificial worlds as *remote snapshot*-tools.

Besides its strengths, the main problem with the realization of the through-the-lens concept is that every time the artificial world is caught on the interaction pad (on which the output window is mapped), a reference or a copy of the latter is added to the rendered data. Thus, the original scene has to pass through the rendering pipeline an additional time for each provided viewing window. Even though these additional viewports are in general much smaller than the primary viewport, the complete scene is rendered once for each additional viewport. Since this is unacceptable for large, detail-rich artificial worlds, we employed an image-based technique for accomplishing this task. The proposed technique allows displaying the remote snapshot of the scene independent of its complexity, while supporting all features of the original remote snapshot tools.

Another contribution of this work is the extension of the through-the-lens-WIM-metaphor described in Section 4.3.2 (p. 60). The problem is similar to the one introduced by the above addressed remote snapshot tools, namely the doubling of the rendered data. The WIM representing image-based structure, however, has to be computed with significantly higher resolution, in order to provide sufficiently detailed information when parts of the map are zoomed. Fortunately, the map tool does not have to be generated on-the-fly during the interaction, thus we can compute it in a preprocessing step. This is the main difference between the image-based realization of the remote snapshot tools and the TTL-WIM-tool. Since the user has the freedom of choosing arbitrary viewpoints for freezing the scene with a remote snapshot tool, it is not possible to pre-compute a number of images in this scenario.

5.2.3 Requirements

To summarize, the through-the-lens props introduce the following requirements, which have to be considered when developing an appropriate image-based rendering technique:

- Fast data acquisition on-the-fly,
- Fast rendering of multiple views of the scene, covering a wide angle of viewing directions,

- Addition and removal of geometry objects to/from the scene.

In order to meet these requirements, which are addressed in the subsequent sections, we developed a new image-based data structure, the *multi LDI* [Stoev et al., 2000b, Stoev et al., 2001c]. A multi LDI consist of several small LDIs instead of providing one large LDI for covering the entire range of view directions (in our case a hemisphere).

5.2.4 Layered Depth Images

The LDI idea was first introduced by Shade et al. [Shade et al., 1998]. LDI stands for Layered Depth Image and is in general an image, in which every pixel represents a ray (Figure 5.1). Each ray stores a number of depth pixels. Thus, parts of the scene

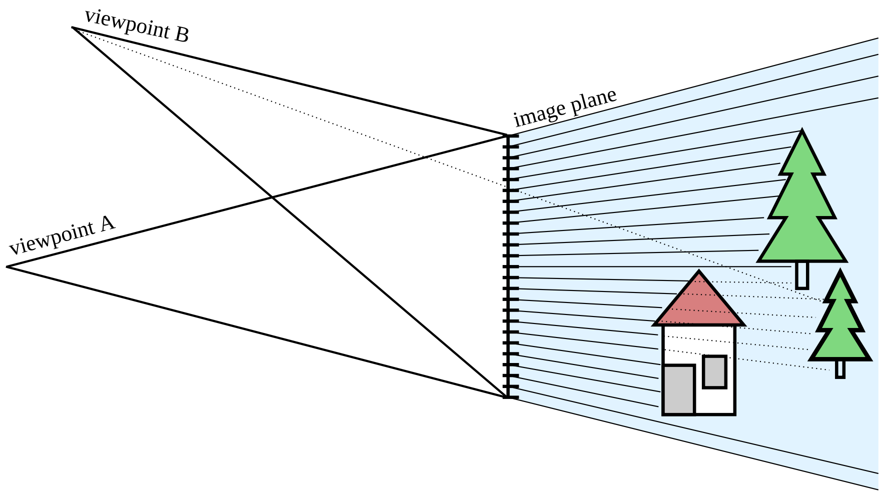


Figure 5.1: LDI storing three objects. The small tree is not visible for viewpoint A, but visible for viewpoint B. A depth ray in the LDI may store more than the first hit of a viewing ray as shown for the small tree.

that are occluded for the main viewpoint (see viewpoint A in Figure 5.1) are stored in the LDI as well. In this way, an LDI permits to calculate different views of a scene without exposing any gaps caused by incomplete object information. The rendering time and the memory requirements of an LDI depend linearly on the total number of all pixels stored in all rays of the LDI structure. In contrast to *layered impostors* [Decoret et al., 1999, Schaufler, 1998], LDIs do not use a fixed depth resolution. In this way, LDIs provide high image quality, while requiring moderate amount of memory. Furthermore, the acquisition of LDIs is very simple. Using a set of depth images, the spatial position for each source pixel in the LDI can be calculated and the pixel inserted into the LDI structure. Pixels representing the same surface point, but originating from different views, are easily removed by comparison of the projected depth values, thus keeping only relevant information stored in the LDI.

LDIs are widely used in various application areas, where complex geometry rendering is replaced by image-based rendering. For instance, in an *Image-Based Object* (IBO) [Oliveira and Bishop, 1999], six LDIs form a single object, which can be viewed from all directions. Up to three of the LDIs, positioned on each side of a cube, are rendered to generate a correct view of the stored object. Chang et al. [Chang et al., 1999] introduce in their work an extension of the LDIs for rendering of multi resolution images, which is another valuable feature of this image-based representation.

As stated before, the memory requirements and rendering time for an LDI depend linearly on the number of depth pixels stored in the LDI in total. The higher the amount of internal occlusion in the LDI, the less efficient will be the rendering. Apparently, for an average scene the following observation is valid: the more the viewing position is altered, the more visibility changes in the scene will occur. In the context of LDIs this means: The greater the range of viewing angles for which an LDIs enables the rendering of a correct image, the more depth pixels have to be stored in it and the less efficient will be the LDI rendering.

Since we have to cover the whole hemisphere of possible viewing angles above the output window for each of the through-the-lens tools, storing all the scene information in a single large LDI would be quite inefficient. Therefore, the proposed multi LDI subdivides this hemisphere into patches and attaches one single LDI to each of them as shown in Figure 5.2. Each time a through-the-lens prop with secondary world fixed with respect to the output window is rendered, the viewing angle relative to the display window is used to determine the appropriate LDI patch. Since each of the LDIs covers only a small range of view directions, it can be rendered very fast.

In contrast to an IBO [Oliveira and Bishop, 1999], which requires the simultaneous rendering of up to three LDIs, with the multi LDI only one LDI is rendered at a time. Furthermore, each LDI of an IBO covers a wider range of viewing directions. Thus, the rendering time for each IBO's LDI is greater than the rendering time for one LDI of the multi LDI. Finally, the LDIs for an IBO have to be pre-computed and cannot be generated on-the-fly as this is done with the multi LDI as shown next.

5.2.5 Application

Initially, the interaction described in Section 4.3 (p. 58) is applied for adjustment of the output window in the primary world and the viewing window in the secondary world. Once this is task is completed, we assume that the viewing window is fixed with respect to the secondary world. Thereafter, the output window can be fixed in the primary world's space in order to perform remote object manipulation or to keep track of the changes at the particular remote location (snapshot tools). This corresponds to state O1/V1, namely to an output window fixed in the primary space and a viewing window fixed in the secondary space.

Since the creation of an LDI can be an expensive task, we left the initial concept of geometry-based rendering unchanged, until the user freezes the output window in the primary space. When this task is completed, we assume that the secondary world will not be significantly manipulated considering its size and position. At this point, the generation of the multi LDI is triggered and the generated LDI is displayed in the output window. In case further transformations are applied to the secondary world, the geometry representation for displaying the scene is activated again, until the secondary world is fixed in space again. In other words, when the scene is being manipulated and when the current state of the primary and secondary world is not state O1/V1, the scene is rendered as geometry, otherwise it is rendered applying the image-based representation.

In this way, the rendered geometry is not more than doubled, since there is always exactly one primary world and one *active* secondary world (being currently manipulated). This doubling is independent of the number of the output windows in the primary space (fixed in space or attached to the interaction pad). Each time a geometry containing output window is frozen in the primary space, the LDIs for it are generated and the rendering of geometry is avoided. Hence, there can be no more than one copy of the scene rendered as geometry at a time. Thus, the rendered geometry is doubled in the worst case.

Furthermore, when the output window is fixed in space, the user can approach it in order to enter the secondary world as described in Section 4.2.2 (p. 56). Since the main application of the proposed image-based approach is to enable viewing from a particular distance, if we would allow the user to enter an image-based representation of the secondary scene, this would be a very coarse sampled scene. Therefore, we have to introduce a criterion for switching from the image-based representation back to a geometry-based representation based on the distance between the current viewpoint and the output window: In case the user is close to the output window, the geometry representation is rendered, otherwise the proposed image-based representation is displayed.

5.2.6 LDI-acquisition

When the user freezes the output window in the primary space, we start with the LDI-generation. First, an imaginary hemisphere over the front side of the output window is aligned with the center of the window (Figure 5.2 (a)). The radius of the

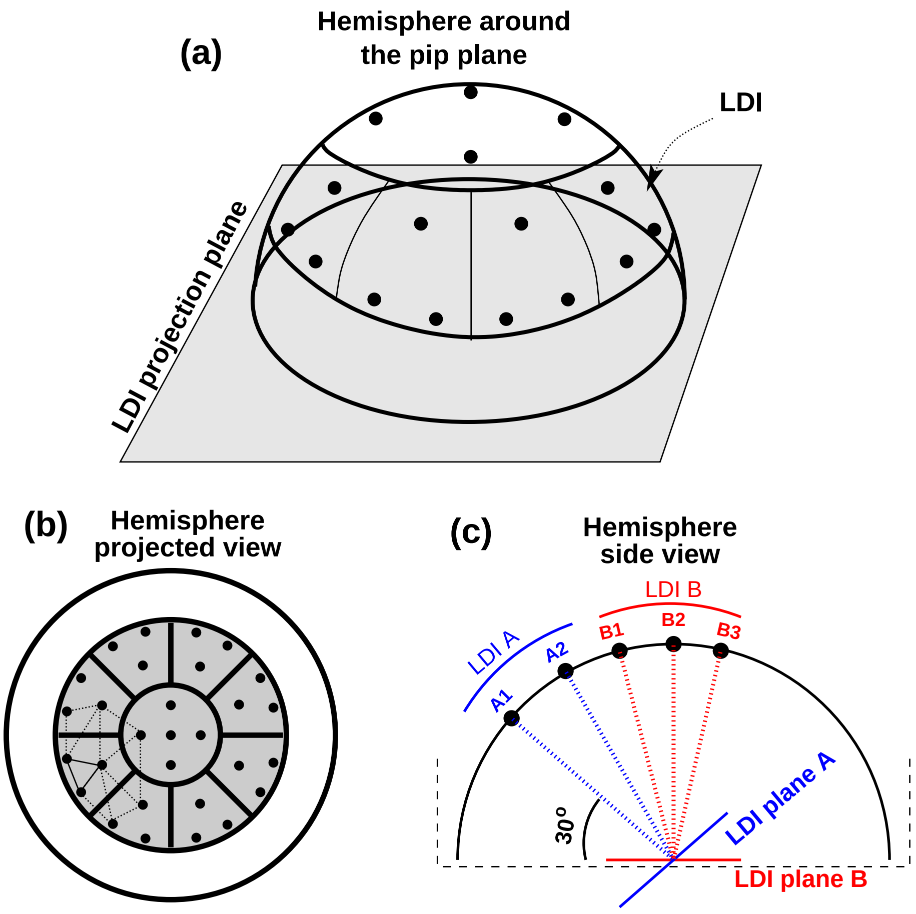


Figure 5.2: The dots indicate the camera positions used to generate the pictures for the single LDIs. The patches correspond to LDIs. The set of all LDIs defines the multi LDI. For the generation of the LDI A in (c), the depth images of the cameras $A1$, $A2$, and $B1$ are applied. The LDI B contains the data from the cameras $B1$, $B2$, $B3$, and $A2$.

hemisphere is set to be the current distance between the user's head position and the center of the output window. Since the output window is generally attached to the interaction pad, in practice this distance corresponds to half up to full hand reach (i.e. $30 - 80cm$).

Now the hemisphere is divided into eight equally sized patches and a central patch, defining the multi LDI, as shown in Figure 5.2 (b). The patches cover 2/3 of the hemisphere's area corresponding to an angle of $120°$. For each patch, which defines an interval of viewing directions, an LDI is created. The image plane (LDI-plane) of each patch is parallel to the plane defined by the three camera positions of the patch and contains the midpoint of the output window (Figure 5.2(c)). The image plane of the central patch is the surface of the output window.

The outer shell of the hemisphere is not covered with patches, since the view-angle in this area is very narrow and rarely used. In general, the user views the window from the front, moving the virtual camera within the space covered by the LDI-patches. We ascertained that this restriction is not limiting or distorting the utilization of the remote snapshot tools. However, in order to not confuse the user, we display a kind of an opened box (Figure 5.2(c)), which occludes the area not covered with LDI-patches before gaps can occur.

In the next step of the LDI-acquisition, we position a number of virtual cameras in each patch as depicted in Figure 5.2 (b). Each camera uses an off-screen-renderer to generate a depth image of the scene (Figure 5.3). Knowing the position and orientation of the camera we can compute the position of each depth pixel in the LDI.

For the generation of a single LDI, we merge the depth images generated with cameras positioned within the patch (dots in each patch in Figure 5.2), as proposed by Shade et al. [Shade et al., 1998]. Initially, we need three images with the corresponding depth values for each pixel in order to enable the rendering of the scene from the current camera position. Note, that these three camera positions may not belong to the same patch. The depth images generated with each of these virtual cameras are assigned to each of the adjacent patches. Thus, the complete depth image information is put into each of the affected patches. This means that to each patch belong not only the cameras within the patch itself, but also five cameras from the adjacent patches (for the eight symmetrical patches, as shown in Figure 5.2(b)). For the central patch, eight additional cameras are involved in the LDI generation. During rendering of the multi LDI we use only the patch, which contains the current viewpoint.

As soon as the current viewpoint of the user moves out of the triangle defined by the three initial camera positions, an additional image has to be offline rendered and added to the corresponding patches. In this way, after generating the initial three depth images, we always add *one* depth image to the region already covered with LDI patches. Hence, the implemented multi LDI behaves as a progressive dynamic data structure, adapted to the current viewpoint.

Due to the multiple insertion of a depth image into all adjacent patches, we can guarantee that no distorting "flickering" appears, when the view direction moves from one LDI to the next. Note, that this is especially awkward, when the LDI is viewed in stereo mode and each eye sees the image splatted from a different LDI patch.

On the other hand, this problem does not occur, when one large LDI is utilized. Using a set of small LDIs, however, provides much better performance than utilizing one large LDI for the whole hemisphere. As stated above, since each two adjacent LDIs contain common depth images, the switching from one LDI to the other is im-

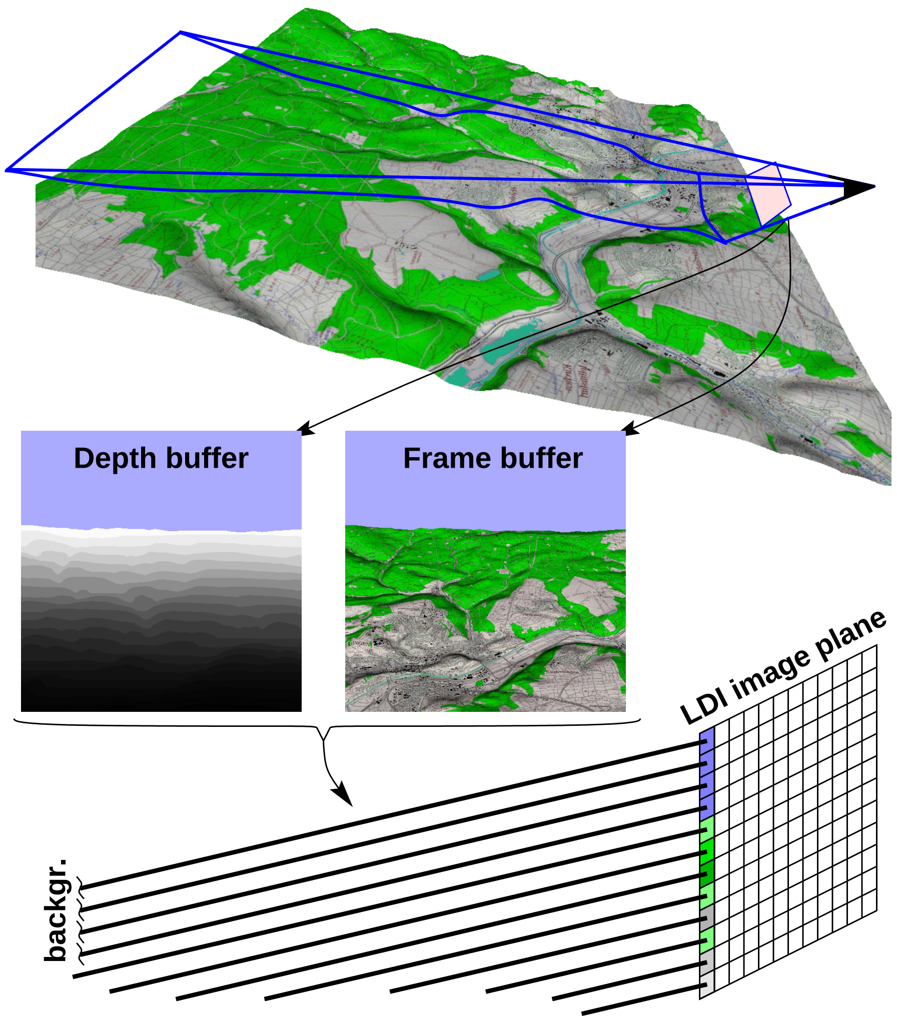


Figure 5.3: In order to create an LDI, we use the frame and depth buffer data of each shot picture and sort the pixels in the appropriate LDI(s).

perceptible for the observer.

5.2.7 The LDI Structure and Splatter

As introduced before, each LDI of the multi LDI consists of a number of rays. Each time a depth image is added to the LDI, the pixels are inserted in the appropriate rays. First, the depth values are used to compute the position of the associated pixels in the space with respect to the origin. Thus, an inverse projection transformation is applied to all pixels. Afterwards, the position and the orientation of the virtual camera, with which the depth image was shot, are taken into account in order to determine the final pixel position in space. Now the pixel is projected onto the image plane for finding out to which ray it has to be assigned. Unfortunately, this approach may cause too many redundant pixels to be stored in each LDI-ray, when multiple depth images are added to the same LDI-patch. Therefore, pixels on a ray with similar depth and color are assumed to represent the same object. Thus, they are merged and replaced by one single depth pixel.

During the rendering of an LDI onto the output window frozen in the primary space, we first determine the position and the angle of the window (LDI-projection plane) relative to the viewer. This determines which LDI-patch of the multi LDI has to be rendered. The current camera position and the projection plane position are used to compute the projective transformation, which maps the LDI pixels onto the projection window (Figure 5.4). The pixels of the LDI are splatted using a software

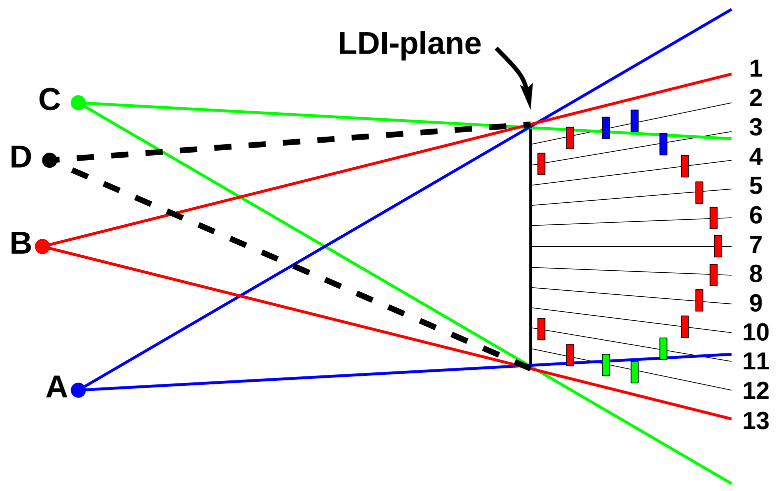


Figure 5.4: The camera positions A, B, and C are used to generate the pixels in the LDI. The contribution of the cameras A and C are the pixels, occluded for camera B (on the rays with the numbers 2 and 3 respectively 11 and 12). When the image seen from the camera position D is generated, the projection transformation is computed and applied to all pixels stored in the LDI.

renderer. In order to access the single pixels in such a way, that they can be successively drawn without applying an explicit depth test, we implemented McMillan's ordering algorithm [McMillan, 1995]. McMillan introduced in his work an approach, which defines how the single rays have to be back-to-front traversed in order to avoid depth tests once the pixels are sorted for one particular viewpoint (e.g. viewpoint A in Figure 5.1, p. 86). Depending on the position of the new viewpoint (e.g. viewpoint B in Figure 5.1) relative to the original viewpoint, the rays have to be traversed in a given order. Applying this algorithm, the depth pixels are sorted once, thereafter, the depth test is not required in order to compute the final image seen from the new viewpoint.

A similar strategy based on McMillan's ordering algorithm was applied by Shade et al.[Shade et al., 1998]. In contrast to their work, however, we compute the splat size for each pixel individually[1]. This increases the precision and avoids visual artifacts caused by the coarse splat-size approximation. The resulting picture is rendered as a texture, which is mapped onto the projection plane using the graphics hardware.

Since the Studierstube is based on Open Inventor, we implemented the LDI-splatter as an Inventor class derived from the regular 2D texture class. The new object is able to draw itself (i.e. the secondary world) depending on the current viewpoint in the secondary world. As stated above, this image-based representation is only used when the secondary world is fixed with respect to the viewing and output window. Thus, the viewpoints in the secondary and the primary worlds are moved

[1]In some cases one depth pixel may be responsible for more than one image pixels in the final image.

simultaneously. This means that we can use the primary virtual camera position in order to determine the position of the secondary viewpoint.

During rendering, each time an instance of the above class is traversed, a new view-dependent texture is generated and mapped onto the given window in space. Thus, it creates the illusion of viewing a real 3D world behind the LDI's projection plane. This holds also for stereo mode setups, in which the appropriate texture is re-computed for each eye and every frame.

5.2.8 Adding and Removing Objects in an LDI

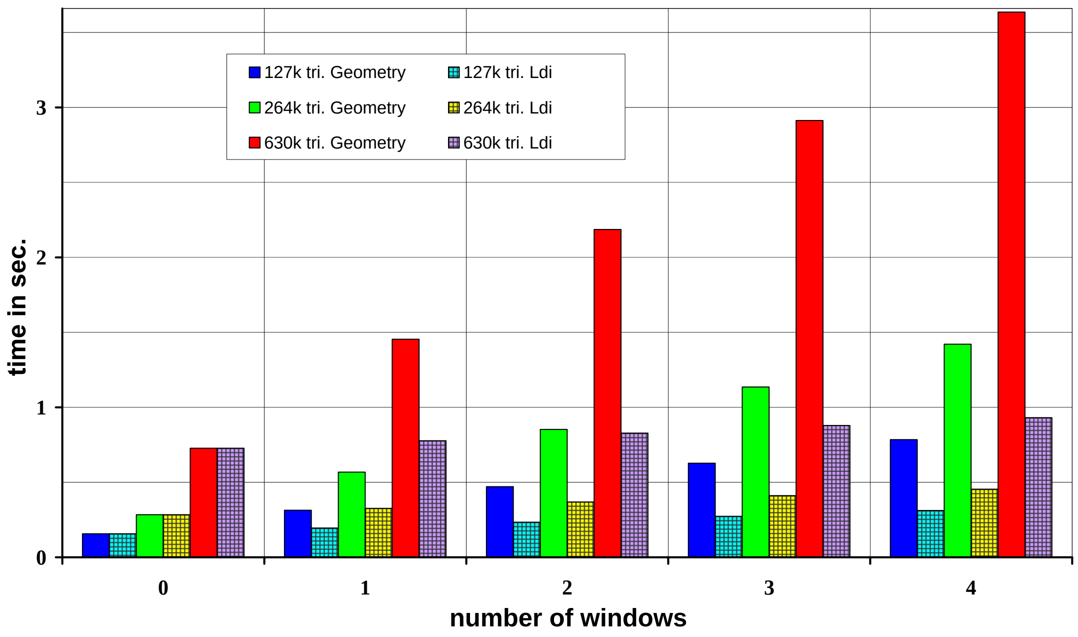
The scene frozen on the multi-LDI contains all details of the latter, being present when the acquisition was started. However, depending on the application area, updates of the LDI might be desirable. For instance, during the construction in a virtual environment a window (multi LDI-rendered) may be applied to show an area invisible from the current viewpoint. In this window, objects added or manipulated in the primary or in the secondary world (using the remote manipulation concept presented in Section 4.4, p. 66) have to be displayed in the multi LDI-rendered window as well.

This concept can be easily adapted to the LDI data presentation. When an object is created and it is visible in the LDI window, we can render the object as a geometry object. For this, we need the depth information for the pixels in the final splatted image. Fortunately, the LDI structure presented above can be easily extended to provide this capability. In this case, we need the depth values of each pixel in the final image. We first render the LDI-representation of the scene, as this is done in the usual rendering step, whereas, we generate beside the color image a depth image as well. Thereafter, we have to copy the depth image into the z-Buffer of the graphics hardware. Finally, the geometry object is drawn as usual and the graphics hardware computes which parts of its geometry are visible and which are not. Thus, a mixed rendering including geometry and LDI data has to be performed. When an object is removed, its geometry is simply removed from the list with objects rendered in the output window.

5.2.9 Results

In order to evaluate the performance of the proposed technique, we compute the times for the geometry rendering of three scenes with $126'790$, $264'247$, and $630'799$ triangles respectively. Afterwards, we compare the times for rendering various number of through-the-lens windows fixed in the primary world, applying traditional geometry rendering and rendering of the multi LDI data (Figure 5.5). The number of windows varies from none, which means that only the surrounding world is rendered as geometry, to 4 windows. The measurements are performed on a Pentium III 733Mhz machine (1Gb main memory) with NVidia Ge-Force 2 GTS graphics hardware (64Mb texture memory).

The window with the LDI containing the scene was positioned in such a way, that the required resolution of the multi LDI was not more than 256×256 pixels. This is quite large when we consider, that the resolution of the Virtual Table we use in stereo mode is 1024×768. After offscreen-rendering a depth image, the time required for inserting it in an LDI-patch was on average 0.06 seconds. Since each pixel is inserted in a depth-sorted order, this time varies depending on the number of the pixels that are already stored in the LDI.



	127k triangles rendered as		264k triangles rendered as		630k triangles rendered as	
Windows	Geometry	Ldi	Geometry	Ldi	Geometry	Ldi
0	0.1567	0.1567	0.2838	0.2838	0.72708	0.72708
1	0.3136	0.1953	0.56788	0.3261	1.45416	0.77736
2	0.4708	0.2341	0.85276	0.3687	2.18624	0.82796
3	0.6272	0.2731	1.1355	0.4111	2.91304	0.87896
4	0.7844	0.3121	1.421	0.4541	3.636	0.93096

Figure 5.5: Comparison of the geometry rendered snapshot tool and the multi LDI-based rendering. All measured times are given in seconds.

More important is the rendering time for the LDI when applied in the proposed tools. Depending on the viewing angle and the distance to the current viewpoint this time may vary as well: when the observer is close to the projection plane a single splatted depth pixel may cover several image pixels. The values for the multi LDI rendering shown in Figure 5.5 are “worst case” values. They include the time for computing a texture with the given resolution (splatting) and mapping it on the output window. On our machine this step took in the worst case 0.02 seconds per frame.

Figures 5.5 and 5.6 clearly show, that the proposed technique is especially useful, when several views are used during exploration of the scene, and when it contains complex geometry. For instance, let us consider a case in which there are two simultaneous views of the scene and the regular surrounding scene, i.e. the scene is rendered three times per frame. Our scene with $126'790$ triangles requires approximately 0.47 seconds per frame using the standard geometry rendering pipeline. Applying our approach, this cost is reduced to rendering the scene as geometry once and rendering two multi LDIs. This results in 0.23 seconds in total. In other words, the rendering time was reduced by more than 50%, which means that the frame rate was more than doubled.

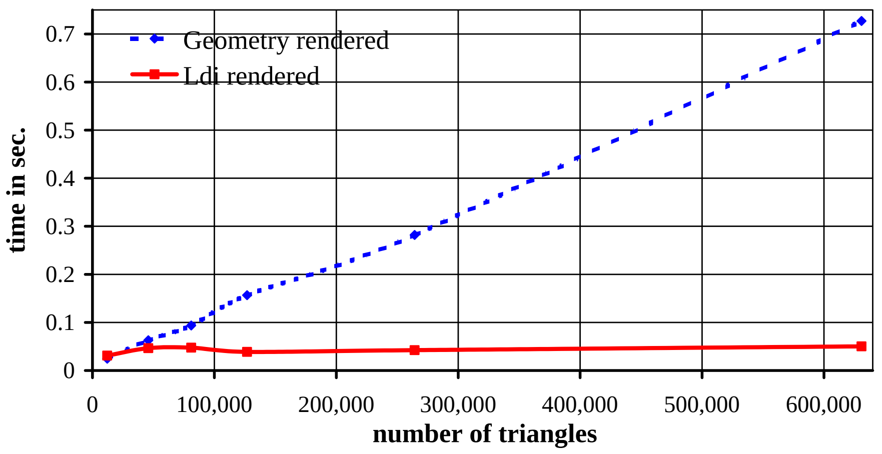


Figure 5.6: Rendering time for scenes with various complexities, rendered as geometry and as multi LDI data.

5.2.10 Conclusions

The image-based rendering structure presented in this section can be rendered significantly faster compared to geometry rendering, as well as compared to rendering a single large LDI. We have also described in detail how the multi LDI can be integrated in a through-the-lens tool and in which scenarios this data structure can be applied for accelerating the rendering performance of the system. Furthermore, we discussed an extension of the standard LDI-approach towards allowing for a dynamic addition and removal of geometry objects. This feature allows us to exchange the rendering algorithm: The geometry based rendering is replaced by image-based rendering, without any perceptual difference for the end-user.

In our opinion, the proposed multi LDI approach is a promising modification of the standard geometry rendering of the through-the-lens tools, considering the growing amount of data managed in virtual worlds nowadays. With the multi LDI we managed to exploit the constraints introduced by the through-the-lens tools, namely the limited angle of viewing directions, as well as the limited size of the viewing window.

5.3 Stereo Rendering of Complex Scenes

The rendering of artificial scenes in stereo mode is a feature typical for virtual reality applications. This process involves the generation of images from two slightly different viewpoints of the virtual camera. In order to speed up the rendering in stereo mode, in this section we present a novel approach for scene splitting in order to define two parts of the entire scene [Stoev et al., 2000a, Stoev et al., 2001b]. The one appears to be different for both eyes and has to be rendered in stereo (for each eye separately) typically including the area close to the observer. The second part of the scene typically includes the background and is rendered once, as seen through the lens of a middle camera. Hence, it is displayed in mono mode. In other words, the proposed technique exploits the spatial coherence of the two cameras involved in the stereo image generation.

One of the most important advantages of the presented approach is its suitability for combining it with alternative rendering methods. For instance, the mono background can be displayed applying image-based approaches, as well as ray-casting approaches.

5.3.1 Introduction and Related Work

Stereo projection techniques are widely used in virtual reality applications nowadays. Typical examples are the CAVE [Cruz-Neira et al., 1993], the responsive workbench [Agrawala et al., 1997], and the virtual table [Schmalstieg et al., 1999]. The aim of displaying stereoscopic images is to enhance the depth perception of the participants. Extensive research has been done in the area of stereo projection and depth perception in the past years [Wann et al., 1995, Rosenberg, 1993, Yeh and Silverstein, 1990, Yeh and Silverstein, 1992]. It has been shown, that the most important cues for depth judgment are:

- the dynamic flow, mainly achieved through tracking the viewer's head position and orientation, and setting the virtual camera properly,
- the perspective, relative object size, and occlusion,
- the scene lights and objects' shadows, and
- the stereo image generation.

Although, much depth information can be inferred from the introduced monocular depth cues alone, the stereoscopic depth cue provides additional information. It often enhances the speed and accuracy of tasks requiring depth perception [Rosenberg, 1993], as is typically the case in virtual environments.

The main problem of rendering large scenes with complex geometry is the computation time, independent of whether rendered in stereo or mono mode. Therefore, many authors made the attempt to combine the pure geometry rendering with geometry and either image based rendering [Wonka and Schmalstieg, 1999] and simplification [Darsa et al., 1998], warping [Rafferty et al., 1997], or even ray-casting [Wimmer et al., 1999b]. Although, these approaches seem to be very promising, none of them considers the stereo mode, which is usually applied in virtual environments.

The idea of combining real 3D objects with "flat" 2D background is not new. The answer to the question whether this approach is possible or not, was given already in the 18th century. One of the most prominent Venetian artists of that time was Giovanni Battista Tiepolo, who shaped the painting in the Residence in Würzburg (1750 - 1753). In his work, he achieved an outstanding combination of real 3D sculptures and 2D paintings (as shown in Figure 5.7). This "mixed world" gives a very

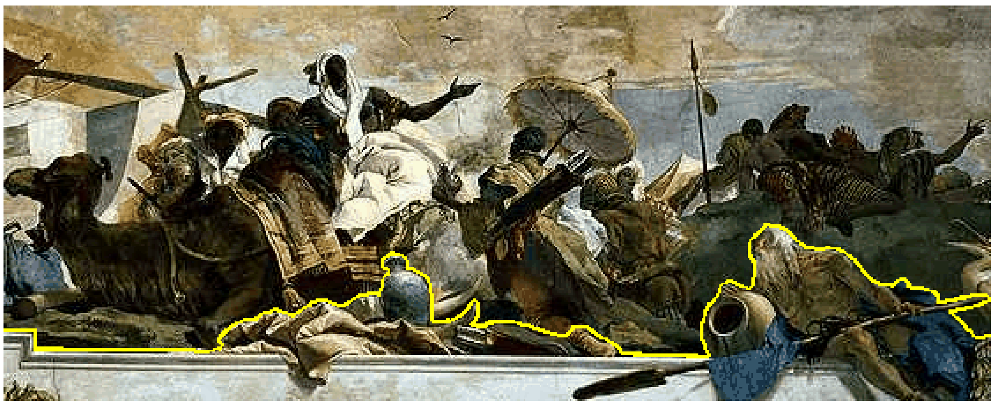


Figure 5.7: The segment "Africa" of the great fresco painting covering the ceiling of the staircase of the Residence in Würzburg. Here the combination of real 3D objects like sculptures and textiles in the foreground and a 2D picture background gives a perfect depth impression.

good depth impression, while showing most of the scene as a regular painting. Unfortunately, due to the static scene composition, this impression only works from a particular point of view. Hence, the main challenge of our partitioning method is the *dynamic* definition of the appropriate *scene partition criterion.* Before describing in detail the proposed approach, we first introduce the basics and the computer aided realization of stereovision.

5.3.2 Stereovision and Stereo Rendering

Most of the stereoscopic systems nowadays make use of the display of left and right images in order to create the impression of permanently perceiving one stereoscopic image [Hodges, 1992, Lipton, 1997]. This is accomplished either by displaying sequentially different images for the left and for the right eyes, or displaying simultaneously images visible only for the left, respectively only for the right eye. In the first case, the user is provided with a shutter that is synchronized with the display system. Whenever the image for the appropriate eye is displayed, the corresponding shutter is open and the eye sees the image. Due to the not perfectly closing shutter glasses and to the conceptual structure of this type display system, *crosstalk* may occur. Crosstalk in a stereoscopic display results in each eye seeing an image of the unwanted perspective view. In a perfect stereoscopic system, each eye sees only its assigned image.

In the second scenario, the images are displayed simultaneously, however using light polarized in different directions. In this case, the user is wearing glasses with polarized foil, which lets only the appropriate light pass to each eye. The effect in both cases is the same: the participant views two different images generated with two different (virtual) cameras. Finally, the last component of the stereo system is the human brain, which fuses two pictures into one with stereopsis.

In order to describe the process of generating stereo image pairs, we first introduce some specific terms. *Disparity* is the distance in a horizontal direction between the corresponding left and right image points of the (superimposed) retinal images. The disparity is caused by the fact that each of the eyes sees the world from a different viewpoint. This is the basis for stereo seeing. The *parallax* is defined in a similar way: *parallax* is the distance between left and right corresponding image points measured at the *display screen*. In other words, in contrast to the disparity measured at the retinal image, the parallax is measured at the display screen.

In a straightforward implementation of a stereo system, the two virtual cameras are positioned at a given distance from each other. This distance corresponds to the real distance between the human eyes (*eye separation*). Experimental studies [Surdick et al., 1997, Yeh and Silverstein, 1990] have shown that users with normal stereoscopic vision often have trouble fusing stereo image pairs if the eyes are modeled based on exact eye separation. Therefore, in practice, it turned out that an underestimation of this modeled eye separation [Wartell et al., 1999] might lead to better depth impression. Note that the quality of a pair of stereo images generated with differently modeled eye separation depends on the scene, as well as on the individual ability to perceive and fuse stereo images. Nevertheless, as a side effect, the underestimation also reduces the crosstalk, since the difference between the two displayed images is smaller. Therefore, the images appear sharper to the user.

Once the cameras for both eyes are placed at the new position, thus, the eye separation is modeled, they have to be rotated in such a way that the view direction of both eyes crosses at the focal point. When the object the human eyes are focusing on is close to the observer, the angle α between the view directions of both eyes increases (see Figure 5.8). In case the object is at infinity, the view directions are

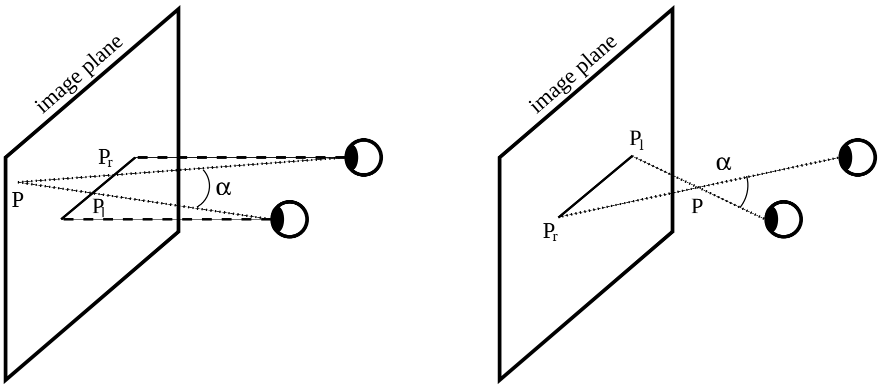


Figure 5.8: Positive parallax on the left: the point of focus P is behind the image plane. A negative parallax is shown on the right: the point of focus P is in front of the image plane.

(almost) parallel. This is the natural way of seeing stereo.

With the human visual system, objects in front and behind the focal plane appear blurred and doubled. This effect cannot be modeled in a synthetic display system. Instead, in order to make an as good as possible approximation, the orientation of the eyes is modeled. Unlike the natural human seeing, however, display systems provide only one fixed focal point. This means that the modeled view direction for both eyes is fixed, independent of the point on which the real eyes are focused. The only way to close the gap between the synthetic stereo and the real stereovision is to employ eye tracking. This will in turn allow the display system to provide the correct images for both eyes. Since this was not possible by the time this thesis was completed, we will not discuss the eye tracking in this work. For the remainder of this section we assume that the focal distance is always fixed until not explicitly (e.g.

manually) changed.

As introduced above, when the object at which the participant is looking is far away from the observer, the view directions of both eyes are almost parallel. In contrast, when viewing objects close to the user, the angle between the view directions increases. Similar to this scenario, and in order to achieve good depth impression for objects in front of the projection plane, a *negative parallax* is used when the displayed objects are in front of the projection plane (see Figure 5.8). Otherwise, when the displayed objects are behind the projection plane (this is often called *within the CRT space*), the parallax is positive.

After positioning and orienting the virtual cameras appropriately, each of them shoots a picture of the artificial scene. For this, a single projection plane is used for both virtual cameras (see Figure 5.9). Since there is only one display, this is the only reasonable way for generating pairs of stereo images. In this scenario, however, we face another problem: the projection plane is not orthogonal to the view direction of both virtual cameras. This means that the projection of the scene has to include a shearing component in order to render the artificial world appropriately. Fortunately, this feature is supported by standard graphics hardware and is easily implemented.

5.3.3 Partitioning the Scene

The approach we present in this section is based on the observation, that objects far away from the observer appear the same for both eyes. In other words, the parallax for such objects corresponds to the modeled eye separation. Note, that this is only valid when the focal point is not too close to the current viewpoint! If this case occurs, the parallax increases significantly with increasing distance to the viewer.

In typical stereo setups, the focal point is set to a default value, which is usually the middle point of the scene's bounding box (see Figure 5.9 upper diagram). In this way, half of the scene is in front of the focus plane, the remaining part is behind the focus plane. Another implicit assumption often made in stereo systems is that the focal plane and the image plane are the same. Decoupling the focal plane from the image plane, however, allows more precise stereo adjustment.

Apparently, the images produced for both eyes are the same in the focal plane, when the projection plane and the focal plane are the same. Otherwise, they are shifted among the distance $2S_p$ between the projections of the focal point viewed with the left and the right virtual cameras (see Figure 5.9). This is the more general case, and we refer to it throughout the remainder of this section.

Based on these observations, we have found that even though the disparity increases with increased distance between the viewer and the focal plane, the "far background" is not perceived as distorted when one and the same background image is displayed for both eyes. This is the main contribution of the proposed approach. In order to implement it, we render first the part of the scene, which is far away from the viewer with a "middle eye"[2]. Once this is done, the images are translated by S_p to the left and to the right, and copied into the left, respectively right buffer. Afterwards, the geometry close to the viewer is rendered twice: once for each eye.

In addition, our approach is not limited to geometry-based rendering of the background. Various other approaches (e.g. image based rendering) can be applied for

[2]This denotes the camera position before translating and rotating it in order to generate the stereo images for the left and the right eyes.

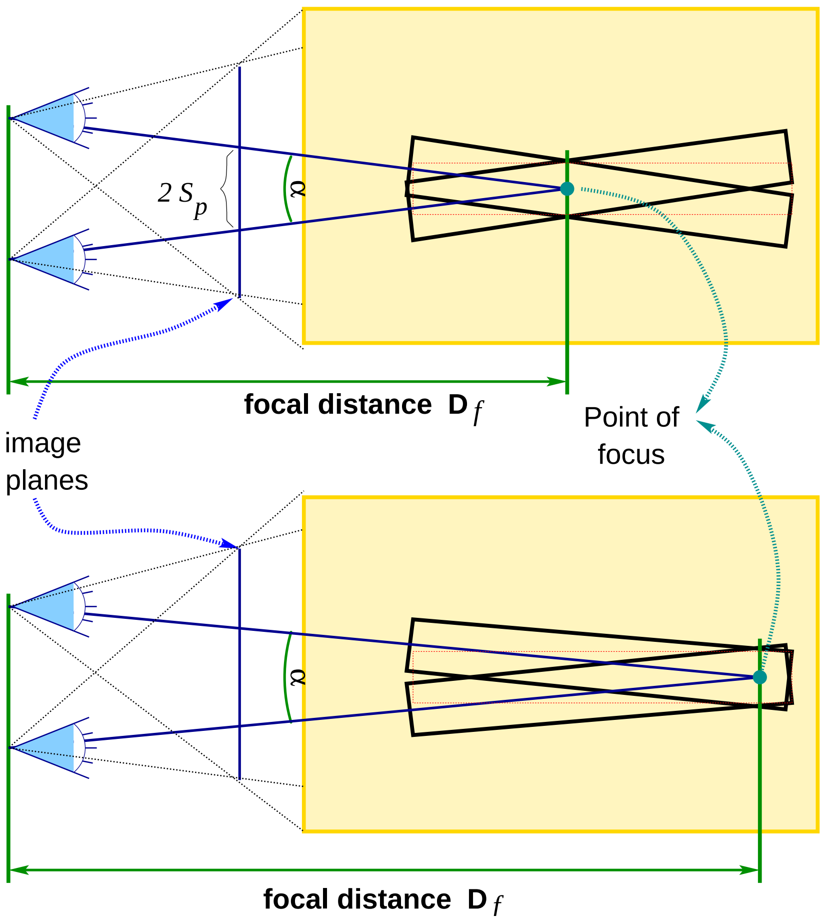


Figure 5.9: Focusing different points in the scene and the resulting stereo images. In the upper image, the default focal point is used (scene's middle point). In the lower image, the focal point is moved away from the user. The dotted object represents the real object, the solid lines show the image of the object generated for the left and the right eyes.

generating images of this part of the scene, which appears the same for both eyes. The resulting images are displayed for and seen by each eye, thus creating the impression of viewing true stereoscopic images.

Since the standard stereo-mode software does not allow direct realization of this idea, we had to perform the following steps, in order to realize the presented concept:

- Render the background seen by the "middle eye" into the provided buffers. This step is performed only once, whereas the same image is rendered (and shifted) into both buffers. These features are supported by the OpenGL standard [Woo et al., 1998].

- Render the eye-specific foreground, setting the appropriate camera position and orientation for each eye, *without* resetting the content of the image buffers *and* the depth buffers for both eyes.

5.3.3.1 Where to Split

In order to make the scene division imperceptible for the observer, the scene has to be partitioned in such a way, that there is no noticeable difference between the pure stereo projection and the stereo foreground projection combined with the mono display mode (at least where the stereo and mono areas meet). This is achieved through defining a splitting plane orthogonal to the observer's view direction and intersecting the point of focus. Since the images for both eyes are the same for objects in this plane, beginning with the monoscopic part of the scene at this plane allows us to imperceptibly switch between these two modes. The realization of such a smooth transition can fool the human brain in its depth perception.

The most challenging task for the scene partitioning is the suitable definition of the focal distance D_f (see Figure 5.10). Since this cannot be measured and the ex-

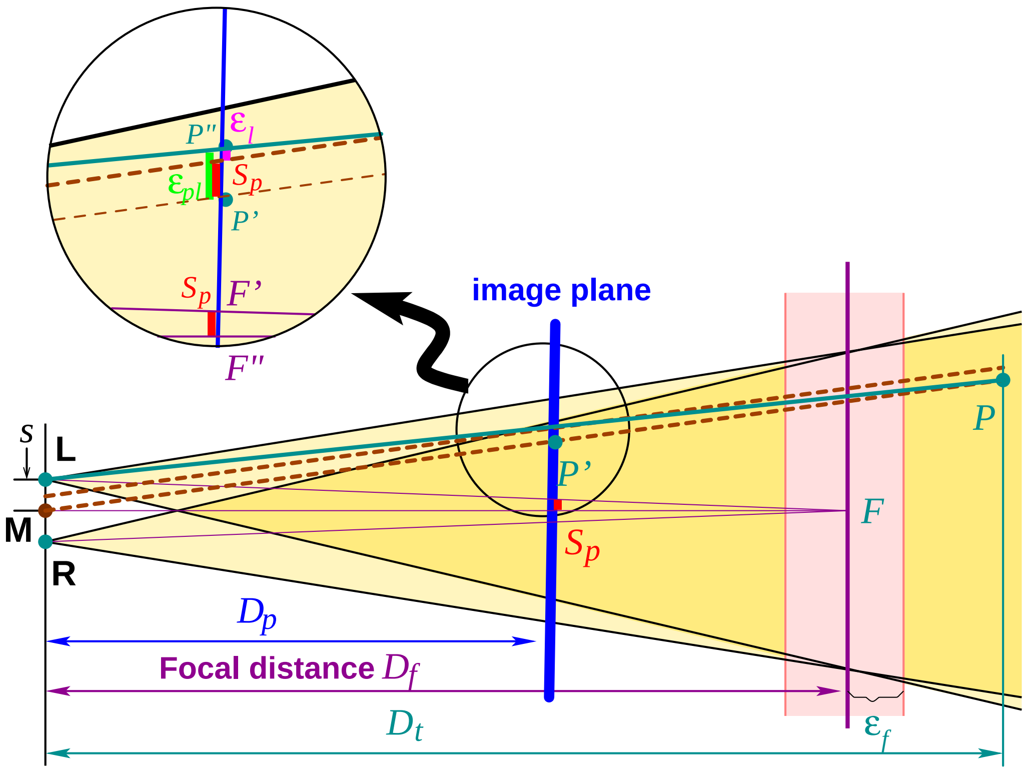


Figure 5.10: Projecting P with the left camera and the middle camera results in P' and P''. After shifting the projected point P' among the factor S_p, there is still an error ε_l compared to the position of P projected with the left camera P''. In order to make the switching of objects from the foreground to the background and vice versa imperceptible for the observer, this error ε_l has to be under a given bound defined by the resolution of the display screen.

act view direction cannot be tracked in traditional stereo applications, the value of D_f is usually set to the distance between the default viewer position and the middle point of the scene's bounding box. For values significantly smaller than this value, *diplopia*[3] occurs and the 3D depth illusion collapses [Yeh and Silverstein, 1990, Southard, 1995]. On the other hand, too large values produce almost equal images and create poor depth impression. For our setup we empirically determined, that the value of the doubled distance between the viewer and the projection plane is for most of the scenes and most of the users a good approximation. Since the observer's head is tracked and the projection plane has a fixed position, the initial distance D_p between the observer and the display is easily determined. This value is now dou-

[3]The images are not fuseable for the observer.

bled in order to compute the final fixed[4] focal distance D_f.

Depending on D_f and the (limited) resolution of the projection display, we can now define not only a plane, but an interval ε_f in front and behind the plane D_f, in which the objects have the same image when projected with the left (or right) camera and the middle camera. Apparently, the larger the distance between the viewer and the focal plane, the larger the value of ε_f, the more objects fit in this region (see Figure 5.11).

This approach is known from the multi-resolution area, where the screen resolution is taken into account to determine which approximation error is below a given bound. Applied to our scenario, this idea defines for which objects, at which distance from the viewer, a given (error perception) bound is overstepped and the user will see distorted images. In particular, the computation of ε_f depends on the participant's (eye) position, the focal distance D_f, and the distance to the projection screen D_p. Hence, it can be calculated at which depth value the pixel difference between the images for the left, the right, and the "middle eye" (positioned between the modeled eyes) are below a given pixel error.

Assuming that D_f has a fixed value, we first determine the offset S_p. This is the value used to shift the image of the point of focus F projected with the middle camera F' to it's position when projected with the left camera F'' (the same for the right camera), as depicted in Figure 5.10. Denoting with s the half of the eye separation, S_p is defined by:

$$S_p = \frac{(D_f - D_p)}{D_f} \cdot s. \tag{5.1}$$

We now consider the projection of the point P onto the image plane as depicted in Figure 5.10. The projection of P seen by the middle eye is the pixel P', which is shifted among the value of S_p in order to determine its final position in the image produced for the left eye (see Figure 5.10). Now, we can compute the error ε_l for the pixel P when rendered in the left eye buffer, with D_t denoting the distance between the viewer and the point P:

$$\begin{gathered} \varepsilon_l = \varepsilon_{pl} - S_p, \quad \text{with} \\ \varepsilon_{pl} = \frac{(D_t - D_p)}{D_t} \cdot s = \frac{(D_f \pm \varepsilon_f - D_p)}{D_f \pm \varepsilon_f} \cdot s, \end{gathered} \tag{5.2}$$

denoting the distance between the projections of P generated with the left (P'') and the middle (P') cameras.

With the equations 5.1 and 5.2 we can write the interval ε_f as a function of the focal distance D_f and the display distance D_p, denoting with ε_r the resolution of the display:

$$\frac{(D_f \pm \varepsilon_f - D_p)}{D_f \pm \varepsilon_f} \cdot s - \frac{(D_f - D_p)}{D_f} \cdot s \leq \varepsilon_r. \tag{5.3}$$

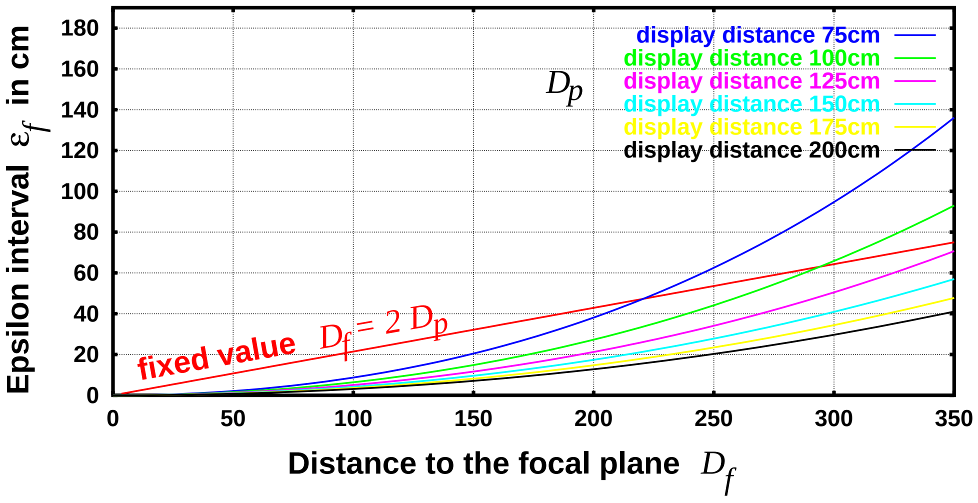
Transformed for both, pixels in front and behind the focal plane, this equation results in:

$$\begin{gathered} \varepsilon_f \geq \frac{-{D_f}^2 \cdot \varepsilon_r}{D_p \cdot s - D_f \cdot \varepsilon_r}, \\ \varepsilon_f \leq \frac{{D_f}^2 \cdot \varepsilon_r}{D_p \cdot s - D_f \cdot \varepsilon_r}. \end{gathered} \tag{5.4}$$

[4]Even though, we could vary the focal distance depending on the head position with respect to the display, this implicit variation of D_f may cause significant confusion to the user.

These equations define a region, where the scene-partitioning plane has to be situated. In this region objects can be displayed in stereo mode, as well as in mono mode, without causing distortion in the stereo perception. For our setup, we utilized the front plane defined by equation 5.4, since in this case the smallest possible part of the scene is rendered twice in stereo mode.

In Figure 5.11, we have shown this dependence graphically, displaying ε_f as a



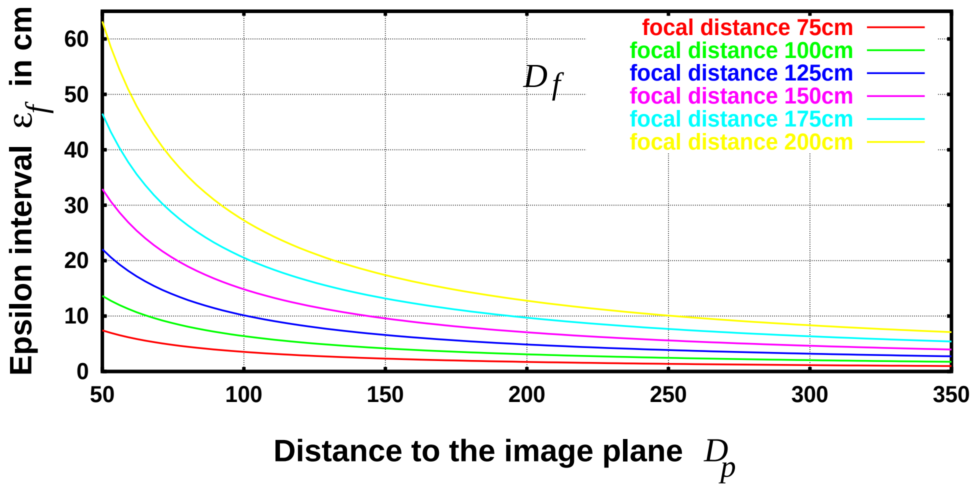


Figure 5.11: Upper diagram: ε_f as a function of D_f. The distance to the display D_p varies from 75cm to 200cm. In the lower diagram, ε_f is shown as a function of the distance to the image plane D_p. All functions are drawn for the exact solution of the as *equations* interpreted inequation 5.4.

function of the distance to the focal plane D_f for several distances between the viewer and the projection plane D_p. Note, that the half eye separation is constant and is set to 2.0cm, since it has been shown in the literature, that an underestimated eye separation enhances the depth perception. The screen resolution ε_r we set to $\approx 0.12cm$, which corresponds to our screen with resolution 1024x768 pixels and size 120x100cm. For $D_f = 200cm$ and $D_p = 100cm$ this results in the interval $[-22cm, 22cm]$.

As shown in Figure 5.11, the value of ε_f varies depending on D_p and on D_f. The upper diagram shows that when the viewer moves toward the projection plane, the interval ε_f gets narrow. When moving away from the projection plane it increases. The lower diagram shows that the greater the focal distance and the smaller the distance to the image plane D_p, the greater the value of ε_f. Thus, the greater the difference between the values of D_p and D_f, the greater the interval defined by ε_f. Therefore, we defined a minimum distance to the projection plane (in our scenario 75cm). This value we use to compute a $\min(\varepsilon_f)$, which is applied for computing the maximum size of an object to guarantee the object's smooth motion from the stereo foreground to the mono background and vice versa. Because each scene contains objects larger than this allowed size, the geometry of these objects is split in a preprocessing step. For accomplishing this task, we applied the tools described in [Optimizer, 1997, Meißner et al., 1999]. When this preprocessing is completed, we can guarantee, that no geometry object has a bounding box with diagonal larger than ε_f, thus all objects can be imperceptibly switched from the foreground to the background and vice versa.

5.3.3.2 The Stereo Frustum

In addition to the plane parallel to the projection plane, four planes defining a stereo frustum within the viewing frustum are introduced. The default stereo frustum is the same as the viewing frustum. However, if the stereo setup provides eye tracking, the stereo frustum can be used to display only a small part of the scene in the view frustum in stereo. In this way, we can cull out objects, which are not in the viewing volume as shown in Figure 5.12.

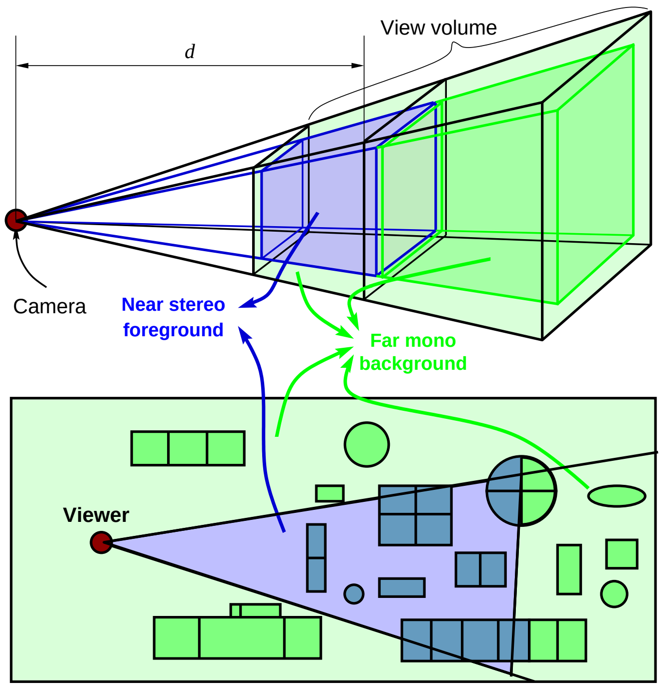


Figure 5.12: The stereo/mono frustum and a 2D scene example. d denotes the distance to the splitting plane.

During the scene partitioning, every object detected to be completely *outside* the near (stereo) frustum, is assigned to the background and rendered once. Otherwise, if an object is intersected by one of the planes, or is between them, it is assigned to the foreground. This allows us to speed up the rendering time, since only a small part of the scene passes the rendering pipeline twice.

5.3.3.3 How to Split

In order to make the scene partition imperceptible for the observer, the dynamic scene splitting process triggered before every rendering step has to be performed as fast as possible. Therefore, for each geometry object we save additional information containing the object's bounding box and the Id of the mode (near stereo foreground or far mono background) it has been assigned to in the previous rendering step. This allows us to circumvent expensive reassignments of an object, which is already in the appropriate mode. Thus, only objects, moving from the near stereo foreground to the far background and vice versa are processed.

Since our mixed-mode-viewer is based on the widely used *OpenInventor* standard, the data is organized in scene graphs [Wernecke, 1994]. We provide two graphs for the two parts of the scene. The first contains the scene's stereo foreground, which is rendered twice. The other scene graph, contains the mono background data and is rendered only once.

In order to determine whether an object is flipped from one graph into the other as the observer moves through the scene, we check the intersection of the object's bounding box with the clipping planes introduced above. For this computation it is sufficient to check the intersection of the four cube diagonals with the current cutting plane. For each diagonal we determine the sign of the scalar product between the vector $\vec{A}$ and the (cutting) plane normal vector $\vec{N}$ respectively the vectors $\vec{B}$ and $\vec{N}$ (see Figure 5.13). This corresponds to determining the angles α and β as shown in

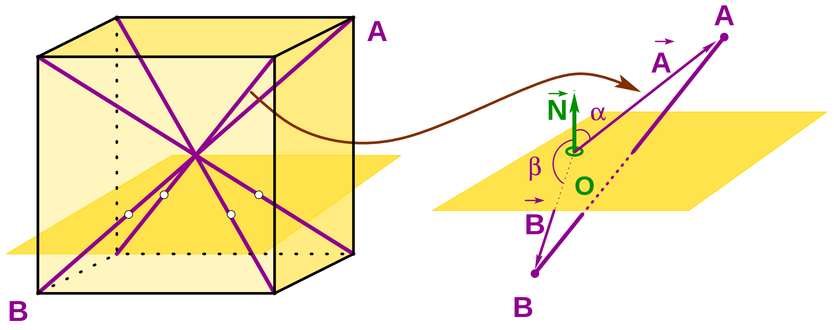


Figure 5.13: Computing the intersection between a bounding box and a cutting plane. For each diagonal on the left, the angles α and β have to be computed.

Figure 5.13. If one of the angles is below $90°$ and the other greater, the cutting plane intersects the diagonal. Thus, the object is assigned to the stereo foreground. Otherwise, if both points A and B are in the same half space defined by the cutting plane, the object is either assigned to the stereo foreground (if both points are *inside* the stereo frustum), or to the mono background.

For speeding up this entire process, the scene graph is modified after it is loaded. When the scene is initially read, we copy the scene graph, keeping only the geometry nodes once in the memory. This saves memory and computation time compared to completely copying the scene graph, since the geometry nodes require the most memory and contain the actual data. Afterwards, we replace each *link* to a geometry node with a *switch node*, as shown in Figure 5.14. The switch nodes are special In-

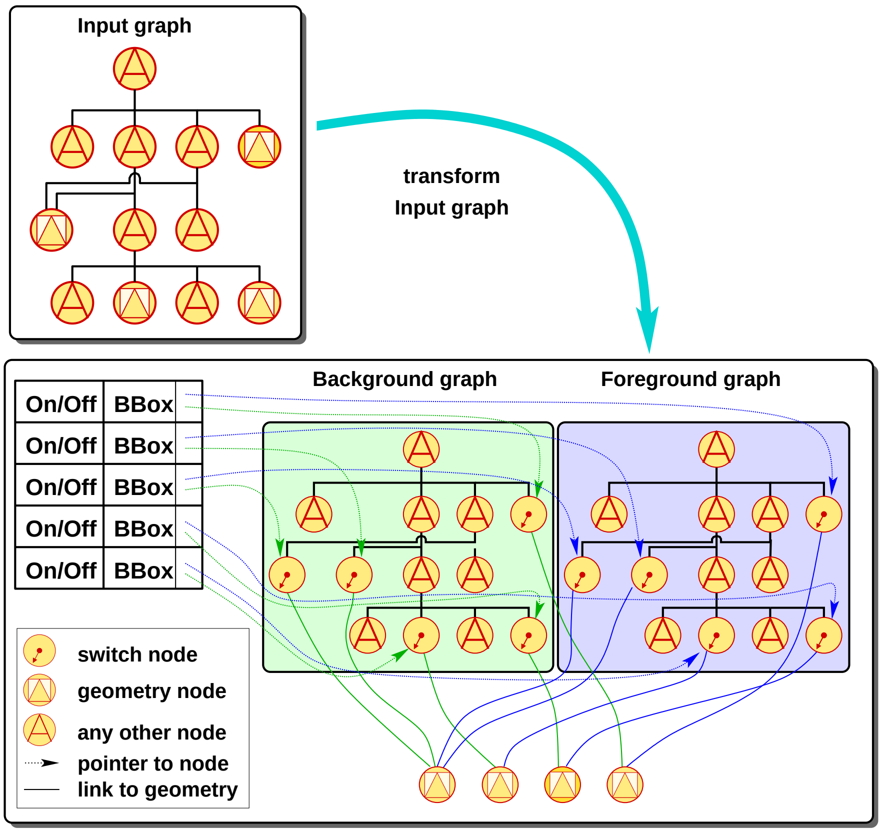


Figure 5.14: Replacing the links to the geometry node with switch nodes.

ventor/VRML/Performer nodes, which can be used to change the visibility of the underlying geometry in the scene graph. Hence, when an object moves from one mode to the other, only the value of the corresponding switch node is changed. Storing an array of pointers to these switch nodes makes it possible to traverse all geometry nodes before each rendering step. The performance results achieved in this way are discussed in the next section.

5.3.4 Performance

Before discussing in detail the performance, we briefly introduce the application of the proposed approach. The method described above is integrated in a dedicated OpenInventor-viewer, used to explore modeled indoor scenes, as well as terrain data. Since it has been shown in the literature [Padmos and Milders, 1992, Olano et al., 1997], that *latency* plays an important role in *VR*-setups, we compare the times for rendering the test scenes with our approach and with the standard stereo rendering approach.

The main latency source nowadays is still the rendering of complex scenes. In our setup, an additional delay source is defined by the scene-partitioning step. This includes the computations of: (1) the cut-planes' intersections with the bounding box of each scene object and (2) the scene graph traversing in order to assign the appropriate switch node's visibility for each object.

In order to measure the quality of the proposed method, we performed walkthroughs in two indoor scenes containing approximately 400'000, respectively 130'000 triangles. During the walkthrough, we recorded the rendering times for the stereo foreground, consisting of the time for rendering the scene's near foreground in each eye's buffer and the rendering and shifting of the background into both buffers (see Figures 5.15 and 5.16). In addition, the time for computing the intersections with the cutting planes and setting the appropriate flags in the scene graph are displayed in Figure 5.15 and Figure 5.16. In the pure stereo mode, the rendering time

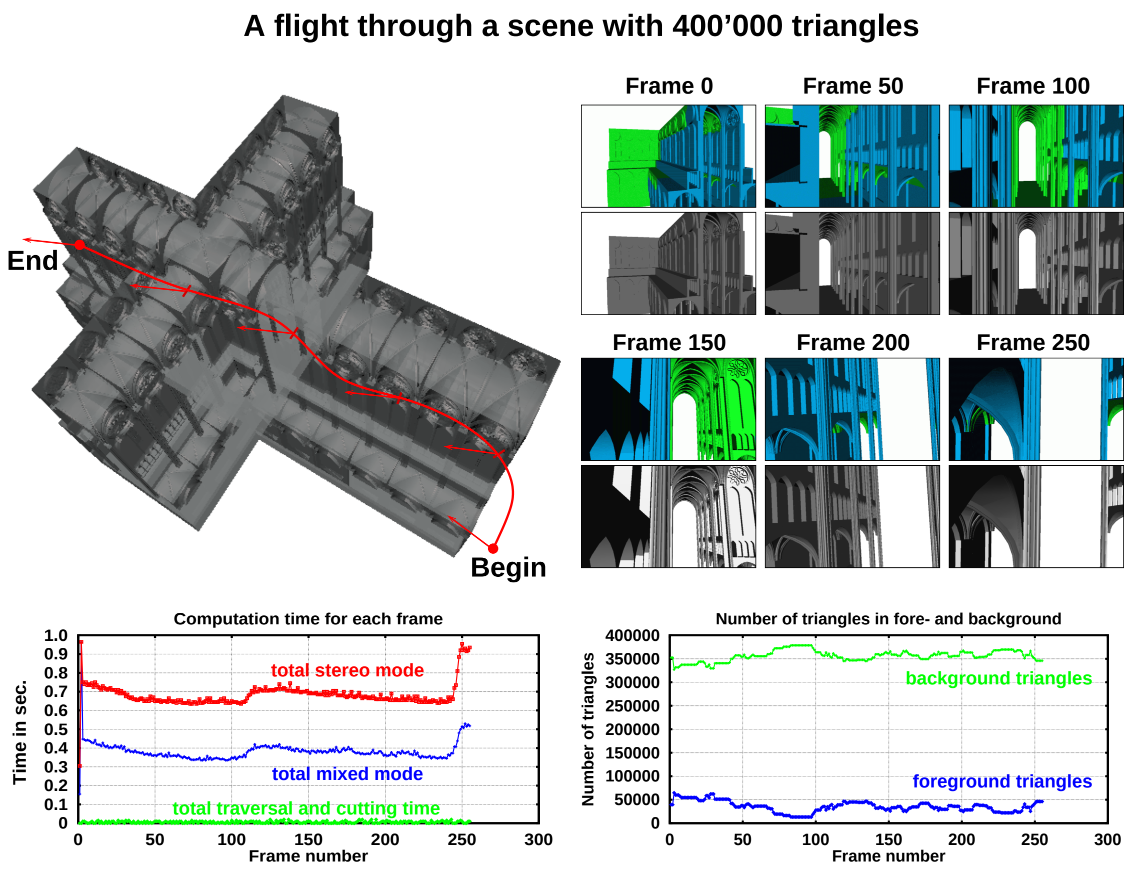


Figure 5.15: Statistic of the path through a scene with 400'000 triangles.

includes the time for rendering the entire scene twice: once for each eye. Furthermore, for each scene we recorded the number of triangles displayed in the stereo foreground and in the mono background. As shown in Figures 5.15 and 5.16, the proposed approach is especially useful when the scene is large and the geometry is regularly distributed. This is the case in the first example (Figure 5.15). In the lower right diagram, we see that only a small part of the scene is rendered in the foreground. In contrast, the second scene was small with many object concentrated in the middle of the scene. This explains the sudden drop of performance and increase of foreground geometry in the middle of the left and right diagrams of Figure 5.16. In this scenario, the user navigated into the scene and turned around before flying out of the scene again.

Nevertheless, Figures 5.15 and 5.16 show that our approach of dynamic splitting the entire scene in once rendered background and twice rendered foreground achieves significant speedups of up to 40% compared to the traditional stereo rendering. More important, however, is the potential of combining the proposed technique with image base rendering algorithms. This will significantly decrease the rendering time for the background area of large scenes with complex geometry and

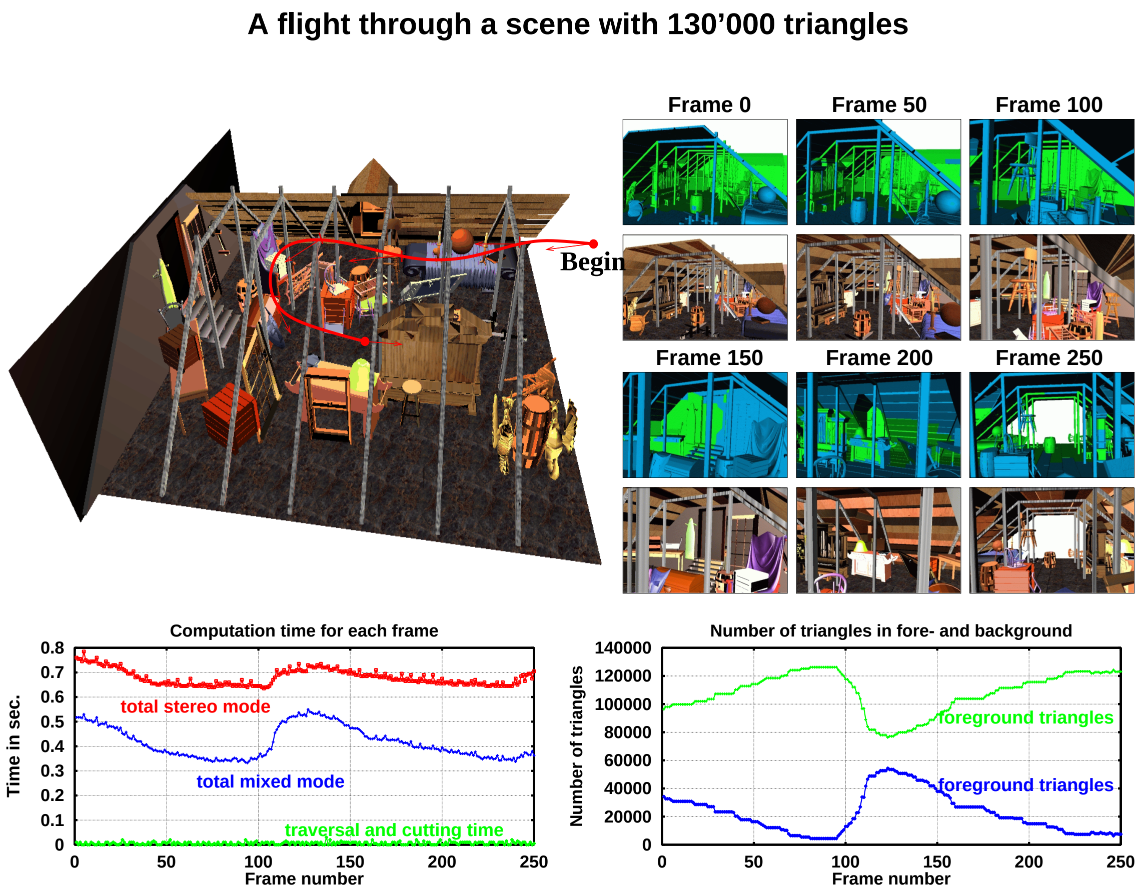


Figure 5.16: Statistic of the path through a scene with 130'000 triangles.

hence increase the frame rate.

5.3.5 Usability

The stereo perception is a very subjective perception, and thus it is very difficult to measure the applicability of the proposed approach in practice. We performed a series of usability tests in order to present, even though only vague, usability measurement. For accomplishing this task, we let eleven users walk through two different scenes and perform given tasks, while recording and evaluating their depth judgment. The participants' experience with stereo projection setups varied from no experience to very experienced users.

The setup for all tests was based on a large screen projection: the *Virtual Table*. The users were seated in front of this device, approximately 150cm away from the display with resolution 1024x786 pixels and size 120x100cm. The interocular distance in all trials was set to 4cm, the focal distance to 200cm.

5.3.5.1 Depth Judgment

The aim of the first trial was to show that our approach is applicable, that is, that the human depth perception is not distorted when the viewed scene is partitioned properly and displayed as described above. For this, we let the participants perform two walkthroughs of each scene: once in each display mode. Thereafter, we asked whether or not there was a perceptible distortion while viewing the scene.

After evaluating the recorded data, it turned out, that 64% of the participants did not recognize any difference between both display modes at all (see Figure 5.17). 18% of the participants even rated the *mixed* mode as being more pleasant to view

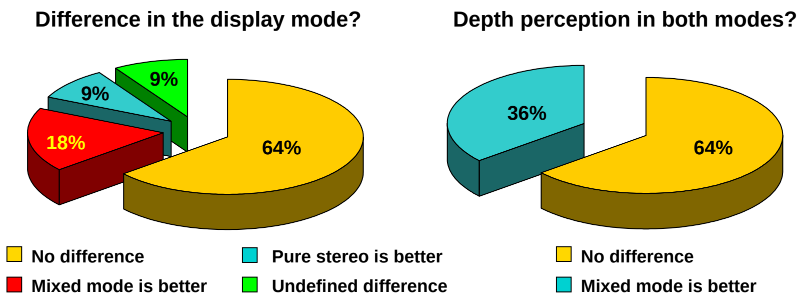


Figure 5.17: Results of the usability tests after displaying the same scene in both modes.

than the normal stereo mode. 9% of them stated, that there was a difference but could not describe it and the remaining 9% of the participants found the normal stereo mode better than the *mixed* stereo/mono mode. None of the subjects complained about headache or sickness.

Being asked about their depth impression, 64% of the participants did not find any differences in the depth of both modes (see Figure 5.17). The remaining 36% had an even better depth impression viewing the scene in the *mixed* stereo/mono mode, than in the true stereo mode. This can be explained with the only once rendered, thus, sharper background part of the scene. In addition, the effect of so called “ghost images”, often caused by shutter-glasses-based stereo setups (i.e. not perfect closing shutter glasses), was significantly reduced.

5.3.5.2 Interaction

In the second test, we let the participants reposition given objects in the scene, while recording the time they needed to perform this task. For this, we utilized the same setup as described above. This time the users were asked to manipulate an object in the scene, such that the object was moved once from the stereo foreground into the mono background (Figure 5.18 A to B), then from the mono background back into the stereo foreground (B to C), and finally within the stereo foreground (C to D and D to A). Every participant performed this task two times, once in the normal stereo mode and the once in the *mixed* stereo/mono mode. In order to achieve comparable interaction results and not navigation dependent measurements, the viewpoint was not allowed to be moved in space. Only the head tracking was still active.

This test showed that all participants needed approximately the same time for moving the given object to the marked target points until the given precision is reached independent of the rendering mode (see Figure 5.19). Hence, we can conclude, that the mixed mode did not affect the interaction time for performing the requested task in the virtual environment. Considering the precision, we also found out that the accurate positioning does not suffer from the “flat”-mono display of the background. In both trails the positioning depended rather on the distance to the target location, than on the display mode. In other words, it was more difficult for the participants to position the object at distant targets independent of the rendering mode (comparison of points B and D). Moreover, none of the subjects recognized

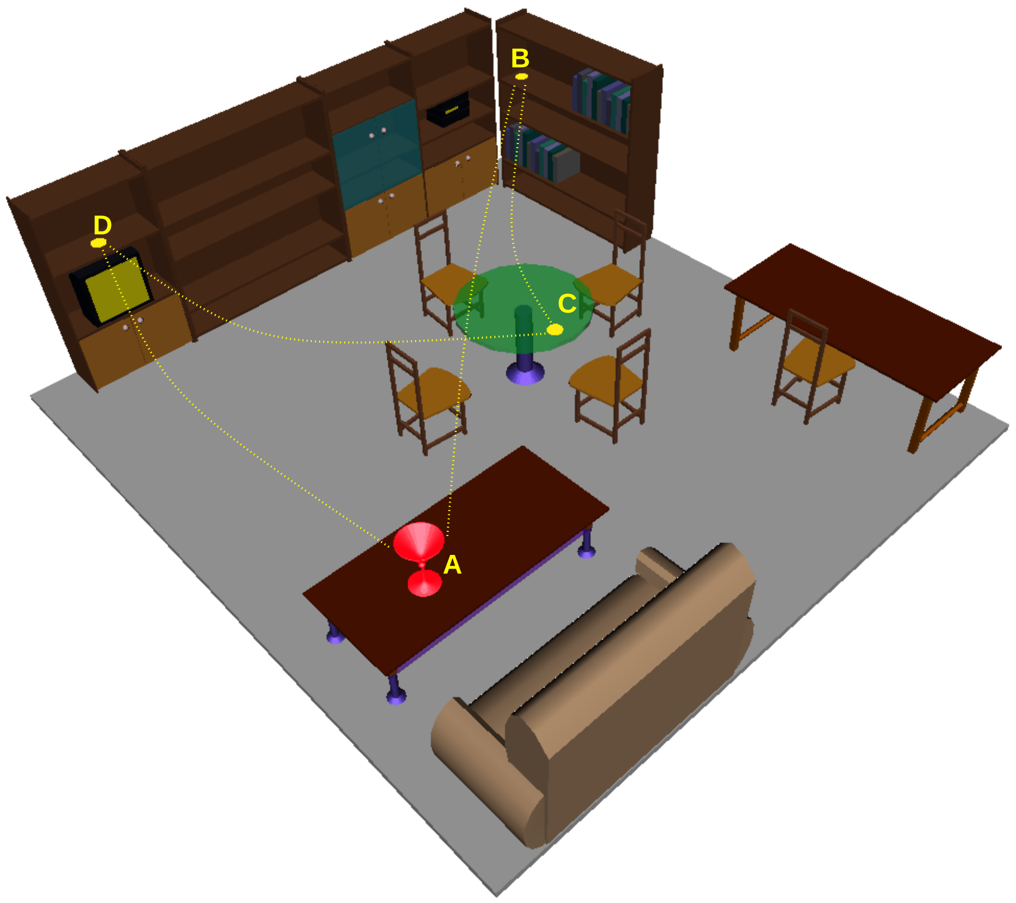


Figure 5.18: The marks A-D show how the participants moved the wine glass (at A) in the scene during the second trial. The target locations A, C, and D are in the foreground, the position B was in the mono-rendered background.

any disturbing effects when the picked object was "switched" from stereo foreground to the mono background and vice versa.

During the tests, special attention was paid on the users fatigue, perception illness, as well as the participants' orientation in the scene. As a result we ascertained, that the mixed display mode did not cause any loss of orientation, neither did it affect the precision and the time the participants needed for performing the requested tasks.

5.3.6 Extensions and Conclusions

An important research direction towards improving the applicability of the proposed algorithm will be the integration of an eye-tracking device in our system. Eye tracking will provide the perfect completion for the proposed approach. In such a setup, we will be able to exactly determine the point of focus and display only this part of the scene in stereo mode, which is close to it. Additionally, the stereo frustum can be narrowed such that only the area the user is looking at is displayed in stereo mode.

The presented approach is also suitable for displaying the scene applying two output devices. For instance, the near foreground can be displayed using a head mounted display, the far background utilizing a large screen projection. Such a scenario will allow integrating real objects into a virtual scene. Knowing the distance to

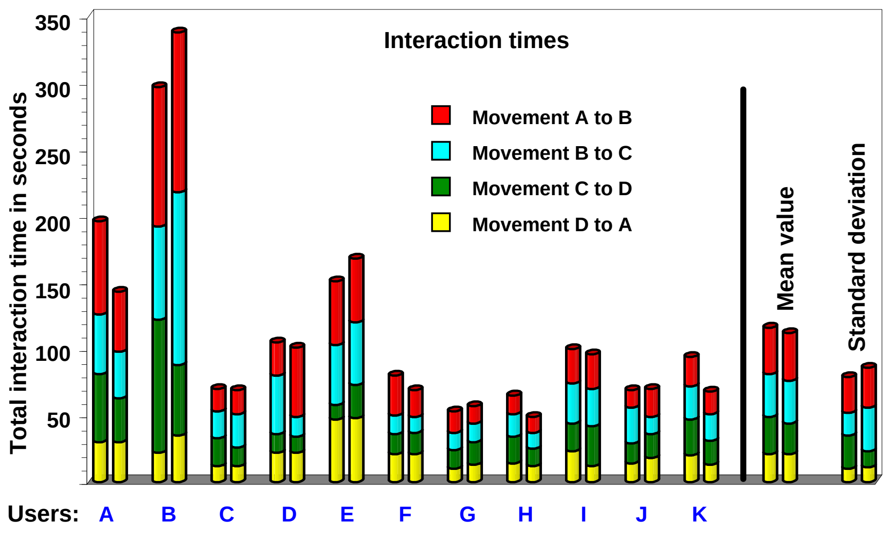


Figure 5.19: The times measured during the participants moved the object from A to B, B to C, C to D, and D to A (depicted in Figure 5.18). For each participant, two bars are displaying the time needed to perform the movements in mixed mode (left bar) and true stereo mode (right bar).

the real object will enable the scene to be split in such a way, that the foreground between the observer and the real object is projected via the head mounted display. On the other hand, the far background, which is behind the real object, can be displayed via large screen projection.

Even though the proposed approach cannot be combined with eye tracking yet, it can be applied for accelerating rendering of large scenes in stereo mode, typically applied in virtual reality applications. The discussed scene partitioning is based on the fact that distant objects appear the same for both eyes, making the presented approach especially suitable for large scenes with detailed background. We have introduced a definition of a criterion for partitioning a VR-scene is such a way, that the near foreground can be displayed in stereo mode and the far background in mono mode, without affecting the subjects' depth perception. This strategy allows us to integrate new techniques for rendering the scene's mono background (e.g. image based rendering) and to further speed up the rendering process.

In addition, with the discussed usability tests, we have shown, that the presented mixed approach does not cause any distortion of the 3D depth perception. It also does not affect the interaction time and precision in virtual environments, while reducing the rendering time up to 40%.

Chapter 6

Summary and Conclusions

In the author's opinion, diverse applications would benefit from being embedded into a virtual reality setup. Such an environment not only provides vivid data display, but it also allows real three-dimensional interaction and visualization. Presently many developers still do not dare to perform this step due to the unreliable hardware, as well as inadequate interaction, visualization, and rendering tools regarding the software. This work addresses various components of the application creation pipeline, considering the software aspects of a virtual reality application system. These components include three distinct groups: the rendering approaches, the interaction techniques, and the data itself. Considering the data, we can in turn identify three additional subgroups: the acquisition, the preprocessing, and the visualization. Throughout this work, we were motivated by the lack of adequate tools and algorithms in each of the above categories tailored to specific problems, as well as regarding some of these areas in general.

Considering the data-components, we first have discussed the preprocessing and preparation. In this area, the most challenging problem was the segmentation of data of interest out of volume/image data. Up till now, there are no reliable approaches for solving this problem. Even worse, most of the segmentations in today's image processing applications are still made by hand.

We proposed an algorithm based on the watershed transformation, which, in contrast to the traditional watershed transformation, is based on the *local* extraction of regions of interest. The proposed approach extracts locally basins of interest applying a rain-falling simulation. Due to the typical over-segmentation produced by the watershed transformation, a basin merging step has to be applied to the extracted basins. Many authors have published various strategies for merging the basins extracted during the watershed transformation. Unfortunately, all presented approaches suffer from their global application aiming to extract all features in the input image/volume data. Instead of developing sophisticated global criteria, valid for all basins in the entire image/volume data, we presented a method for deriving rules meeting local conditions. Thus, achieving more precise definition of the merging criteria compared to the traditional watershed algorithms. In this way, we retrieve good segmentation results, which are the basis for further visualization and exploration applications.

Besides the processing of image/volume data, we have also paid attention to the visualization of historical data, which is still a challenging task due to the included time dimension. Although we are able to visualize almost any type of data nowadays, the particular challenge in this scenario is that we have to deal with data, which components may vary in the time *and* in the space. In particular, when we consider a migration of people within a period of time, we have to visualize not only the position at a given time, but also various parameters like the number of the participants, the mood of the people, how far they could see from their current location, etc. These parameters are especially important for the analysis of the data. They help us to understand why some decisions are made and to explain the behavior of the participants in given situations.

In Section 3.3, we have described how the data is acquired and visualized in order to enable vivid display of various dimensions of the original data. We have demonstrated the entire process utilizing a particular dataset: the peasant's war in 1525 in Germany. With the proposed display techniques we are not only able to encode different parameters, (e.g. affiliation, mood of the troops, number of the participants, etc.), but we also presented various visualization aids enabling better orientation in the space and time dimension and thus allowing for better analysis of the data. The described tools, which were developed in cooperation with the historical department of the University in Stuttgart, have proved to be a valuable companion offering significant support for the analysis of historical data. Furthermore, the described visualization greatly improves upon the visualization of historical data, compared to the maps commonly used in the historical area.

In the second part of this work (Chapter 4, p. 47), we discussed in detail the interaction in a virtual environment based on the well-known personal pad and interaction pen (see Section 2.3.2, p. 11). First, we addressed the interaction problem in general and presented a set of new interaction techniques for virtual environments. Thereafter, we proposed a set of tools particularly tailored to the interaction with historical data.

Although many different paradigms and interaction tools are known in the literature, there are still problems associated with each of these tools. In order to overcome their limitations, while still supporting the features of these tools, we introduced the *through-the-lens* (TTL) concept, applied mainly as an extension to existing navigation techniques and used to develop new interaction tools. The basic philosophy of this metaphor is that there are two virtual worlds existing in the same physical space. One of these worlds is interactively explored and surrounds the user. This is also called the *primary world*. The other world, also called the *secondary world*, is explored using a dedicated window that acts like a wormhole connecting both synthetic worlds. There are various different states of the secondary world with respect to the primary world. In Section 4.2 (p. 51), we discussed a taxonomy for these states, as well as a classification of the states of the window connecting both worlds.

With the through-the-lens metaphor presented in this work, a secondary viewpoint in the secondary world additional to the primary (current) user viewpoint in the primary world is defined. This viewpoint allows observing and traveling to any arbitrary location of the secondary world. Moreover, the secondary world can be set to be a transformed copy of the primary world. In this way, the same synthetic scene can be explored with the through-the-lens tools, either by manipulating the viewpoint in the secondary world or by transforming the secondary world itself. Although the result of these two strategies is the same, they differ in the way of the application. This is done without having to perform any (viewpoint) motion in the user surrounding virtual environment– primary world. Due to the fact, that the position of the participant in the surrounding environment remains unchanged, the through-the-lens tools help to overcome typical problems of other interaction and navigation techniques like the caused disorientation, motion sickness, fatigue, etc. Since the navigation and the interaction are responsible to a great degree for the acceptance and the usability of a virtual reality application, the development of adequate tools for performing these tasks is of great importance.

Based on the above through-the-lens concept, we developed various tools for performing different tasks in a virtual environment. Although with the proposed tools the secondary world is an interactively transformed copy of the primary world, the through-the-lens tools can be applied in the same way in a scenario, in which the secondary world is an arbitrary synthetic scene. On the other hand, for the navigation of an artificial scene the secondary world and the primary world have to be the same, since the secondary world offers a form of a preview for a given location in the surrounding environment. This holds for the remote object manipulation and the interactive visualization of historical data as well:

Navigation: The through-the-lens concept was applied for navigation within a virtual world, where the user interactively sets the secondary viewpoint to any arbitrary position. This adjustment of the secondary world was realized with three different viewpoint manipulation techniques: the TTL-scene-in-hand, the TTL-eyeball-in-hand, and the TTL-world-in-miniature techniques. The result is a set of tools for coarse navigation, as well as high-precision adjustment of the viewpoint in the secondary world. These tools offer high degree of interactivity, while not exposing side effects like the original implementations, which strongly require a mental model in order to apply them appropriately.

Remote Object Manipulation: The through-the-lens concept was also extended towards enabling remote object manipulation in a virtual environment. The window, through which the remote location is viewed, is used in this case to reach through it and manipulate objects that are out of the physical reach of the user in the primary world. An important advantage of this technique is that the remote objects manipulated through the viewing window are manipulated in their natural environment at freely chosen scale, as if they were within the physical reach of the user. Furthermore, this tool can be also applied to manipulate a magnified copy of the primary world within the physical reach. Hence, the proposed approach enables precise manipulation of objects in the (primary and thus in the secondary) world.

Exploration of 4D Time-containing Data: Finally, we presented a set of through-the-lens tools for visualizing and interacting with data containing a time dimension (*TTL-time-tools*). To this set count the *time lens*, the *fly-with* tool, as well as the tools for interactive traveling the 4D time-space. The main characteristic of these tools is that they exploit the features of the above TTL-tools for navigation, while providing additionally the capability to travel the time dimension. They help the user to explore the time and space simultaneously. In particular, we implemented tools, which support the integration of expert data. In order to facilitate the exploration and to guarantee that the user will not miss important features of the data, an expert user can predefine motion in time and space. This is employed either to guide the user through the 4D data, or to display the data seen from the predefined path in the time and space as secondary world behind the through-the-lens window.

All these interaction techniques have in common is that they provide a preview-like window connecting the primary and the secondary world to each other. In particular, the secondary world (and eventually another event time) is displayed in a dedicated window in the primary world. This feature helps to circumvent the typical “loss of orientation”-problems in virtual environments. In case that the aim of the through-the-lens tool is to navigate within an artificial world, the secondary and the primary worlds are the same. Otherwise, the proposed tools offer a form of telepresence in two synthetic worlds simultaneously. Our informal usability tests with ca. 50 participants have shown that the TTL-metaphor is easy to understand and to apply even for users who do not have experience in interacting within virtual worlds.

In Chapter 5, we have addressed the rendering in virtual environments. First, we have considered a TTL-specific problem: the multiple rendering of the data as primary world and as secondary world(s). Even though the observer sees only a small part of the secondary world(s) from viewpoints with only little variation of the position and view direction, the entire secondary scene passes through the rendering pipeline. If the user applies multiple windows for exploring the input data, this means that for each window the complete data is rendered one more time. In order to accelerate the process of rendering the secondary world, we presented a new image-based approach tailored to this specific problem: the multi LDI. The multi LDI is an image-based scene representation, which allows dynamic data generation, fast rendering of the resulting data structure, and behaves like a geometry rendered scene considering the addition and the removal of objects after the multi LDI was generated.

We have also shown that the multi LDI may significantly increase the frame rate especially when multiple windows into the secondary world are defined and when the scene contains complex geometry. This makes the multi LDI a valuable extension of the through-the-lens tools presented above.

Finally, we addressed the stereo rendering of complex scenes. In the traditional scenario, the scene is rendered twice each time with slightly shifted and rotated virtual cameras, representing the observer's eyes. Instead of doing this, we proposed a method for splitting the scene in two parts: the near foreground, which is seen through each eye in a different way; and the far background, which is similar for both eyes, when the eyes are focused at a distant point. This division of the scene allows us to render only the part close to the observer twice. The remaining part of the scene is rendered only once and appropriately combined with the images generated for the left and the right eyes. To make the transition between the two parts of the scene imperceptible for the observer, we deduced a criterion for positioning the splitting plane depending on the resolution of the screen. We have shown that the proposed method enables the display of complex scenes in stereo mode up to 40% faster than the traditional method for stereo rendering.

In order to demonstrate that our method is applicable and that the human depth perception is not distorted applying our approach, we performed various usability studies. The latter have shown that the mixed (stereo/mono) display mode was indistinguishable from the pure stereo rendering and does not affect the interaction in the virtual environment.

In conclusion, we can state that the presented tools for visualization, interaction, and rendering in virtual environments enhance the usability of their original predecessors and offer new ways for rendering, analysis, and interactive exploration of the input data. The algorithms and techniques proposed in this work provide a powerful extension of existing tools, while overcoming many of the drawbacks of the traditional techniques. Moreover, we have shown that the presented concepts can be used as a basis for the development of various new interaction and exploration techniques.

List of Figures

List of Tables

Bibliography

[Agrawala et al., 1997] Agrawala, M., A. C. Beers, B. Fröhlich, P. Hanrahan, I. McDowall, and M. Bolas: 1997. The two-user responsive workbench: Support for collaboration through independent views of a shared space. In Whitted, T., editor, *SIGGRAPH 97 Conference Proceedings*, Annual Conference Series 327–332. ACM SIGGRAPH, Addison Wesley. ISBN 0-89791-896-7.

[Aliaga and Lastra, 1997] Aliaga, D. G. and A. A. Lastra: 1997. Architectural walkthroughs using portal textures. In Yagel, R. and H. Hagen, editors, *IEEE Visualization '97* 355–362. IEEE.

[Aliaga and Lastra, 1999] Aliaga, D. G. and A. A. Lastra: 1999. Automatic image placement to provide a guaranteed frame rate. In Rockwood, A., editor, *SIGGRAPH 99 Conference Proceedings* 307–316, N.Y. Addison Wesley.

[Angus and Sowizral, 1995] Angus, I. and H. Sowizral: 1995. Embedding the 2D interaction metaphor in a real 3D virtual environment. *Stereoscopic Displays and Virtual Reality Systems. Proceedings SPIE*, **2409**, 282–293.

[Badler et al., 1986] Badler, N. I., K. H. Manoochehri, and D. Baraff: 1986. Multidimensional input techniques and articulated figure positioning by multiple constraints. In *Proc. 1986 ACM Workshop on Interactive 3D Graphics* 151–170, Chapel Hill, NC.

[Bajura et al., 1992] Bajura, M., H. Fuchs, and R. Ohbuchi: 1992. Merging virtual objects with the real world: Seeing ultrasound imagery within the patient. *Computer Graphics*, **26**(2), 203–210.

[Band, 1986] Band, L. E.: 1986. Topographic partition of watersheds with digital elevation models. *Water Recources Res.*, **22**(1), 15–24.

[Beucher and Lantuéjoul, 1979] Beucher, S. and C. Lantuéjoul: 1979. Use of watersheds in contour detection. In *International Workshop on Image Processing*, Rennes. CCETT/IRISA.

[Beucher and Meyer, 1993] Beucher, S. and F. Meyer: 1993. The morphological approach to segmentation: the watershed transformation. In Dougherty, E. R., editor, *Mathematical Morphology in Image Processing*chapter 12, 433–481. Marcel Dekker, New York.

[Bier et al., 1993] Bier, E. A., M. C. Stone, K. Pier, W. Buxton, and T. D. DeRose: 1993. Toolglass and magic lenses: The see-through interface. In *SIGGRAPH 93 Conference Proceedings*, volume 27 73–80.

[Bier et al., 1994] Bier, E. A., M. C. Stone, K. Pier, K. Fishkin, T. Baudel, M. J. Conway, W. Buxton, and T. D. DeRose: 1994. Toolglass and magic lenses: The see-through interface. In *Proceedings of ACM CHI'94 Conference on Human Factors in Computing Systems*, volume 2 of *VIDEOS: Part II – Techniques for Improved Human-Computer Interaction* 445–446.

[Billinghurst et al., 1997] Billinghurst, M., S. Baldis, L. Matheson, and M. Philips: 1997. 3D palette: A virtual reality content creation tool. In *Proceedings of the ACM Symposium on Virtual Reality Software and Technology (VRST-97)* 155–156, New York. ACM Press.

[Bowman and Hodges, 1997] Bowman, D. A. and L. F. Hodges: 1997. An evaluation of techniques for grabbing and manipulating remote objects in immersive virtual environments. In Cohen, M. and D. Zeltzer, editors, *1997 Symposium on Interactive 3D Graphics* 35–38. ACM SIGGRAPH. ISBN 0-89791-884-3.

[Bowman et al., 1997] Bowman, D. A., D. Koller, and L. F. Hodges: 1997. Travel in immersive virtual environments: An evaluation of viewpoint motion control techniques. In *IEEE Proceedings of VRAIS'97* 45–52.

[Brooks, Jr., 1988] Brooks, Jr., F. P.: 1988. Grasping reality through illusion- interactive graphics serving science. In *Proceedings of ACM CHI 88 Conference on Human Factors in Computing Systems* 1–11.

[Brooks, Jr. et al., 1990] Brooks, Jr., F. P., M. Ouh-Young, J. J. Batter, and P. J. Kilpatrick: 1990. Project GROPE — haptic displays for scientific visualization. In Baskett, F., editor, *SIGGRAPH 90 Conference Proceedings*, volume 24 177–185.

[Buxton and Myers, 1986] Buxton, W. and B. A. Myers: 1986. A study in two-handed input. In *Proceedings of ACM CHI 86 Conference on Human Factors in Computing Systems* 321–326.

[Chang et al., 1999] Chang, C.-F., G. Bishop, and A. A. Lastra: 1999. LDI tree: a hierarchical representation for image-based rendering. *Computer Graphics*, **33**(Annual Conference Series), 291–298.

[Coquillart and Wesche, 1999] Coquillart, S. and G. Wesche: 1999. The virtual palette and the virtual remote control panel: A device and an interaction paradigm for the responsive workbench. In *Proceedings of the IEEE Virtual Reality* 213–216.

[Cruz-Neira et al., 1993] Cruz-Neira, C., D. J. Sandin, and T. A. DeFanti: 1993. Surround-screen projection-based virtual reality: The design and implementation of the CAVE. In Kajiya, J. T., editor, *Computer Graphics (SIGGRAPH '93 Proceedings)*, volume 27 135–142.

[Cutler et al., 1997] Cutler, L. D., B. Fröhlich, and P. Hanrahan: 1997. Two-handed direct manipulation on the responsive workbench. *Proceedings of SIGGRAPH Symposium on Interactive 3D Graphics '97, RI, USA* 39–43.

[Darken et al., 1997] Darken, R. P., W. R. Cockayne, and D. Carmein: 1997. The omnidirectional treadmill: A locomotion device for virtual worlds. In *Proceedings of the ACM Symposium on User Interface Software and Technology*, Blurring Physical and Virtual 213–221.

[Darken and Sibert, 1993] Darken, R. P. and J. L. Sibert: 1993. A toolset for navigation in virtual environments. In *Proceedings of the ACM Symposium on User Interface Software and Technology*, Virtual Reality 157–165.

[Darsa et al., 1998] Darsa, L., B. Costa, and A. Varshney: 1998. Walkthroughs of complex environments using image-based simplification. *Computers and Graphics*, **22**(1), 55–69.

[Decoret et al., 1999] Decoret, X., F. Sillion, G. Schaufler, and J. Dorsey: 1999. Multi-layered impostors for accelerated rendering. *Computer Graphics Forum*, **18**(3), 61–73. ISSN 1067-7055.

[Drucker and Zeltzer, 1994] Drucker, S. M. and D. Zeltzer: 1994. Intelligent camera control in a virtual environment. In *Proceedings of Graphics Interface '94* 190–199, Banff, Alberta, Canada. Canadian Information Processing Society.

[Ehricke et al., 1993] Ehricke, H.-H., G. Daiber, and W. Straßer: 1993. The vision camera: An interactive tool for volume data exploration and navigation. In Nielson, G. M. and D. Bergeron, editors, *Proceedings of the Visualization '93 Conference* 25–31, San Jose, CA. IEEE Computer Society Press.

[Encarnação, 1997] Encarnação, L. M.: 1997. *Concept and realisation of intelligent user support in interactive graphics applications.* PhD thesis, Eberhard-Karls-Universität Tübingen, Fakultät für Informatik.

[Farin, 1988] Farin, G.: 1988. *Curves and Surfaces for Computer Aided Geometric Design, A Practical Guide.* Academic Press, Inc. ISBN 0-12-249050-9.

[Fedak et al., 1996] Fedak, M. A., P. Lovell, and B. J. McConnell: 1996. MAMVIS: A marine mammal behaviour visualization system. *The Journal of Visualization and Computer Animation*, **7**(3), 141–147.

[Feiner et al., 1993] Feiner, S., B. MacIntyre, M. Haupt, and E. Solomon: 1993. Windows on the world: 2D windows for 3D augmented reality. In *Proceedings of the ACM Symposium on User Interface Software and Technology*, Virtual Reality 145–155.

[Galyean, 1995] Galyean, T. A.: 1995. Guided navigation of virtual environments. In Hanrahan, P. and J. Winget, editors, *1995 Symposium on Interactive 3D Graphics* 103–104. ACM SIGGRAPH. ISBN 0-89791-736-7.

[Gibson, 1998] Gibson, S. F. F.: 1998. Using distance maps for accurate surface reconstruction in sampled volumes. In *Proceedings of the 1998 Symposium on Volume Visualization (VOLVIS-98)* 23–30, New York. ACM Press.

[Gleicher and Witkin, 1992] Gleicher, M. and A. Witkin: 1992. Through-the-lens camera control. In Catmull, E. E., editor, *SIGGRAPH 92 Conference Proceedings*, volume 26 331–340.

[Guiard, 1988] Guiard, Y.: 1988. The kinematic chain as a model for human asymmetrical bimanual cooperation. In Colley, A. and J. Beech, editors, *Cognition and action in skilled behavior* 205–228. Amsterdam: North-Holland.

[Heguy et al., 2001] Heguy, O., N. Rodriguez, and H. Luga: 2001. Virtual environment for cooperative assistance in teleoperation. In Skala, V., editor, *Proceedings of 9-th International Conference on Computer Graphics, Visualization, and Interactive Digital Media (WSCG'2001).*

[Higgins and Ojard, 1993] Higgins, W. and E. Ojard: 1993. Interactive morphological watershed analysis for 3D medical images. *Computerized Medical Imaging and Graphics*, **17**(4/5), 387–395.

[Hirose et al., 2001] Hirose, M., K. Hirota, T. Ogi, H. Yano, N. Kakehi, M. Saito, and M. Nakashige: 2001. Hpaticgear: The development of a wearable force display system for immersive projection displays. In *Proceedings of the IEEE Virtual Reality* 123–129.

[Hodges, 1992] Hodges, L. F.: 1992. Tutorial: Time-multiplexed stereoscopic computer graphics. *IEEE Computer Graphics and Applications*, **12**(2), 20–30.

[Hodges et al., 1999] Hodges, L. F., B. O. Rothbaum, R. Alarcon, D. Ready, F. Shahar, K. Graap, J. Pair, P. Hebert, B. Wills, and D. Baltzell: 1999. Virtual vietnam: A virtual environment for the treatment of chronic post-traumatic stress disorder. *CyberPsychology & Behavior*, **2**(1), 1–9.

[Hoschek and Lasser, 1993] Hoschek, J. and D. Lasser: 1993. *Fundamentals of Computer Aided Geometric Design.* A. K. Peters. ISBN 1-56881-007-5.

[Iwate et al., 2001] Iwate, H., H. Yano, and F. Nakaizumi: 2001. Gait master: A versatile locomotion interface for uneven virtual terrain. In *Proceedings of the IEEE Virtual Reality* 131–137.

[Jackway, 1995] Jackway, P. T.: 1995. Morphological multiscale gradient watershed image analysis. In Borgefors, G., editor, *9th SCIA Scandinavian Conference on Image Analysis* 87–94.

[Jaswa, 1997] Jaswa, V.: 1997. CAVEvis: distributed real-time visualization of time-varying scalar and vector fields using the CAVE virtual reality theater. In *IEEE Visualization '97 (VIS '97)* 301–308, Washington - Brussels - Tokyo. IEEE.

[Kabbash et al., 1994] Kabbash, P., W. Buxton, and A. Sellen: 1994. Two-handed input in a compound task. In *Proceedings of ACM CHI'94 Conference on Human Factors in Computing Systems*, volume 2 of *PAPER ABSTRACTS: Evaluating Pointing Devices* 230.

[Lasseter, 1987] Lasseter, J.: 1987. Principles of traditional animation applied to 3D computer animation. In Stone, M. C., editor, *SIGGRAPH 90 Conference Proceedings*, volume 21 35–44.

[Lipton, 1997] Lipton, L.: 1997. Stereo3d handbook. Technical Report http://www.stereographics.com/html/whtpaprs.html, Stereo-Graphics Corp., San Rafael.

[Lorensen and Cline, 1987] Lorensen, W. E. and H. E. Cline: 1987. Marching cubes: a high resolution 3D surface construction algorithm. In Stone, M. C., editor, *SIGGRAPH '87 Conference Proceedings (Anaheim, CA, July 27–31, 1987)* 163–170. Computer Graphics, Volume 21, Number 4.

[Mangan and Whitaker, 1999] Mangan, A. P. and R. T. Whitaker: 1999. Partitioning 3D Susface Meshes Using Watershed Segmentation. *IEICE Transactions on Visualization and Computer Graphics*, **5**(4).

[Mapes and Moshell, 1995] Mapes, D. and J. Moshell: 1995. A two-handed interface for object manipulation in virtual environments. *Presence*, **4**(4), 403–416.

[McMillan, 1995] McMillan, L.: 1995. A list-priority rendering algorithm for redisplaying projected surfaces. Technical Report TR95-005, Department of Computer Science, University of North Carolina - Chapel Hill. Wed, 26 Jun 1996 18:10:53 GMT.

[Meißner et al., 1999] Meißner, M., D. Bartz, T. Hüttner, G. Müller, and J. Einighammer: 1999. Generation of Subdivision Hierarchies for Efficient Occlusion Culling of Large Polygonal Models. Technical Report WSI-99-13, Wilhelm Schickard Institut für Informatik, Graphische Interaktive Systeme.

[Meyer, 1995] Meyer, B.: 1995. *Object Success*. Prentice-Hall.

[Meyer, 1994] Meyer, F.: 1994. Topographic distance and watershed lines. *Signal Processing*, **38**(1), 113–125.

[Meyer and Beucher, 1990] Meyer, F. and S. Beucher: 1990. Morphological segmentation. *Journal of Visual Communication and Image Representation*, **1**(1), 21–46.

[Mine, 1995] Mine, M. R.: 1995. Virtual environment interaction techniques. Technical Report TR95-018, Department of Computer Science, University of North Carolina - Chapel Hill.

[Mine et al., 1997] Mine, M. R., F. P. Brooks, Jr., and C. H. Séquin: 1997. Moving objects in space: Exploiting proprioception in virtual-environment interaction. In Whitted, T., editor, *SIGGRAPH 97 Conference Proceedings*, Annual Conference Series 19–26. ACM SIGGRAPH, Addison Wesley. ISBN 0-89791-896-7.

[Moga et al., 1995] Moga, A. N., B. Cramariuc, and M. Gabbouj: 1995. A parallel watershed algorithm based on rainfalling simulation. In *European Conference on Circuit Theory and Design*, volume 1 339–342, Istanbul, Turkey.

[Moga and Gabbouj, 1997] Moga, A. N. and M. Gabbouj: 1997. Parallel image component labeling with watershed transformations. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **19**(5), 441–450.

[Najman and Schmitt, 1994] Najman, L. and M. Schmitt: 1994. Watershed of a continuous function. *Signal Processing*, **38**, 99–112.

[Najman and Schmitt, 1996] Najman, L. and M. Schmitt: 1996. Geodesic saliency of watershed contours and hierarchical segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **18**(12), 1163–1173.

[Olano et al., 1997] Olano, M., J. Cohen, M. R. Mine, and G. Bishop: 1997. Combatting Rendering Latency. In *Proceedings of the 1995 Symposium on Interactive 3D graphics* 19–24, Monterey, CA.

[Oliveira and Bishop, 1999] Oliveira, M. M. and G. Bishop: 1999. Image-based objects. In Spencer, S. N., editor, *Proceedings of the Conference on the 1999 Symposium on interactive 3D Graphics* 191–198, New York. ACM Press.

[Optimizer, 1997] Optimizer: 1997. *Optimizer Manual*. Silicon Graphics Inc., Mountain View, CA, USA.

[Padmos and Milders, 1992] Padmos, P. and M. V. Milders: 1992. Quality Criteria for Simulated Images: A Literature Review. *Human Factors*, **34**(6), 727–748.

[Palamidese, 1996] Palamidese, P.: 1996. A camera motion metaphor based on film grammar. *The Journal of Visualization and Computer Animation*, **7**(2), 61–78.

[Pausch, 1991] Pausch, R.: 1991. Virtual reality on five dollars a day. In *Proceedings of ACM CHI'91 Conference on Human Factors in Computing Systems*, Virtual Reality 265–270.

[Pausch et al., 1995] Pausch, R., T. Burnette, D. Brockway, and M. E. Weiblen: 1995. Navigation and locomotion in virtual worlds via flight into Hand-Held miniatures. In Cook, R., editor, *SIGGRAPH 95 Conference Proceedings*, Annual Conference Series 399–400. ACM SIGGRAPH, Addison Wesley.

[Pierce et al., 1997] Pierce, J. S., A. S. Forsberg, M. J. Conway, S. Hong, R. C. Zeleznik, and M. R. Mine: 1997. Image plane interaction techniques in 3D immersive environments. In Cohen, M. and D. Zeltzer, editors, *1997 Symposium on Interactive 3D Graphics* 39–44. ACM SIGGRAPH. ISBN 0-89791-884-3.

[Pierce et al., 1999] Pierce, J. S., B. C. Stearns, and R. Pausch: 1999. Voodoo dolls: Seamless interaction at multiple scales in virtual environments. In Spencer, S. N., editor, *Proceedings of the Conference on the 1999 Symposium on interactive 3D Graphics* 141–145, New York. ACM Press.

[Poupyrev et al., 1996] Poupyrev, I., M. Billinghurst, S. Weghorst, and T. Ichikawa: 1996. The go-go interaction technique: Non-linear mapping for direct manipulation in VR. In *Proceedings of the ACM Symposium on User Interface Software and Technology*, Papers: Virtual Reality (TechNote) 79–80.

[Pratt, 1991] Pratt, W. K.: 1991. *Digital Image Processing*. Wiley-Interscience Publication, New York / Chichester / Brisbane / Toronto / Singapore.

[Rafferty et al., 1997] Rafferty, M. M., D. G. Aliaga, and A. A. Lastra: 1997. 3D image warping in architecutral walkthroughs. Technical Report TR97-019, Department of Computer Science, University of North Carolina - Chapel Hill. Tue, 7 Oct 1997 21:49:24 GMT.

[Rafferty et al., 1998] Rafferty, M. M., D. G. Aliaga, V. Popescu, and A. A. Lastra: 1998. Images for accelerating architectural walkthroughs. *IEEE Computer Graphics & Applications*, **18**(6). ISSN 0272-1716.

[Rau et al., 1998] Rau, R. T., D. Weiskopf, and H. Ruder: 1998. Special relativity in virtual reality. In Hege, H.-C. and K. Polthier, editors, *Mathematical Visualization* 269–279. Springer Verlag, Heidelberg.

[Reitmayr and Schmalstieg, 2001] Reitmayr, G. and D. Schmalstieg: 2001. Open-Tracker - An Open Software Architecture for Reconfigurable Tracking based on XML. In *Proceedings of the IEEE Virtual Reality*.

[Rosenberg, 1993] Rosenberg, L. B.: 1993. The effect of interocular distance upon operator performance using stereoscopic displays to perform virtual depth tasks. *IEEE Proceedings of Virtual Reality Annual International Sysmposium* 27–32.

[Rothbaum and Hodges, 1999] Rothbaum, B. O. and L. F. Hodges: 1999. The use of virtual reality exposure in the treatment of anxiety disorders. *Behavior Modification*, **23**(4), 507–526.

[Rothbaum et al., 2000] Rothbaum, B. O., L. F. Hodges, and S. Smith: 2000. Virtual reality exposure therapy abbreviated treatment manual: Fear of flying application. *Cognitive and Behavioral Practice*.

[Sachs et al., 1991] Sachs, E., A. Roberts, and D. Stoops: 1991. 3-draw: A tool for designing 3D shapes. *IEEE Computer Graphics and Applications*, **11**(6), 18–26.

[Schaufler, 1998] Schaufler, G.: 1998. Per-object image warping with layered impostors. In Drettakis, G. and N. Max, editors, *Rendering Techniques '98*, Eurographics 145–156. Springer-Verlag Wien New York.

[Schmalstieg et al., 1999] Schmalstieg, D., L. M. Encarnação, and Z. Szalavári: 1999. Using transparent props for interaction with the virtual table (color plate S. 232). In Spencer, S. N., editor, *Proceedings of the Conference on the 1999 Symposium on Interactive 3D Graphics* 147–154, New York. ACM Press.

[Schmalstieg et al., 1996] Schmalstieg, D., A. L. Fuhrmann, M. Gervautz, and Z. Szalavári: 1996. 'Studierstube' - An Environment for Collaboration in Augmented Reality'. In *Proceedings of Collaborative Virtual Environments '96, Nottingham, UK, Sep. 19-20*.

[Schmalstieg and Schaufler, 1999] Schmalstieg, D. and G. Schaufler: 1999. Sewing worlds together with SEAMS: A mechanism to construct complex virtual environments. *Presence - Teleoperators and Virtual Environments*, **8**(4), 449–461.

[Shade et al., 1998] Shade, J. W., S. J. Gortler, L. He, and R. Szeliski: 1998. Layered depth images. In Cohen, M., editor, *SIGGRAPH 98 Conference Proceedings*, Annual Conference Series 231–242. ACM SIGGRAPH, Addison Wesley. ISBN 0-89791-999-8.

[Sijbers et al., 1997] Sijbers, J., P. Scheunders, M. Verhoye, A. V. der Linden, D. V. Dyck, and E. Raman: 1997. Watershed-based segmentation of 3D MR data for volume quantization. *Magnetic Resonance Imaging*, **15**(4).

[Southard, 1995] Southard, D. A.: 1995. Viewing Model for Virtual Environment Displays. *Journal of Electronic Imaging*, **4**(4), 413–420.

[Stoakley et al., 1995] Stoakley, R., M. J. Conway, and R. Pausch: 1995. Virtual reality on a WIM: Interactive worlds in miniature. In *Proceedings of ACM CHI'95 Conference on Human Factors in Computing Systems* 265–272.

[Stoev, 2000] Stoev, S. L.: 2000. Rafsi - a Fast Watershed Algorithm Based on Rainfalling Simulation. In *Proceedings of 8-th International Conference on Computer Graphics, Visualization, and Interactive Digital Media (WSCG'2000)*, volume 1 100–107, Pilzen.

[Stoev et al., 2001a] Stoev, S. L., M. Feurer, M. Ruckaberle, and W. Straßer: 2001a. Exploring the Past: a Toolset for Visualization of Historical Events in Virtual Environments. In *Proceedings of the ACM Symposium on Virtual Reality Software and Technology (VRST-2001)* 63–70. ACM Press.

[Stoev et al., 2000a] Stoev, S. L., T. Hüttner, and W. Straßer: 2000a. Accelerated Rendering in Stereo-Based Projections. In *Proc. 3. International Conference on Collaborative Virtual Environments (ACM CVE2000)* 213–214. ACM.

[Stoev et al., 2001b] Stoev, S. L., T. Hüttner, and W. Straßer: 2001b. Mixed Virtuality - A Way to Create a Better Multi-User Virtual Environment. Technical Report WSI-2001-10, University of Tübingen, (WSI/GRIS).

[Stoev et al., 2000b] Stoev, S. L., I. Peter, and W. Straßer: 2000b. The Multi-LDI: an Image-Based Rendering Approach for Interaction, Navigation, and Visualization in Complex Virtual Environments. Technical Report WSI-2000-23, University of Tübingen, (WSI/GRIS).

[Stoev et al., 2001c] Stoev, S. L., I. Peter, and W. Straßer: 2001c. Interaction, Navigation, and Visualization Props in Complex Virtual Environments Using Image Based Rendering Techniques. In *Proceedings of the IEEE Virtual Reality*.

[Stoev et al., 2000c] Stoev, S. L., D. Schmalstieg, and W. Straßer: 2000c. Through-The-Lens Techniques for Motion, Navigation, and Remote Object Manipulation in Immersive Virtual Environments. Technical Report WSI-2000-22, University of Tübingen, (WSI/GRIS).

[Stoev et al., 2001d] Stoev, S. L., D. Schmalstieg, and W. Straßer: 2001d. Manipulate the Unreachable: Through-The-Lens Remote Object Manipulation in Virtual Environments. In *SIGGRAPH 2001 Conference Abstracts and Applications*.

[Stoev et al., 2001e] Stoev, S. L., D. Schmalstieg, and W. Straßer: 2001e. Two-Handed Through-The-Lens-Techniques for Navigation in Virtual Environments. In *Proceedings of the Eurographics Workshop on Virtual Environments*.

[Stoev et al., 2002] Stoev, S. L., D. Schmalstieg, and W. Straßer: 2002. The Through-The-Lens Metaphor: Taxonomy and Application. In *Proceedings of the IEEE Virtual Reality*.

[Stoev and Straßer, 2000] Stoev, S. L. and W. Straßer: 2000. Extracting Regions of Interest Applying a Local Watershed Transformation. In *Proceedings of the 11th Annual IEEE Conference on Visualization (VIS-2000)* 21–28. ACM Press.

[Stoev and Straßer, 2001] Stoev, S. L. and W. Straßer: 2001. A Case Study on Interactive Exploration and Guidance Aids for Visualizing Historical Data. In *Proceedings of the 12th Annual IEEE Conference on Visualization (VIS-2001)* 485–488. ACM Press.

[Stroustrup, 1997] Stroustrup, B.: 1997. *The C++ Programming Language: Third Edition.* Addison-Wesley Publishing Co., Reading, Mass.

[Surdick et al., 1997] Surdick, R. T., R. A. K. Elizabeth T. Davis, and L. F. Hodges: 1997. The perception of distance in simulated displays. *Presence*, **6**(5), 513–531.

[Sutherland, 1968] Sutherland, I. E.: 1968. A head-mounted three dimensional display. In *Proc. Fall Joint Computer Conf.*, volume 33 757–764. American Federation of Information Processing Societies, AFIPS, Thompson Books.

[Szalavári and Gervautz, 1997] Szalavári, Z. and M. Gervautz: 1997. The personal interaction panel - A two handed interface for augmented reality. *Computer Graphics Forum (Proceedings of EUROGRAPHICS'97)*, **16**(3), 335–346.

[Tiede et al., 1998] Tiede, U., T. Schiemann, and K. H. Höhne: 1998. High quality rendering of attributed volume data (color plate p. 541). In *Proceedings of the 9th Annual IEEE Conference on Visualization (VIS-98)* 255–263, New York. ACM Press.

[Turner et al., 1996] Turner, R., E. Gobbetti, and I. Soboroff: 1996. Head-tracked stereo viewing with two-handed 3D interactionfor animated character construction. *Computer Graphics Forum*, **15**(3), 197–206. Proceedings of Eurographics '96. ISSN 1067-7055.

[van Dam et al., 2000] van Dam, A., A. S. Forsberg, D. H. Laidlaw, J. J. LaViola, Jr., and R. M. Simpson: 2000. Immersive VR for scientific visualization: A progress report. *IEEE Computer Graphics and Applications*, **20**(6), 26–52.

[Vepstas, 2000] Vepstas, L.: 2000. Gle tubing and extrusion library, http://linas.org/gle/index.html.

[Viega et al., 1996] Viega, J., M. J. Conway, G. Williams, and R. Pausch: 1996. 3D magic lenses. In *Proceedings of the ACM Symposium on User Interface Software and Technology*, Papers: Information Visualization 51–58.

[Vincent and Soille, 1991] Vincent, L. and P. Soille: 1991. Watersheds in digital spaces: An efficient algorithm based on immersion simulations. *IEEE PAMI, 1991*, **13**(6), 583–598.

[Wann et al., 1995] Wann, J., S. Rushton, and M. Mon-Williams: 1995. Natural problems for stereoscopic depth perception in virtual environment. *Vision Research*, **35**(19), 2731–2736.

[Ware and Jessome, 1988] Ware, C. and D. R. Jessome: 1988. Using the BAT: a six-dimensional mouse for object placement. *IEEE Computer Graphics and Applications*, **8**(6), 65–70.

[Ware and Osborne, 1990] Ware, C. and S. Osborne: 1990. Exploration and virtual camera control in virtual three dimensional environments. In *Proceedings of the 1990 Symposium on Interactive 3D Graphics, Special Issue of Computer Graphics, Vol. 24* 175–183.

[Wartell et al., 1999] Wartell, Z., L. F. Hodges, and W. Ribarsky: 1999. Balancing fusion, image depth and distortion in stereoscopic head-tracked displays. *Computer Graphics*, **33**(Annual Conference Series), 351–358.

[Wegner et al., 1995] Wegner, S., T. Harms, J. H. Builtjes, and H. Oswald: 1995. The watershed transformation for multiresolution image segmentation. *Lecture Notes in Computer Science*, **974**, 31–37.

[Weiskopf, 2000] Weiskopf, D.: 2000. An immersive virtual environment for special relativity. In *WSCG 2000 Conference Proceedings.*

[Weiss and Jessel, 1998] Weiss, P. and A. S. Jessel: 1998. Virtual reality applications to work. *WORK*, **11**(3), 277–293.

[Wernecke, 1994] Wernecke, J.: 1994. *The Inventor Mentor*. Addison-Wesley, Reading, Massachusetts, U.S.A.

[Wernecke, 1999] Wernecke, J.: 1999. *The Inventor Mentor*. Addison-Wesley, Reading, Massachusetts, U.S.A.

[Wimmer et al., 1999a] Wimmer, M., M. Giegl, and D. Schmalstieg: 1999a. Fast walkthroughs with image caches and ray casting. *Computers and Graphics*, **23**(6), 831–838.

[Wimmer et al., 1999b] Wimmer, M., M. Giegl, and D. Schmalstieg: 1999b. Fast walkthroughs with image caches and ray casting. In Gervaut, M., D. Schmalstieg, and A. Hildebrand, editors, *Virtual Environments '99. Proceedings of the Eurographics Workshop in Vienna, Austria* 73–84. Springer-Verlag Wien.

[Wloka and Greenfield, 1995] Wloka, M. M. and E. Greenfield: 1995. The virtual tricorder. Technical Report CS-95-05, Department of Computer Science, Brown University. Sun, 13 Jul 1997 18:30:14 GMT.

[Wonka and Schmalstieg, 1999] Wonka, P. and D. Schmalstieg: 1999. Occluder shadows for fast wakthroughs of urban environments. In Seidel, H.-P. and S. Coquillart, editors, *Computer Graphics Forum*, volume 18 51–60. Eurographics Association.

[Woo et al., 1998] Woo, M., J. Neider, and T. Davis: 1998. *OpenGL Programming Guide*. Addison-Wesley, Reading, Massachusetts, U.S.A.

[Yeh and Silverstein, 1990] Yeh, Y.-Y. and L. D. Silverstein: 1990. Limits of fusion and depth judgement in stereoscopic color displays. *Human Factors*, **32**(2), 45–60.

[Yeh and Silverstein, 1992] Yeh, Y.-Y. and L. D. Silverstein: 1992. Spatial judgments with monoscopic and stereoscopic presentation of perspective displays. *Human Factors*, **34**(5), 583–600.

Author Index

Lebens- und Bildungsgang

24. Januar 1974	geboren in Sofia, Bulgarien
1980 - 1985	Grundschule, Sofia, Bulgarien
1985 - 1987	Gymnasium, Sofia
1987 - 1991	Deutsches Fremdsprachengymnasium, Sofia
1991 - 1992	Studienkolleg in Heidelberg, Abschluß Abitur
1992 - 1997	Studium der Informatik an der Eberhard-Karls-Universität Tübingen
1997	Diplomarbeit am Lehrstuhl für Graphisch-Interaktive Systeme am Wilhelm-Schickard-Institut für Informatik der Eberhard-Karls-Universität Tübingen Prof. Straßer
seit 1997	Wissenschaftlicher Mitarbeiter am Lehrstuhl für Graphisch Interaktive Systeme am Wilhelm-Schickard-Institut für Informatik der Eberhard-Karls-Universität Tübingen (Prof. Straßer)
1994 - 1997	Studentische Hilfskraft am Lehrstuhl für Graphisch-Interaktive Systeme am WSI für Informatik der Eberhard-Karls-Universität Tübingen (Prof. Straßer)
1995 - 1997	Übungsgruppenleiter (Informatik 1 und 2) am WSI für Informatik der Eberhard-Karls-Universität Tübingen (Prof. Loos und Prof. Küchlin)